ECOLE
POLYTECHNIQUE
DE BRUXELLES

Université Libre de Bruxelles
Ecole Polytechnique de Bruxelles
CODE - Computers and Decision Engineering
IRIDIA - Institut de Recherches Interdisciplinaires
et de Développements en Intelligence Artificielle

Population-based Heuristic Algorithms for Continuous and Mixed Discrete-Continuous Optimization Problems

Tianjun LIAO

Promoteur:
Prof. Marco DORIGO

Co-promoteur:
Dr. Thomas STÜTZLE

Thése présentée à la Ecole Polytechnique de Bruxelles de l'Université Libre de Bruxelles en vue de l'obtention du titre de Docteur en Sciences de l'Ingenieur.

Année Académique 2012-2013

Summary

Continuous optimization problems are optimization problems where all variables have a domain that typically is a subset of the real numbers; mixed discrete-continuous optimization problems have additionally other types of variables, so that some variables are continuous and others are on an ordinal or categorical scale. Continuous and mixed discrete-continuous problems have a wide range of applications in disciplines such as computer science, mechanical or electrical engineering, economics and bioinformatics. These problems are also often hard to solve due to their inherent difficulties such as a large number of variables, many local optima or other factors making problems hard. Therefore, in this thesis our focus is on the design, engineering and configuration of high-performing heuristic optimization algorithms.

We tackle continuous and mixed discrete-continuous optimization problems with two classes of population-based heuristic algorithms, ant colony optimization (ACO) algorithms and evolution strategies. In a nutshell, the main contributions of this thesis are that (i) we advance the design and engineering of ACO algorithms to algorithms that are competitive or superior to recent state-of-the-art algorithms for continuous and mixed discrete-continuous optimization problems, (ii) we improve upon a specific state-of-the-art evolution strategy, the covariance matrix adaptation evolution strategy (CMA-ES), and (iii) we extend CMA-ES to tackle mixed discrete-continuous optimization problems.

More in detail, we propose a unified ant colony optimization (ACO) framework for continuous optimization (UACOR). This framework synthesizes algorithmic components of two ACO algorithms that have been proposed in the literature and an incremental ACO algorithm with local search for continuous optimization, which we have proposed during my doctoral research. The design of UACOR allows the usage of automatic algorithm configuration techniques to automatically derive new, high-performing ACO algorithms for continuous optimization. We also propose iCMAES-ILS, a hybrid algorithm that loosely couples IPOP-CMA-ES, a CMA-ES variant that uses a restart schema coupled with an increasing population

size, and a new iterated local search (ILS) algorithm for continuous optimization. The hybrid algorithm consists of an initial competition phase, in which IPOP-CMA-ES and the ILS algorithm compete for further deployment during a second phase. A cooperative aspect of the hybrid algorithm is implemented in the form of some limited information exchange from IPOP-CMA-ES to the ILS algorithm during the initial phase. Experimental studies on recent benchmark functions suites show that UACOR and iCMAES-ILS are competitive or superior to other state-of-the-art algorithms.

To tackle mixed discrete-continuous optimization problems, we extend $\text{ACO}_{\mathbf{MV}}$ and propose $\text{CES}_{\mathbf{MV}}$, an ant colony optimization algorithm and a covariance matrix adaptation evolution strategy, respectively. In $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$, the decision variables of an optimization problem can be declared as continuous, ordinal, or categorical, which allows the algorithm to treat them adequately. $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ include three solution generation mechanisms: a continuous optimization mechanism, a continuous relaxation mechanism for ordinal variables, and a categorical optimization mechanism for categorical variables. Together, these mechanisms allow $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ to tackle mixed variable optimization problems. We also propose a set of artificial, mixed-variable benchmark functions, which can simulate discrete variables as ordered or categorical. We use them to automatically tune $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$'s parameters and benchmark their performance. Finally we test $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ on various real-world continuous and mixed-variable engineering optimization problems. Comparisons with results from the literature demonstrate the effectiveness and robustness of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ on mixed-variable optimization problems.

Apart from these main contributions, during my doctoral research I have accomplished a number of additional contributions, which concern (i) a note on the bound constraints handling for the CEC'05 benchmark set, (ii) computational results for an automatically tuned IPOP-CMA-ES on the CEC'05 benchmark set and (iii) a study of artificial bee colonies for continuous optimization. These additional contributions are to be found in the appendix to this thesis.

Acknowledgements

I express my deep gratitude to my supervisors, Prof. Marco Dorigo and Dr. Thomas Stützle for giving me the chance to do research at IRIDIA. I thank them for their supervision that had a great influence not only on this thesis but also on my attitude to work and life. They are very professional. I feel very lucky to work with them. They are deeply engraved in my mind.

I first contacted Marco by email on January 6, 2009. He swiftly replied me with interest. From then on, he fully supported me in many aspects including invitation letter, fellowship, inscription and so on. He was patient and always efficiently replied me in everything I proposed to him. Finally, on November 11, 2009, I started my PhD research in IRIDIA. Marco is not only the "rider on a swarm" as reported by magazine *The Economist* but also a very nice PhD supervisor. I like his smile, his straightforward and serious attitude to work, and especially his clear-cut, short sentences. I am proud of him. I greatly appreciate his careful proof reading of my articles and thesis. We cooperate well in many aspects with a tacit mutual understanding.

I thank my co-supervisor, Dr. Thomas Stützle, a super kind person. He plays a crucial role in my research. His charisma and his spirit of scientific research deeply affected me. He is a fantastic scientist and PhD supervisor. In the field of heuristic optimization, his knowledge is just like the ocean, never a rim. He soaks up new subjects as a sponge soaks up water. His comments are always very helpful. He delivers positive energy and passion. His passion, expertise and critical thought promoted each of my research activities. We work together almost every day, on each of my research articles. We exchange opinions and share what we like and what we do not. I enjoy much working with him. In fact, he is more like my colleague and friend. As writing here, my mind has flashed back to many scenes of the past. I do not want to continue because those words sound that I am going to leave. In fact, I expect working longer with him, even a life time.

I express my appreciation to Prof. Yuejin Tan, Associate Prof. Kewei Yang and their related departments. They gave me the chance to study abroad and helped

me to apply for a fellowship from the China Scholarship Council.

Special thanks to the co-authors in my publications and fellow IRIDIAns. All of them, Anthony Antoun, Doğan Aydın, Prasanna Balaprakash, Hugues Bersini, Leonardo Bezerra, Stefano Benedettini, Saifullah bin Hussin, Mauro Birattari, Manuele Brambilla, Arne Brutschy, Alexandre Campo, Sara Ceschia, Muriel Decreton, Antal Decugniére, Jérémie Dubois-Lacoste, Eliseo Ferrante, Gianpiero Francesca, Matteo Gagliolo, Lorenzo Garattoni , Kiyohiko Hattori, Stefanie Kritzinger, Benjamin Lacroix, Manuel López-Ibáñez, Renaud Lenne, Dhananjay Ipparthi, Bruno Marchal, Franco Mascia, Nithin Mathews, Marie-Éléonore Marmion, Roman Miletitch, Marco A. Montes de Oca, Daniel Molina, Prospero C. Naval, Rehan O'Grady, Sabrina Oliveira, Michele Pace, Paola Pellegrini, Leslie Pérez Cáceres, Carlo Pinciroli, Giovanni Pini, Carlotta Piscopo, Gaëtan Podevijn, Andreagiovanni Reina, Andrea Roli, Francesco Sambo, Francisco C. Santos, Alexander Scheidler, Krzysztof Socha, Touraj Soleymani, Alessandro Stranieri, Wenjie Sun, Vito Trianni, Roberto Tavares, Ali Emre Turgut, Gabriele Valentini and Zhi Yuan helped me in different ways throughout these years. I also thank Prof. Hugues Bersini, co-director of IRIDIA with Prof. Marco Dorigo, for making IRIDIA such an enjoyable research lab. I also want to thank Prof. Marc Schoenauer from INRIA Saclay-Île-de-France, Prof. Bernard Fortz, Prof. Hugues Bersini, Dr. Mauro Birattari and Dr. Manuel López-Ibáñez from Université Libre de Bruxelles for their useful comments.

My special appreciation goes to my parents, Yingmin Liao and Xiaoou Xu, and Miss Jinyu Zhang. I thank very much for their love and care. I also thank all my dear friends. They gave me very much support during the time I lived in Brussels.

This work was supported by the E-SWARM project, *Engineering Swarm Intelligence Systems*, funded by the European Union's Seventh Framework Programme (FP7/2007-2013) / ERC grant agreement n^{o} 246939 and by the Meta-X project, *Metaheuristics for Complex Optimization Problems*, funded by the Scientific Research Directorate of the French Community of Belgium. Tianjun Liao acknowledges a fellowship from the China Scholarship Council.

Tianjun Liao
June 18th, 2013
Brussels, Belgium.

Contents

Chapter 1

Introduction

Continuous and mixed discrete-continuous optimization problems arise in many real-world optimization tasks in many areas such as computer science, mechanical or electrical engineering, economy and bioinformatics where solution improvement potentially leads to considerable benefit. Consider a few examples of such problems arising in engineering. A wind turbine may consist of two, three, four, or even more blades. The shape of each blade itself can be described by a set of continuous variables that describe length, thickness, curvatures, etc. In addition, different material compositions for the blades are available. Hence, such a problem may have continuous, integer, but also ordinal or categorical variables that need to be set appropriately to optimize one or several aspects of performance. Another example is the design of an aircraft system that consists of many engineering components. The coordinate and shape of each mechanical component itself can be described by a set of continuous or ordinal variables. Different material compositions, structural models and even high-level conceptual models can be described by a set of categorical variables. The values of these variables need to be optimized for one or several aspects of performance of an aircraft system.

Continuous optimization problems are optimization problems where all variables have a domain that typically is a subset of the real numbers; mixed discrete-continuous optimization problems have additionally other types of variables, so that some variables are continuous and others can be ordinal or categorical. These problems are often hard to solve. In the case of continuous variables, the search space contains an infinite number of solutions. The search space may involve complex landscape properties due to nonlinear objective functions; objective functions where derivatives may not be easily computable; correlated variables; and objective functions may have multiple local optima. Often, problems are black-box problems in the sense that there is no available explicit mathematical formulation. For example, this is always the case when the performance associated to specific vari-

able settings has to be estimated by simulations. Due to these difficulties inherent to these problems and to these problems' importance in the real world, the design and configuration of high-performing continuous and mixed discrete-continuous optimization algorithms is a highly active research area.

Algorithms for solving continuous optimization problems include analytical methods and approximation algorithms. Analytical methods guarantee to find an exact globally or locally optimal solution and can theoretically provide the certification of the optimality of the solution if enough time is permitted [Andréasson et al., 2005, Griva et al., 2009]. These methods require either exhaustive search or symbolic mathematical computation. They often turn out to be impractical for complex real-world optimization problems. Approximation algorithms are algorithms that are used to find approximate solutions to optimization problems, such as derivative-based approximation methods [Stoer et al., 1993], direct search methods [Conn et al., 2009], and heuristic algorithms [Hoos and Stützle, 2005]. Derivative-based approximation methods (e.g., steepest descent [Battiti, 1992], conjugate gradient method [Stoer et al., 1993] and Newton's method [Peitgen, 1989]) require the computation of numerical objective function values from the search space and derivative information of the function being optimized. Direct search methods such as downhill simplex method [Nelder and Mead, 1965], Powell's method [Powell, 1964] and pattern search [Torczon, 1997]) do not require derivative information. Derivative-based approximation methods and direct search methods are typically developed in mathematical programming.

In this thesis, we focus on heuristic algorithms [Hoos and Stützle, 2005], an important class of approximation algorithms. Heuristic algorithms are used to quickly and heuristically find satisfactory solutions. The higher level strategies to guide and improve subordinate heuristic methods are also called metaheuristics. Many metaheuristics have been proposed so far, in particular for tackling continuous optimization problems. They include Genetic Algorithms (GA) [Goldberg, 1989], Evolution Strategies (ESs) [Beyer and Schwefel, 2002, Hansen and Ostermeier, 2001], Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995], Differential Evolution (DE) [Storn and Price, 1997], Ant Colony Optimization (ACO) [Dorigo and Di Caro, 1999, Dorigo and Stützle, 2004, Socha and Dorigo, 2008] and Artificial Bee Colony (ABC) [Karaboga and Basturk, 2007]. Several of these methods such as PSO, DE, or ABC have originally been proposed for tackling continuous optimization problems. However, also most other metaheuristic methods have been adapted for solving continuous optimization problems and the design and configuration of high-performing heuristic algorithms for this task is

one of the most active areas in optimization.

Building further on continuous optimization, mixed discrete-continuous optimization come forth. This is often a harder task because both continuous and discrete variables have to be considered in the optimization process. Mixed integer programming (linear or nonlinear) refers to mathematical programming with continuous and integer variables with linear or nonlinear objective function and/or constraints [Bussieck and Pruessner, 2003]. In this thesis, we consider mixed discrete-continuous optimization problems where the discrete variables can be ordinal or categorical. Ordinal variables exhibit a natural ordering relation (e.g., $\{\text{small}, \text{medium}, \text{large}\}$). Categorical variables take their values from a finite set of categories [Abramson et al., 2009], which often identify non-numeric elements of an unordered set (e.g., colors, shapes or types of material). Categorical variables do not have a natural ordering relation and therefore require the use of specific algorithm techniques for handling them. To the best of our knowledge, the approaches to mixed discrete-continuous problems available in the literature are targeted to either handle mixtures of continuous and ordinal variables or mixtures of continuous and categorical variables. In other words, they do not consider the possibility that the formulation of a problem may involve at the same time the three types of variables. Hence, there is a need for algorithms that allow the explicit declaration of each variable as either continuous, ordinal or categorical. Note that mixed discrete-continuous optimization does not enjoy, despite its high practical relevance, such a great popularity as continuous optimization and therefore fewer algorithms for handling these problems are available.

1.1 Goal and methodology

The main goals of this thesis are

- to design and configure high performing algorithms for continuous optimization; and
- to design and configure high performing algorithms for mixed discrete-continuous optimization that can explicitly declare each variable as either continuous, ordinal or categorical and treat them adequately.

Instead of inventing completely new algorithms or new metaphors for optimization, in this thesis we focus on the enhancement of one new and promising method, ant colony optimization algorithm for continuous optimization [Socha and Dorigo,

2008] and another, already well-established method, covariance matrix adaptation evolution strategy [Hansen and Ostermeier, 2001]. We show how, by systematically engineering new algorithmic variants of these methods and by exploiting systematically automatic algorithm configuration tools, we can improve significantly their performance and apply them to new domains.

1.2 Main contributions

In this section, we highlight the main contributions of this thesis, which can be seen as a systematic engineering of high-performance, population-based heuristic algorithms for continuous and mixed discrete-continuous optimization problems.

A unified ant colony optimization framework for continuous optimization (UACOR)

We first studied the most popular ACO algorithm for continuous domains proposed by Socha and Dorigo [2008], called $ACO_{\mathbb{R}}$. In this thesis, we describe the UACOR framework. It combines algorithmic components from three existing algorithms, $ACO_{\mathbb{R}}$, another recent ACO algorithm, $DACO_{\mathbb{R}}$, proposed by Leguizamón and Coello [2010], and an incremental ACO algorithm with local search for continuous optimization ($IACO_{\mathbb{R}}$-LS), which we recently proposed ourselves[1]. UACOR can be seen as an algorithmic framework for continuous ACO algorithms from which earlier continuous ACO algorithms can be instantiated by using specific combinations of the available algorithmic components and parameter settings. The design of UACOR also allows the usage of automatic algorithm configuration techniques to automatically derive new ACO algorithms for continuous optimization. We use `irace` [López-Ibáñez et al., 2011] to automatically configure two new ACO algorithms. Their competitive performance to other state-of-the-art algorithms shows the high potential ACO algorithms have for continuous optimization and the high potential automatic algorithm configuration techniques have for the development of continuous optimizers from

[1] $IACO_{\mathbb{R}}$-LS [Liao et al., 2011b] is a significant algorithm component that contributes to UACOR. It is a variant of $ACO_{\mathbb{R}}$ that uses local search and that features a growing solution archive. I proposed $IACO_{\mathbb{R}}$-LS during my doctoral research and the paper describing $IACO_{\mathbb{R}}$-LS received **the best paper award** of the ACO-SI track at GECCO 2011. While $IACO_{\mathbb{R}}$-LS is actually an original contribution that was obtained during the initial stages of my doctoral research, it is now superseded by the UACOR framework from which also the original $IACO_{\mathbb{R}}$-LS can be instantiated. Therefore, I decided to not present $IACO_{\mathbb{R}}$-LS in very detail in this thesis but to directly focus on the presentation and the experimental analysis of the UACOR framework.

algorithm components. The UACOR framework is presented in Chapter 3.

Enhancing CMA-ES: A cooperative-competitive hybrid algorithm

We investigate evolution strategies and, in particular, the covariance matrix adaptation evolution strategy (CMA-ES) [Hansen and Ostermeier, 1996, 2001, Hansen et al., 2003], which is an established state-of-the-art algorithm for continuous optimization. A CMA-ES variant that uses a restart schema coupled with an increasing population size, called IPOP-CMA-ES, was the best performing algorithm on the CEC'05 benchmark set for continuous function optimization [Auger and Hansen, 2005, Suganthan et al., 2005]. This CEC'05 benchmark function set and some of the algorithms proposed for it play an important role in the assessment of the state of the art in continuous optimization. We propose iCMAES-ILS, which is a structurally simple, hybrid algorithm that loosely couples IPOP-CMA-ES with an iterated local search (ILS) algorithm. The hybrid iCMAES-ILS algorithm consists of an initial competition phase, in which IPOP-CMA-ES and the ILS algorithm compete for further execution in the deployment phase, where only one of the two algorithms is run until the budget is exhausted. The initial competition phase features also a cooperative aspect between IPOP-CMA-ES and the ILS algorithm.

We compare iCMAES-ILS to its component algorithms, IPOP-CMA-ES and ILS, and also to a number of alternative hybrids we propose between IPOP-CMA-ES and ILS. These hybrids are based on using algorithm portfolios, interleaving the execution of IPOP-CMA-ES and ILS, and using ILS as an improvement method between restarts of IPOP-CMA-ES. These comparisons indicate that iCMAES-ILS reaches statistically significantly better performance than almost all competitors and, thus, establishes our hybrid design as the most performing one. Additional comparisons to two state-of-the-art CMA-ES hybrid algorithms (MA-LSCh-CMA [Molina et al., 2010a] and PS-CMA-ES [Müller et al., 2009]) further show statistically significantly better performance of iCMAES-ILS over these competitors. We use `irace` [López-Ibáñez et al., 2011] to automatically tune iCMAES-ILS to obtain further performance improvements. Overall, these experimental results establish iCMAES-ILS as a new state-of-the-art algorithm for continuous optimization. iCMAES-ILS is presented in Chapter 4.

A set of artificial, mixed discrete-continuous benchmark functions

Many benchmark instances for continuous optimization problems are available, but this is not the case for mixed discrete-continuous optimization problems.

Therefore, we propose a new set of artificial, mixed discrete-continuous benchmark functions, which can simulate discrete variables as ordered or categorical. The artificial mixed-variable benchmark functions have characteristics such as non-separability, ill-conditioning and multi-modality. These benchmark functions provide a flexible environment for investigating the performance of mixed discrete-continuous optimization algorithms and the effect of different parameter settings on their performance. They are very useful as a training set for deriving high-performance parameter settings through the usage of automatic configuration methods.

Ant colony optimization for mixed discrete-continuous optimization problems ($\mathbf{ACO_{MV}}$)

We present $\mathrm{ACO_{MV}}$, an ACO algorithm to tackle mixed-variable optimization problems. The original $\mathrm{ACO_{MV}}$ algorithm was proposed by Socha in his PhD thesis. Starting from this initial work of Socha, we re-implemented the $\mathrm{ACO_{MV}}$ algorithm in C++, which reduced strongly the computation time when compared to the original implementation in R. We also refined the original $\mathrm{ACO_{MV}}$ algorithm and added a restart operator. We used subset of the artificial, mixed discrete-continuous benchmark functions we have proposed in this thesis for a detailed study of specific aspects of $\mathrm{ACO_{MV}}$ and to derive high-performance parameter settings through automatic algorithm configuration. In addition, also methodological improvements were made. While in the original work of Socha $\mathrm{ACO_{MV}}$ was specifically tuned on each engineering test problem, now one single parameter setting, obtained by a tuning process on the new benchmark functions (which are independent of the engineering test functions), is applied for the final test on the mixed-variable engineering problems. Finally, a further contribution is that we strongly extended the test bed of engineering benchmark functions that were originally considered by Socha.

We used `irace` [López-Ibáñez et al., 2011] to automatically tune the parameters of $\mathrm{ACO_{MV}}$. We applied $\mathrm{ACO_{MV}}$ to eight mixed discrete-continuous engineering optimization problems. Experimental results showed that $\mathrm{ACO_{MV}}$ reaches a very high performance: it improves over the best known solutions for two of the eight engineering problems, and in the remaining six it finds the best-known solutions using fewer objective function evaluations than most algorithms from the literature. $\mathrm{ACO_{MV}}$ is presented in Chapter 5.

Covariance matrix adaptation evolution strategy for mixed discrete-

continuous optimization problems ($\mathbf{CES_{MV}}$)

We propose $\mathrm{CES}_{\mathbf{MV}}$, a covariance matrix adaptation evolution strategy for mixed discrete-continuous optimization problems. $\mathrm{CES}_{\mathbf{MV}}$ allows the user to explicitly declare each variable as continuous, ordinal or categorical. In $\mathrm{CES}_{\mathbf{MV}}$, continuous variables are handled with a continuous optimization approach (CMA-ES), ordinal variables are handled with a continuous relaxation approach (Round), and categorical variables are handled with a categorical optimization approach ($\mathrm{CES}_{\mathbf{MV}}$-c) in each generation.

We also propose CES-RoundC and CES-RelayC which use other approaches to handle categorical variables from $\mathrm{CES}_{\mathbf{MV}}$. In CES-RoundC, categorical variables, together with ordinal variables, are handled by rounding continuous variables in each generation. CES-RelayC is a relay version of the two-partition strategy where variables of one partition (usually the categorical ones) are optimized separately for fixed values of the variables of the other partition (usually the continuous and ordinal ones).

Using the benchmark functions, the proposed algorithms are automatically configured. We evaluate $\mathrm{CES}_{\mathbf{MV}}$, CES-RoundC and CES-RelayC, and we identify $\mathrm{CES}_{\mathbf{MV}}$ as the most high-performing variant. Then we compare $\mathrm{CES}_{\mathbf{MV}}$ to $\mathrm{ACO}_{\mathbf{MV}}$ on the artificial, mixed discrete-continuous functions. The experimental results show that $\mathrm{CES}_{\mathbf{MV}}$ is competitive or superior to $\mathrm{ACO}_{\mathbf{MV}}$. Finally we test $\mathrm{CES}_{\mathbf{MV}}$ and $\mathrm{ACO}_{\mathbf{MV}}$ on mixed-variable engineering benchmark problems and compare their results with those found in the literature. The experimental results establish $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$ as new state-of-the-art algorithms for mixed discrete-continuous optimization problems. $\mathrm{CES}_{\mathbf{MV}}$ is presented in Chapter 5.

1.3 Additional contributions

In this section, we highlight the additional contributions of this thesis.

A note on the handling of bound constraints for the CEC'05 benchmark function set

The benchmark functions and some of the algorithms proposed for the special session on real parameter optimization of the 2005 IEEE Congress on Evolutionary Computation (CEC'05) play an important role in the assessment of the state of the art in continuous optimization. In this note, we first show that, if boundary constraints are not enforced, state-of-the-art algorithms produce on a majority of the CEC'05 benchmark functions infeasible best candidate solutions, even though the optima of 23 out of the 25 CEC'05 functions are within the specified bounds.

This observation has important implications on algorithm comparisons. In fact, this note also draws the attention to the fact that authors may have drawn wrong conclusions from experiments using the CEC'05 problems. We refer to Appendix C for more details.

Computational results for an automatically tuned IPOP-CMA-ES on the CEC'05 benchmark set

We experimentally show that IPOP-CMA-ES can be significantly improved with respect to its default parameters by applying an automatic algorithm configuration tool. In particular, we consider a separation between tuning and test sets. We refer to the Appendix D for more details.

Artificial bee colonies for continuous optimization: experimental analysis and improvements

The artificial bee colony (ABC) algorithm is a recent class of swarm intelligence algorithms that is loosely inspired by the foraging behavior of a honeybee swarm. It was introduced in 2005 using continuous optimization as example application. Similar to what has happened with other swarm intelligence techniques, after the initial proposal several researchers have studied variants of the original algorithm. Unfortunately, often the tests of these variants have been made under different experimental conditions and under different fine-tuning efforts for the algorithm parameters. We review various of the proposed variants to the original ABC algorithm. Next, we experimentally study several of the proposed variants under two settings, namely under their original parameter settings and after the use of an automatic algorithm configuration tool to provide a same effort for parameter fine-tuning. Finally, we study the effect of an additional local search phase on the performance of the ABC algorithms. We refer to Appendix E for more details.

1.4 Publications

During the development of the research presented in this thesis, a number of articles have been produced that the author, together with co-authors, has published or submitted for publication to journals and international conferences or workshops. The majority of these articles deal with ant colony optimization and evolutionary algorithms for continuous and mixed discrete-continuous optimization problems.

1.4.1 International journal submissions

1. Tianjun Liao and Thomas Stützle. A Simple and Effective Cooperative-Competitive Hybrid Algorithm for Continuous Optimization. Submitted to IEEE Transactions on Systems, Man, and Cybernetics, Part B.

   This article is about the enhancements on CMA-ES using a cooperative-competitive hybrid algorithm. The article establishes the proposed iCMAES-ILS algorithm as a new state-of-the-art algorithm for continuous optimization. This article mainly contributes to Chapter 4.

2. Tianjun Liao, Doğan Aydın, and Thomas Stützle. Artificial Bee Colonies for Continuous Optimization: Framework, Experimental Analysis, and Improvements. Conditionally accepted for Swarm Intelligence.

   This article reviews the proposed variants to the original ABC algorithm, and experimentally analyzes the proposed variants under both original and tuned parameter settings and shows the impact of an additional local search phase has on the performance of ABC algorithms. This article is given in Appendix E.

3. Tianjun Liao, Thomas Stützle, Marco A. Montes de Oca, and Marco Dorigo. A Unified Ant Colony Optimization Algorithm for Continuous Optimization. Revision submitted to European Journal of Operational Research.

   This article proposes a unified ant colony optimization framework for continuous optimization (UACOR). It shows the high potential ACO algorithms have for continuous optimization and that automatic algorithm configuration has a high potential also for the development of continuous optimizers out of algorithm components. This article mainly contributes to Chapter 3.

4. Tianjun Liao, Krzysztof Socha, Marco A. Montes de Oca, Thomas Stützle, and Marco Dorigo. Ant Colony Optimization for Mixed-Variable Optimization Problems. Conditionally accepted for IEEE Transactions on Evolutionary Computation.

   This article proposes $\text{ACO}_{\mathbf{MV}}$, an ant colony optimization algorithm to tackle mixed discrete-continuous optimization problems. This article also proposes a new set of artificial, mixed-variable benchmark functions. Comparisons with results from the literature demonstrate the effectiveness and robustness of $\text{ACO}_{\mathbf{MV}}$. This article mainly contributes to parts of Chapter 5.

5. Tianjun Liao, Daniel Molina, Marco A. Montes de Oca, and Thomas Stützle. A Note on the Bound Constraints Handling for the IEEE CEC'05 Benchmark Function Suite. Third revision submitted to Evolutionary Computation.

   The note shows that, if boundary constraints are not enforced, state-of-the-art algorithms produce on a majority of the CEC'05 benchmark functions infeasible best candidate solutions. This note also draws the attention to the fact that authors may have drawn wrong conclusions from experiments using the CEC'05 problems. This article is given in Appendix C.

6. Tianjun Liao, Marco A. Montes de Oca, and Thomas Stützle. Computational Results for an Automatically Tuned CMA-ES with Increasing Population Size on the CEC'05 Benchmark Set. Soft Computing, 17(6):1031-1046. 2013.

   The article shows that IPOP-CMA-ES can be significantly improved with respect to its default parameters by applying an automatic algorithm configuration tool. In particular, we consider a separation between tuning and test sets. This article is given in Appendix D.

1.4.2 International conferences and workshops (peer-reviewed)

1. Tianjun Liao and Thomas Stützle. Benchmark Results for a Simple Hybrid Algorithm on the CEC 2013 Benchmark Set for Real-parameter Optimization. Proceeding of IEEE Congress on Evolutionary Computation (CEC 2013). Special Session & Competition on Real-parameter Single Objective Optimization. IEEE Press, Piscataway, NJ, USA, 2013. Accepted.

2. Tianjun Liao and Thomas Stützle. Expensive Optimization Scenario: IPOP-CMA-ES with a Population Bound Mechanism for Noiseless Function Testbed. In A. Auger et al. (eds.), Proceedings of the Workshop for Real-Parameter Optimization of the Genetic and Evolutionary Computation Conference (GECCO 2013). ACM Press, New York, NY, 2013. Accepted.

3. Tianjun Liao and Thomas Stützle. Testing the Impact of Parameter Tuning on a Variant of IPOP-CMA-ES with a Bounded Maximum Population Size on the Noiseless BBOB Testbed. In A. Auger et al. (eds.), Proceedings of the Workshop for Real-Parameter Optimization of the Genetic and Evolutionary Computation Conference (GECCO 2013). ACM Press, New York, NY, 2013. Accepted.

4. Tianjun Liao and Thomas Stützle. Bounding the Population Size of IPOP-CMA-ES on the Noiseless BBOB Testbed. In A. Auger et al. (eds.), Proceedings of the Workshop for Real-Parameter Optimization of the Genetic and Evolutionary Computation Conference (GECCO 2013). ACM Press, New York, NY, 2013. Accepted.

5. Manuel López-Ibáñez, Tianjun Liao, and Thomas Stützle. On the Anytime Behavior of IPOP-CMA-ES. In C.A. Coello Coello et al. (eds.), Proceedings of Parallel Problem Solving from Nature 2012 (PPSN 2012), Vol. 7491 in Lecture Notes in Computer Science, pages 357-366, Springer, Heidelberg, Germany.

6. Tianjun Liao, Daniel Molina, Thomas Stützle, Marco A. Montes de Oca, and Marco Dorigo. An ACO Algorithm Benchmarked on the BBOB Noiseless Function Testbed. In A. Auger et al. (eds.), Proceedings of the Workshop for Real-Parameter Optimization of the Genetic and Evolutionary Computation Conference (GECCO 2012), pages 159-166, ACM Press, New York, NY, 2012.

7. Doğan Aydın, Tianjun Liao, Marco Montes de Oca, and Thomas Stützle. Improving Performance via Population Growth and Local Search: The Case of the Artificial Bee Colony Algorithm. In Jin-Kao Hao et al. (eds.), Proceedings of the 10th International Conference on Artificial Evolution (EA 2011), Vol. 7401 in Lecture Notes in Computer Science, pages 85-96, Springer, Heidelberg, Germany, 2012.

8. Tianjun Liao, Marco A. Montes de Oca, and Thomas Stützle. Tuning Parameters across Mixed Dimensional Instances: A Performance Scalability Study of Sep-G-CMA-ES. In E.Özcan et al. (eds.), Proceedings of the Workshop on Scaling Behaviours of Landscapes, Parameters, and Algorithms of the Genetic and Evolutionary Computation Conference (GECCO 2011), pages 703-706, ACM Press, New York, NY, 2011.

9. Tianjun Liao, Marco A. Montes de Oca, Doğan Aydın, Thomas Stützle, and Marco Dorigo. An Incremental Ant Colony Algorithm with Local Search for Continuous Optimization. In N. Krasnogor et al. (eds.), Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2011), pages 125-132, ACM Press, New York, NY, 2011. **This paper received the best paper award of the ACO-SI track at GECCO 2011.**

1.5 Structure of the thesis

The thesis is structured as follows. Chapter 2 describes the models of the continuous and mixed discrete-continuous optimization problems, and reviews the main

related optimization techniques relevant for this thesis. The two basic algorithms, ($ACO_{\mathbb{R}}$ and CMA-ES), on which large parts of this thesis rely on, are then introduced. We also describe the automatic algorithm configuration technique `irace` and the tuning methodology we used for the thesis.

Chapter 3 proposes a unified ACO framework for continuous optimization, which combines algorithmic components from three existing algorithms, $ACO_{\mathbb{R}}$, $DACO_{\mathbb{R}}$ and $IACO_{\mathbb{R}}$-LS. We call this framework UACOR. UACOR's design makes the automatic generation of new and high performing continuous ACO algorithms possible through the use of automatic algorithm configuration tools.

Chapter 4 proposes a structurally simple, hybrid algorithm that loosely couples IPOP-CMA-ES with an iterated local search (ILS) algorithm. We call this algorithm iCMAES-ILS. iCMAES-ILS consists of an initial competition phase that also features a cooperative aspect between IPOP-CMA-ES and the ILS algorithm. We also propose an automatically tuned iCMAES-ILS to examine further possible performance improvements.

Chapter 5 is concerned with mixed discrete-continuous optimization. We detail $ACO_{\mathbf{MV}}$ and study specific aspects of $ACO_{\mathbf{MV}}$. We present three CMA-ES extensions for tackling mixed discrete-continuous optimization problems. We also propose a set of artificial mixed-variable benchmark functions. Using the benchmark functions, we identify $CES_{\mathbf{MV}}$ as the most high-performing variant and then compare $CES_{\mathbf{MV}}$ and $ACO_{\mathbf{MV}}$. Finally, we test $CES_{\mathbf{MV}}$ and $ACO_{\mathbf{MV}}$ on mixed-variable engineering benchmark problems and compare their results with those found in the literature.

Chapter 6 concludes the thesis and give directions for future work.

This thesis includes appendices A, B, C, D and E. Appendix A gives the results obtained by a re-designed and improved UACOR framework. Appendix B provides mathematical formulations for the mixed-variable engineering optimization problems used in Chapter 5. Appendix C gives a note on the bound constraints handling for the CEC'05 benchmark function set. Appendix D shows computational results for an automatically tuned CMA-ES with increasing population size on the CEC'05 benchmark set. Appendix E provides experimental analysis and shows improvements to the artificial bee colonies for continuous optimization.

Chapter 2

Background

Generally speaking, optimization is the task of finding in a search space a solution with a best possible objective function value for a given problem. In this chapter, we describe in concise terms the research area and the most important techniques on which this thesis is based. In particular, we focus mainly on the description of the problem classes that we will tackle in this thesis and the main algorithmic techniques we use. We will discuss in a bit more detail the specific algorithmic techniques that form the basis of this thesis, while the others are only sketched. Some room is also given to the description of the benchmark sets that are used to evaluate the algorithms we proposed. In this thesis, we applied automatic algorithm configuration tools to obtain high-performing parameter settings and we studied the impact the usage of automatic configuration tools has when compared to an algorithm using default parameters. The automatic algorithm configuration tool used in this thesis and the methodology to apply these tools in continuous optimization are also shortly discussed.

This chapter is structured as follows. In Section 2.1, we describe the model of continuous optimization problems. We concisely review the main local search and metaheuristic optimization techniques for continuous optimization and the related benchmark function sets relevant for this thesis. In Section 2.2, we describe the model of mixed discrete-continuous optimization problems. We then classify and briefly describe the main approaches to tackle mixed variable problems available in the literature. In Section 2.3, we introduce in more detail the basic algorithms that contribute to this thesis. Section 2.4 describes an automatic algorithm configuration technique and the tuning methodology that we used in this thesis. We summarize the chapter in Section 2.5.

2.1 Continuous optimization

Continuous optimization problems are optimization problems where all variables have a domain that typically is a subset of the real numbers. We describe a model for a continuous optimization problem as follows:

Definition *A model $R = (\mathbf{S}, \mathbf{\Omega}, f)$ of a continuous optimization problem consists of*

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

The search space $\mathbf{S}$ is defined by a set of D variables $x_i, i = 1, \ldots, D$, all of which are continuous and for each $x_i, i = 1, \ldots, D$, we have that the domain of a variable x is a subset of the real numbers. A solution $S \in \mathbf{S}$ is a complete value assignment, that is, each decision variable is assigned a value. A feasible solution is a solution that satisfies all constraints in the set $\mathbf{\Omega}$. A global optimum $S^* \in \mathbf{S}$ is a feasible solution that satisfies $f(S^*) \leq f(S)\ \forall S \in \mathbf{S}$. The set of all globally optimal solutions is denoted by $\mathbf{S}^*, \mathbf{S}^* \subseteq \mathbf{S}$. Solving a continuous optimization problem requires finding at least one $S^* \in \mathbf{S}^*$.

This thesis considers, without loss of generality, minimization since maximizing an objective function is equivalent to minimizing the objective function multiplied by minus one. If the set of constraints $\mathbf{\Omega}$ is empty, the resulting continuous optimization problems are called unconstrained. In this thesis, we consider only problems with the simplest types of constraints, called bound constraints, which consist in lower and upper bounds on the values that each variable can take. In the case of the bound constraints, a solution $S \in \mathbf{S}$ is often given by $S = (x_1, x_2, \ldots, x_D)$ and $x_i \in [A_i, B_i]$, where $[A_i, B_i]$ is the search interval of dimension i, $1 \leq i \leq D$. Particularly, we have $x_i \in [A, B]$ if all variables are within the same range. Bound constraints arise in many practical problems, and are frequent in the benchmark problems for continuous optimization (e.g., CEC'05 benchmark functions [Suganthan et al., 2005]) that currently play an important role in the evaluation of continuous optimization algorithms.

Continuous optimization problems arise in a wide variety of real-world applications [Gönen, 1986, Jones and Pevzner, 2004, Zenios, 1996]. Often, these problems are difficult to solve. They involve an infinite number of possible solutions already

for a single variable and usually they have (many) more than one variable. The search space frequently involves complicated landscape properties such as nonlinear objective functions, objective functions where derivatives may not be easily computable or may not be computable at all; the variables may be correlated; and objective functions may have multiple local optima [Conn et al., 2009]. Some problems are black-box problems in the sense that there is no available explicit mathematical formulation. For example, this is always the case when the performance associated to specific variable settings has to be estimated by simulations. Due to these difficulties, which are inherent to many continuous problems, analytical or derivative-based methods [Stoer et al., 1993] are often impractical. As alternatives, derivative-free direct search methods [Conn et al., 2009, Kolda et al., 2003] such as downhill simplex method [Nelder and Mead, 1965], Powell's method [Powell, 1964, 2006, 2009] and pattern search [Torczon, 1997] have been developed in the mathematical programming literature. Metaheuristic optimization algorithms for tackling continuous optimization problems have also seen rapid development [Goldberg, 1989, Hansen and Ostermeier, 2001, Karaboga and Basturk, 2007, Kennedy and Eberhart, 1995, Molina et al., 2010a, Socha and Dorigo, 2008, Storn and Price, 1997].

2.1.1 Local search algorithms

Since many derivative-free direct search methods were designed to converge to local optima, we consider them in the scope of local search methods. Note that the local search methods described here do not necessarily converge to the closest local minimum as would do local iterative improvement algorithms in combinatorial optimization [Hoos and Stützle, 2005]. In fact, all the local search methods discussed here make use of a step size parameter that defines how far away new tentative search points are from the current ones. This step size is a crucial parameter of all these methods and in virtually all these methods the step size is adapted during the run of the algorithm.

The downhill simplex method for nonlinear continuous optimization was originally proposed by Nelder and Mead [1965], and it is also called Nelder-Mead method; a simplex is a convex polytope of $D + 1$ vertices in D dimensions.[1] This method measures the objective function at each candidate solution arranged as a simplex, and then replaces one of these candidate solution with a new solution.

[1]The Nelder-Mead "simplex" method should not be confused with the Simplex method [Dantzig and Thapa, 1997, 2003] for linear programming.

Much attention has been given to this method in literature and a large number of modifications [Kelley, 1999, Price et al., 2002, Tseng, 1999] to downhill simplex method have been proposed.

Powell's method, strictly called Powell's conjugate direction set, was originally proposed by Powell [1964]. It minimizes a D-dimensional objective function by searching along a set of conjugate directions that are constructed by line search vectors. A new line search vector is generated by a linear combination of line search vectors, and it is added to the line search vector list for next usage. Recently, Powell [2002] proposed UOBYQA, an unconstrained derivative-free direct search method using a quadratic model and the trust region paradigm [Conn et al., 1987]. Vanden Berghen and Bersini [2005] proposed a parallel, constrained extension of the UOBYQA, called CONDOR, which uses the results of the parallel computation to increase the quality of multivariate Lagrange interpolation [De Boor and Ron, 1990]. More recently, Powell [2006] proposed NEWUOA, which accelerates the creation of the quadratic model for UOBYQA. BOBYQA [Powell, 2009] is an extension of the NEWUOA that is able to handle bound constraints.

Pattern search is stated using a "pattern" of solutions to search. The original pattern search algorithm was proposed by Hooke and Jeeves [1961]. Torczon [1997] unifies a class of derivative-free direct search methods [Box, 1957, Davidon, 1991, Dennis and Torczon, 1991, Hooke and Jeeves, 1961] and introduces a generalized definition of pattern search methods. The definition requires the existence of a lattice T that if $\{S_1, \ldots, S_N\}$ are the first N iterates generated by a pattern search method, then there exists a scale factor ϕ_N such that the steps $\{S_1 - S_0, S_2 - S_1, \ldots, S_N - S_{N-1}\}$ all lie in the scaled lattice $\phi_N T$; lattice T is independent of the objective function. Audet and Dennis Jr [2006] proposed a mesh-based direct search algorithm called MADS.

In the following, we describe in some more detail the local search methods that are used in Chapters 3 and 4.

Powell's conjugate direction set

Powell's conjugate direction set method starts from an initial solution $S_0 \in \mathbb{R}^D$. In the first iteration, it performs D line searches using the unit vectors $\boldsymbol{e}_i$ as initial search directions $\boldsymbol{u}_i$. The initial step size of the search is a parameter ss. At each step, the new initial solution for the next line search is the best solution found by the previous line search. A solution S' denotes the minimum found after all D line searches. Next, the method eliminates the first search direction by doing $\boldsymbol{u}_i = \boldsymbol{u}_{i+1}$, $\forall i \in \{1 : D-1\}$, and replacing the last direction $\boldsymbol{u}_D$ by $S' - S_0$.

Then a move along the direction $\boldsymbol{u}_D$ is performed. The next iteration is executed from the best solution found in the previous iteration. The method terminates after a maximum number *LSIterations* of iterations, or when the tolerance, that is the relative change between solutions found in two consecutive iterations, is lower than a certain threshold. We used an implementation of the method as described in [Press et al., 1992].

BOBYQA

The bound constrained optimization by quadratic approximation (BOBYQA) algorithm [Powell, 2009] constructs at each iteration a quadratic model that interpolates m solutions in the current trust region, and samples a new, minimal solution using the model. The model is then updated by the true evaluation value of the new solution. If a new best-so-far solution is obtained, the trust region centers at this solution and enlarges its radius; otherwise, if the new solution is worse than the best-so-far solution, the trust region reduces its radius. The recommended minimal number of solutions to compute the quadratic model is $m = 2D + 1$ [Powell, 2009]. The method terminates after a maximum number *LSIterations* of iterations, or when the tolerance, that is the relative change between solutions found in two consecutive iterations, is lower than a certain threshold. We used the BOBYQA implementation of NLopt, a library for nonlinear optimization [Johnson, 2008].

Mtsls1

Mtsls1 [Tseng and Chen, 2008] is a recent, derivative-free local search algorithm for continuous optimization. The Mtsls1 local search has shown high performance and it is a crucial component of several high-performing algorithms for continuous optimization [LaTorre et al., 2011, Tseng and Chen, 2008]. Mtsls1 starts from an initial candidate solution $S = (x_1, x_2, \ldots, x_D)$ with an initial step size ss. The default setting of ss in Mtsls1 [Tseng and Chen, 2008] is $0.5 \times (B - A)$. In each Mtsls1 iteration, Mtsls1 searches along all dimensions one by one in a fixed order. The search in dimension i works as follows. First, set $x'_i \leftarrow x_i - ss$ and evaluate the resulting solution x'. If $f(x') < f(x)$, $x_i \leftarrow x'_i$ and the search continuous in the next dimension $i + 1$; otherwise, set $x''_i \leftarrow x_i + 0.5 \times ss$ and evaluate x''. If $f(x'') < f(x)$, $x_i \leftarrow x''_i$ and the search continues in dimension $i + 1$. If both tests fail, x_i remains as is and the next dimension $i + 1$ is examined. If one Mtsls1 iteration does not find an improvement in any of the dimensions, the next Mtsls1 iteration halves the search step size. The method terminates after a maximum

number *LSIterations* of iterations.

2.1.2 Metaheuristic based algorithms

Over the past decade, research efforts in the development of new, improved continuous optimization algorithms based on standard metaheuristic techniques have seen rapid development. These efforts mainly include techniques such as Genetic Algorithms (GAs) [Goldberg, 1989], Evolution Strategies (ESs) [Beyer and Schwefel, 2002, Hansen and Ostermeier, 2001], Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995], Differential Evolution (DE) [Storn and Price, 1997], Ant Colony Optimization (ACO)[Dorigo and Stützle, 2004, Socha and Dorigo, 2008], Artificial Bee Colonies (ABC) [Karaboga and Basturk, 2007] and Memetic Algorithms (MAs) [Kramer, 2010, Molina et al., 2010a, Moscato, 1999].

Genetic Algorithms

Genetic Algorithms (GAs) are inspired from natural evolution [Goldberg, 1989, Holland, 1975]. GAs make use of a population of solutions and iteratively adapt it by selection, crossover and mutation. Each iteration is called a generation. In each generation, promising solutions are stochastically selected from the current population, and solutions are recombined or mutated to produce new solutions. From the newly generated solutions through recombination and mutation and the solutions in the current generation, a new population of solutions is selected for the next generation.

Originally, problems were usually coded in chromosome using binary strings [Goldberg, 1989]. When tackling continuous optimization problems, nowadays real-numbers are frequently used in the solution representation and these algorithms are called real-coded GAs [Herrera et al., 1998, Lucasius and Kateman, 1989, Wright, 1991]. Much research on real-coded GAs is focused on the development of effective crossover operators. A crossover operator usually uses two parent solutions and combines their features to generate an offspring. Herrera et al. [2003] provide a taxonomy of crossover operators and analyze representative crossover operators such as discrete crossover operator, aggregation based crossover operator, neighbourhood-based crossover operator. Promising performance was also observed for hybrid crossover operators that combine two or more basic ones [Herrera et al., 2005]. Mutation is an operation that provides a random diversity in the population. The design of the effective mutation operators, the probability of applying mutation operator and the strength of the perturbation in

the mutation operator have frequently been studied [Herrera and Lozano, 2000, Hinterding, 1995, Michalewicz, 1992, Smith and Fogarty, 1997].

A common trend nowadays is to combine genetic algorithms with local search algorithms that improve some or all individuals of a population. These genetic local search or evolutionary local search algorithms are also often called memetic algorithms [Moscato, 1999]. This trend has mainly been started in applications to combinatorial optimization, but recently it became clear that this is also a very promising approach to enhance genetic algorithms for continuous optimization problems. For example, Lozano et al. [2004], Molina et al. [2010a,b, 2011] combine a real-coded genetic algorithm [Herrera et al., 1998] and a local search procedure [Hansen and Ostermeier, 2001, O'Reilly and Oppacher, 1995] to improve solutions.

Evolution Strategies

Evolution Strategies (ESs), as also GAs, belong to the class of evolutionary algorithms. ES was developed in the 1970s by Ingo Rechenberg, Hans-Paul Schwefel and their co-workers [Rechenberg, 1971, Schwefel, 1975]. ESs primarily apply mutation and selection to a population of individuals (or even a "single" solution) to iteratively search better solutions. When applied to continuous optimization problems, ESs use a real encoding, that is, for each variable a real number is given. Mutation is usually performed by adding a normally distributed random value to each variable value. The step size, which is also called mutation strength, is the standard deviation of the normal distribution. The selection in ESs is based on fitness rankings, not on the actual fitness values. The resulting algorithm is therefore invariant with respect to monotonic transformations of the objective function. The first evolution strategy is the $(1 + 1)$-ES [Rechenberg, 1971, Schwefel, 1975]. The term $(1 + 1)$ refers to its selection strategy. It selects among two solutions: the current solution (parent) and the result of its mutation. Only if the mutant's fitness is at least as good as the parent's solution, it becomes the parent of the next generation. Otherwise the mutant is disregarded.

Two canonical versions of ESs were defined. They are $(\mu+\lambda)$-ES and (μ, λ)-ES. μ and λ denote the number of parents and offspring, respectively. In $(\mu+\lambda)$-ES, μ parents generate λ offspring and then μ elite parents are selected out of the $\mu + \lambda$ solutions. In (μ, λ)-ES, the μ parents are discarded and the μ new parents for the next iteration are selected only from the λ offspring, $\lambda > \mu$.

A particularly successful ES algorithm for continuous optimization, called Covariance Matrix Adaptation Evolution Strategy (CMA-ES), was proposed by Hansen and Ostermeier [1996, 2001], Hansen et al. [2003]. It is a (μ, λ)-evolution

strategy that uses a multivariate normal distribution to sample new solutions at each iteration. CMA-ES can be used as a stand-alone algorithm but also as a local search as done by Ghosh et al. [2012], Molina et al. [2010a], Müller et al. [2009]. IPOP-CMA-ES is a CMA-ES variant that uses a restart schema coupled with an increasing population size. It was the best performing algorithm of the special session on real parameter optimization of the 2005 IEEE Congress on Evolutionary Computation (CEC'05) [Suganthan et al., 2005].

Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a swarm intelligence optimization algorithm that was inspired by the behavior of flocks of birds [Kennedy and Eberhart, 1995, 2001, Poli et al., 2007]. It was primarily used to tackle continuous optimization problems. In the PSO algorithm, the position of a particle corresponds to a solution in the search space and each particle has an associated velocity. The particles are organized according to some population topology such as a fully-connected one or ring graphs before the algorithm is run. This topology defines the neighborhood between particles [Clerc and Kennedy, 2002]. The algorithm then proceeds iteratively by updating in each iteration first the velocity and then the position of the particle. The velocity update for a particle is influenced by the best position the particle had so far and typically the position of the best particle in its neighborhood, which is defined by the population topology. Since the original PSO algorithm was proposed [Kennedy and Eberhart, 1995], many different variants of PSO algorithms have been proposed. These variants mainly refer to variations of the velocity update of a particle and the population topology [Clerc and Kennedy, 2002, Gimmler et al., 2006, Kennedy and Mendes, 2002, Mendes et al., 2004, Petalas et al., 2007, Poli et al., 2007, Shi and Eberhart, 1998].

Recently, the performance of PSO was further improved by using effective learning techniques or local search procedure. Liang et al. [2006] proposed a comprehensive learning PSO that updates a particle's velocity by using all other particles' best information in history. Montes de Oca et al. [2011] proposed an incremental social learning in particle swarms that consists in a PSO algorithm with a growing population size in which the initial position of new particles is biased toward the best-so-far solution; Powell's conjugate directions set [Powell, 1964] was used in this algorithm as a local search procedure to improve solutions. Müller et al. [2009] embedded CMA-ES as a local optimizer into PSO.

Differential Evolution

Differential Evolution (DE) [Storn and Price, 1997] is an efficient population-based algorithm for continuous optimization. DE's basic strategy includes a cycle of mutation, crossover and selection. After initialization, DE produces a new vector by adding the weighted difference between two vectors in the population to a third vector. This operation is called mutation and the newly generated vector is called a mutant vector. After the mutation phase, a crossover operation is applied to a pair of a predetermined vector, the so called target vector, and its corresponding mutant vector to generate a trial vector. The objective function value of each trial vector is then compared to that of its corresponding target vector. If the trial vector is better than the corresponding target vector, the trial vector replaces the target vector and it is added to the next generation. Otherwise, the target vector remains in the current population.

A large number of DE variants were developed [Das et al., 2011]. The developments mainly refer to parameter control, new mutation, crossover and selection operators, adaptive mechanisms and hybridization with other techniques. Some prominent DE variants are described in the following. SaDE [Qin et al., 2009] adaptively updates trial vector generation strategies and the control parameters; JADE [Zhang and Sanderson, 2009] introduces a new mutation operator, and uses an optional external archive to record history information and to adaptively update control parameters; DEGL [Das et al., 2009] introduces two neighborhood topologies to balance exploitation and exploration; HDDE [Dorronsoro and Bouvry, 2011] includes two heterogeneous islands in which different mutation operators are generated; MOS-DE [LaTorre et al., 2011] hybridizes DE with a local search technique. Das et al. [2011] surveyed more details of state-of-the-art DE variants and we refer to this paper for more details.

Ant Colony Optimization

Ant Colony Optimization (ACO) algorithms were first proposed for tackling combinatorial optimization problems [Dorigo et al., 1991, 1996]. ACO algorithms for combinatorial optimization problems make use of a so-called pheromone model in order to probabilistically construct solutions. A pheromone model consists of a set of numerical values, called pheromones, that are a function of the search experience of the algorithm. The pheromone model is used to bias the solution construction towards regions of the search space containing high quality solutions. The ACO metaheuristic [Dorigo and Di Caro, 1999, Dorigo and Stützle, 2004] de-

fines a class of optimization algorithms inspired by the foraging behavior of real ants. The main algorithmic components of the ACO metaheuristic are the ants' solution construction and the update of the pheromone information. Additional "daemon actions" are procedures that carry out tasks that cannot be performed by single ants. A common example is the activation of a local search procedure to improve an ant's solution or the application of additional pheromone modifications derived from globally available information about, for example, the best solutions constructed so far. Although daemon actions are optional, in practice they can greatly improve the performance of ACO algorithms.

After the initial proposals of ACO algorithms for combinatorial problems, often ant-inspired algorithms for continuous optimization problems were proposed [Bilchev and Parmee, 1995, Dréo and Siarry, 2004, Hu et al., 2008, 2010, Monmarché et al., 2000]. However, as explained in [Socha and Dorigo, 2008], most of these algorithms use search mechanisms different from those used in the ACO metaheuristic. The first algorithm that can be classified as an ACO algorithm for continuous domains is $ACO_{\mathbb{R}}$ [Socha and Dorigo, 2008]. In $ACO_{\mathbb{R}}$, the discrete probability distributions used in the solution construction by ACO algorithms for combinatorial optimization are substituted by probability density functions (PDFs) (i.e., continuous probability distributions). $ACO_{\mathbb{R}}$ uses a solution archive [Guntsch and Middendorf, 2002] for the derivation of these PDFs over the search space. Additionally, $ACO_{\mathbb{R}}$ uses sums of weighted Gaussian functions to generate multimodal PDFs. $DACO_{\mathbb{R}}$ [Leguizamón and Coello, 2010] is another recent ACO algorithm for continuous optimization. $IACO_{\mathbb{R}}$-LS algorithm [Liao et al., 2011b] is a more recent variant that I have proposed during my doctoral research.

Artificial Bee Colonies

Artificial Bee Colony (ABC) is a recent class of swarm intelligence algorithms that is loosely inspired by the foraging behavior of a honeybee swarm [Karaboga, 2005, Karaboga and Basturk, 2007]. The ABC algorithm involves three phases of foraging behavior which are conducted by employed bees, onlookers and scouts, respectively. Each food source corresponds to a solution of the problem. Employed bees and onlooker bees both exploit current food sources (solutions) by visiting its neighborhood. While there is a one-to-one correspondence between employed bees and food sources, that is, each employed bee is assigned to a different food source, the onlooker bees select randomly the food source to exploit, preferring better quality food sources. Scout bees explore the area for new food sources (solutions) if current food sources are deemed to be depleted. If more than *limit* times an

employed bee or an onlooker bee has visited unsuccessfully a food source, a scout bee searches for a new, randomly located food source.

The original ABC algorithm was introduced in 2005 using its example application to continuous optimization problems [Karaboga, 2005]. The original ABC algorithm obtained encouraging results on some standard benchmark problems, but, being an initial proposal, still a considerable performance gap with respect to state-of-the-art algorithms was observed. In particular, it was found to be relatively poor performing on composite and non-separable function as well as having a slow convergence rate towards high quality solutions [Akay and Karaboga, 2012]. Therefore, in the following years, a number of modifications of the original ABC algorithm were introduced trying to improve performance [Alataş, 2010, Aydın et al., 2012, Banharnsakun et al., 2011, Diwold et al., 2011a, Gao and Liu, 2011, Kang et al., 2011, Zhu and Kwong, 2010]. Unfortunately, so far there is no comprehensive comparative evaluation of the performance of ABC variants on a significantly large benchmark set available.

2.1.3 Benchmark functions sets

Test functions are commonly used to evaluate continuous optimization algorithms. The history of test functions can be traced back to many years ago in the field of mathematical optimization and many of the functions defined there are still nowadays used to evaluate continuous optimization. A well known example is Rosenbrock [1960], who introduced a non-convex function, called Rosenbrock function, which is known to have a global minimum inside a long, narrow, parabolic shaped flat valley. There were also many other well known functions proposed such as Ackley [Ackley, 1987], Griewank [Griewank, 1981] and Rastrigin [Törn and Zilinskas, 1989]. These test functions have often been used to tune, improve and compare continuous optimization algorithms.

To measure the performance of continuous optimization algorithms on a variety of function characteristics, sets of test functions were introduced. One early test function set is De Jong's set of five functions [Jong, 1975], which was used to test the performance of various genetic algorithms. In May 1996, Bersini et al. [1996] organized the first international contest on evolutionary optimization at the IEEE international conference on evolutionary computation, where a benchmark set of five functions with different characteristics (e.g., unimodality, multi-modality and separability) was given. Eight participants tested their algorithms on this continuous function benchmark. In the same year, Whitley et al. [1996] discussed some

basic principles that can be used to develop benchmark function sets. Several years later, various benchmark function sets [Hansen et al., 2009a, Herrera et al., 2010, Suganthan et al., 2005, Tang et al., 2007] were established and are nowadays widely used for evaluating continuous optimization algorithms. One influential, and nowadays widely used benchmark set was proposed for the special session on real parameter optimization of the 2005 IEEE Congress on Evolutionary Computation (CEC'05) [Suganthan et al., 2005]. The CEC'05 benchmark set comprises 25 benchmark functions, each of which is freely scalable. It specifies problem dimensionality, defined domains of variables, algorithm termination error values and the maximum number of function evaluations available for the continuous optimizers. The CEC'05 set has become a standard benchmark set that any researcher can use to evaluate the performance of new algorithms and compare algorithms. The central role that the CEC'05 benchmark function set currently plays is illustrated by the more than 600 citations in google scholar (as of April 2013) to the original technical report that introduced the benchmark function set. These benchmark functions and some of the algorithms proposed for them play an important role in the assessment of the state of the art in the continuous optimization field.

A recent special issue of the Soft Computing journal (SOCO) [Herrera et al., 2010, Lozano et al., 2011] provided another set of 19 freely scalable benchmark functions to evaluate the latest development of continuous algorithms for tackling large scale functions. Thirteen continuous optimizers based on different metaheuristics were finally selected for publication [Lozano et al., 2011]. Functions f_{soco1}–f_{soco6} were originally proposed for the special session on large scale global optimization organized for the IEEE 2008 Congress on Evolutionary Computation (CEC'08) Tang et al. [2007]. Functions f_{soco7}-f_{soco11} were proposed at the 9th International Conference on Intelligent Systems Design and Applications (ISDA'09). Functions f_{soco12}-f_{soco19} are hybrid functions that each combine two functions belonging to f_{soco1}–f_{soco11}. The CEC'05 and the SOCO benchmark sets include both functions that show a number of different characteristics. Their functions include unimodal functions such as Sphere and Schwefel 1.2, multi-modal functions such as Ackley and Rastrigin, and some functions contained in the benchmark sets are separable while others are not. Both benchmark sets comprise a number of hybrid functions. However, algorithm behavior and relative algorithm performance may differ quite strongly between the two benchmark sets. For example, IPOP-CMA-ES is among the top performers across the CEC'05 benchmark function set [Suganthan et al., 2005], while its performance compared to other algorithms on the SOCO benchmark set is much worse [Herrera et al., 2010, Lozano et al., 2011].

In fact, a significant difference between the CEC'05 and the SOCO benchmark function set is that 16 of the 25 CEC benchmark functions are rotated with respect to the orthogonal Cartesian coordinate system, which is not done in the SOCO benchmark function set.[2] The CEC'05 and SOCO benchmark function set are used in this thesis for evaluating continuous optimization algorithms. The CEC'05 and SOCO benchmark functions together with their main properties are listed in Tables 2.1 and 2.2.

Another relevant recent benchmark set is the Black-Box Optimization Benchmarking (BBOB) benchmark set [Hansen et al., 2009a]. Despite that this benchmark set was not used in this thesis, it has attracted our attention in our ongoing works [Liao and Stützle, 2013a,b,c, Liao et al., 2012]. The BBOB benchmark set was originally proposed in the Black-Box Optimization Benchmarking workshop that was organized as part of GECCO 2009. Since this first edition, the workshop has been organized three more times. In the BBOB benchmark set [Hansen et al., 2009a], 24 systematic, scalable benchmark functions are presented. Except the first subgroup of separable functions, the benchmark functions are non-separable and most of the functions are rotated, such as those in the CEC'05 benchmark set. BBOB also provides tools for post-processing and presenting results. Different from other benchmark sets [Herrera et al., 2010, Suganthan et al., 2005, Tang et al., 2007], the BBOB benchmark set especially puts emphasis on the success rates and the used number of function evaluations to reach thresholds on the function values to be reached. The maximum number of function evaluations is not fixed in the experimental setups.

[2]In the SOCO benchmark set the functions are not rotated mainly because rotating the functions for evaluation is rather costly especially for the large dimensions that are considered in the SOCO benchmark function set [Herrera et al., 2010, Lozano et al., 2011].

Table 2.1: High-level description of the benchmark functions of the CEC'25 benchmark set. Given is in the column ID the function identifier, in column Name/Description the common name for the function or a short description, in column range the feasible range of the variables' values, whether the functions are unimodal or multi-modal (column Uni/Multi-modal), whether the functions are separable (Y) or not (N) and whether the functions are rotated (Y) or not (N).

ID	Name/Description	Range $[X_{\min}, X_{\max}]^D$	Uni/Multi-modal	Separable	Rotated
f_{cec1}	Shift.Sphere	[-100,100]^D	U	Y	N
f_{cec2}	Shift.Schwefel 1.2	[-100,100]^D	U	N	N
f_{cec3}	Shift.Ro.Elliptic	[-100,100]^D	U	N	Y
f_{cec4}	Shift.Schwefel 1.2 Noise	[-100,100]^D	U	N	N
f_{cec5}	Schwefel 2.6 Opt on Bound	[-100,100]^D	U	N	N
f_{cec6}	Shift.Rosenbrock	[-100,100]^D	M	N	N
f_{cec7}	Shift.Ro.Griewank No Bound	[0,600]^{D†}	M	N	Y
f_{cec8}	Shift.Ro.Ackley Opt on Bound	[-32,32]^D	M	N	Y
f_{cec9}	Shift.Rastrigin	[-5,5]^D	M	Y	N
f_{cec10}	Shift.Ro.Rastrigin	[-5,5]^D	M	N	Y
f_{cec11}	Shift.Ro.Weierstrass	[-0.5,0.5]^D	M	N	Y
f_{cec12}	Schwefel 2.13	[-π,π]^D	M	N	N
f_{cec13}	Griewank plus Rosenbrock	[-3,1]^D	M	N	N
f_{cec14}	Shift.Ro.Exp.Scaffer	[-100,100]^D	M	N	Y
f_{cec15}	Hybrid Composition	[-5,5]^D	M	N	N
f_{cec16}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec17}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec18}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec19}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec20}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec21}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec22}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec23}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec24}	Ro. Hybrid Composition	[-5,5]^D	M	N	Y
f_{cec25}	Ro. Hybrid Composition	[2,5]^{D†}	M	N	Y

† denotes initialization range but not bound constraints. Its global optimum is outside of initialization range.

Table 2.2: High-level description of the benchmark functions of the SOCO benchmark set. Given is in the column ID the function identifier, in column Name/Description the common name for the function or a short description, in column range the feasible range of the variables' values, whether the functions are unimodal or multi-modal (column Uni/Multi-modal), whether the functions are separable (Y) or not (N) and whether the functions are rotated (Y) or not (N).

ID	Name/Description	Range $[X_{\min}, X_{\max}]^D$	Uni/Multi-modal	Separable	Rotated
f_{soco1}	Shift.Sphere	[-100,100]^D	U	Y	N
f_{soco2}	Shift.Schwefel 2.21	[-100,100]^D	U	N	N
f_{soco3}	Shift.Rosenbrock	[-100,100]^D	M	N	N
f_{soco4}	Shift.Rastrigin	[-5,5]^D	M	Y	N
f_{soco5}	Shift.Griewank	[-600,600]^D	M	N	N
f_{soco6}	Shift.Ackley	[-32,32]^D	M	Y	N
f_{soco7}	Shift.Schwefel 2.22	[-10,10]^D	U	Y	N
f_{soco8}	Shift.Schwefel 1.2	[-65.536,65.536]^D	U	N	N
f_{soco9}	Shift.Extended f_{10}	[-100,100]^D	U	N	N
f_{soco10}	Shift.Bohachevsky	[-15,15]^D	U	N	N
f_{soco11}	Shift.Schaffer	[-100,100]^D	U	N	N
f_{soco12}	Composition $f_{soco9} \oplus_{0.25} f_{soco1}$	[-100,100]^D	M	N	N
f_{soco13}	Composition $f_{soco9} \oplus_{0.25} f_{soco3}$	[-100,100]^D	M	N	N
f_{soco14}	Composition $f_{soco9} \oplus_{0.25} f_{soco4}$	[-5,5]^D	M	N	N
f_{soco15}	Composition $f_{soco10} \oplus_{0.25} f_{soco7}$	[-10,10]^D	M	N	N
f_{soco16}	Composition $f_{soco9} \oplus_{0.5} f_{soco1}$	[-100,100]^D	M	N	N
f_{soco17}	Composition $f_{soco9} \oplus_{0.75} f_{soco3}$	[-100,100]^D	M	N	N
f_{soco18}	Composition $f_{soco9} \oplus_{0.75} f_{soco4}$	[-5,5]^D	M	N	N
f_{soco19}	Composition $f_{soco10} \oplus_{0.75} f_{soco7}$	[-10,10]^D	M	N	N

2.2 Mixed discrete-continuous optimization

Many real-world optimization problems can be modeled using combinations of continuous and discrete variables. Mixed integer programming (linear or nonlinear) refers to mathematical programming with continuous and integer variables in the (linear or nonlinear) objective function and constraints [Bussieck and Pruessner, 2003, Nemhauser and Wolsey, 1988, Wolsey, 1998]. In this thesis, we consider mixed discrete-continuous optimization problems where the discrete variables can be ordinal or categorical. Ordinal variables exhibit a natural ordering relation (e.g., integers or $\{\text{small}, \text{medium}, \text{large}\}$). Categorical variables take their values from a finite set of categories [Abramson et al., 2009], which often identify non-numeric elements of an unordered set (e.g., colors, shapes or types of material). We describe a model for a mixed discrete-continuous optimization problem as follows:

Definition *A model $R = (\mathbf{S}, \mathbf{\Omega}, f)$ of a mixed discrete-continuous optimization problem 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 $f : \mathbf{S} \to \mathbb{R}$ to be minimized.*

The search space $\mathbf{S}$ is defined by a set of $n = d + r$ variables $x_i, i = 1, \ldots, n$, of which d are discrete and r are continuous. The discrete variables include o ordinal variables and c categorical ones, $d = o + c$. o or c can be equal to zero. When $o = 0$, the discrete variables only include categorical variables; when $c = 0$, the discrete variables only include ordinal variables; when $o = c = 0$, the problem is a particular case of mixed variables with an empty set of discrete variables, namely a continuous optimization problem. A solution $S \in \mathbf{S}$ is a complete value assignment, that is, each decision variable is assigned a value. A feasible solution is a solution that satisfies all constraints in the set $\mathbf{\Omega}$. A global optimum $S^* \in \mathbf{S}$ is a feasible solution that satisfies $f(S^*) \leq f(S)\ \forall S \in \mathbf{S}$. The set of all globally optimal solutions is denoted by $\mathbf{S}^*, \mathbf{S}^* \subseteq \mathbf{S}$. Solving a mixed discrete-continuous optimization problem requires finding at least one $S^* \in \mathbf{S}^*$.

Mixed discrete-continuous optimization does not enjoy, despite its high practical relevance, such a great popularity as continuous optimization and therefore fewer algorithms for handling these problems are available in the literature. These algorithms may be divided into three groups.

The first group is based on a *two-partition approach*, in which the variables are partitioned into continuous variables and discrete variables. Variables of one partition are optimized separately for fixed values of the variables of the other partition [Lucidi et al., 2005, Praharaj and Azarm, 1992, Sambo et al., 2012, Wagner et al., 2011]. This approach often leads to a large number of objective function evaluations [Stelmack and Batill, 1997]. Additionally, since the dependency between variables belonging to different partitions is not explicitly handled, algorithms using this approach are prone to finding sub-optimal solutions.

The second group takes a *continuous relaxation approach* [Dimopoulos, 2007, Gao and Hailu, 2010, Guo et al., 2004, Lampinen and Zelinka, 1999a,b, Mashinchi et al., 2011, Rao and Xiong, 2005, Turkkan, 2003]. In this group, all variables are handled as continuous variables. Ordinal variables are relaxed to continuous variables, and are repaired when evaluating the objective function. The repair mechanism is used to return a discrete value in each iteration. The simplest repair mechanisms are truncation and rounding [Guo et al., 2004, Lampinen and Zelinka, 1999a]. It is also possible to treat categorical variables using continuous relaxations [Abhishek et al., 2010]. In general, the performance of algorithms based on the continuous relaxation approach depends on the continuous solvers and on the repair mechanism.

The third group uses a *categorical optimization approach* [Abramson, 2002, 2004, Abramson et al., 2009, Audet and Dennis, 2001, Deb and Goyal, 1998, Kokkolaras et al., 2001, Ocenasek and Schwarz, 2002, Socha, 2008] to directly handle discrete variables without a continuous relaxation. Thus, any possible ordering relations that may exist between discrete variables are ignored and, thus, all discrete variables, ordinal and categorical, are treated as categorical ones.[3] In this group, continuous variables are handled by a continuous optimization method. Genetic adaptive search [Deb and Goyal, 1998], pattern search [Audet and Dennis, 2001], and mixed Bayesian optimization [Ocenasek and Schwarz, 2002] are among the approaches that have been proposed.

The mixed discrete-continuous optimization problems found in the literature often originate from the mechanical engineering field. Unfortunately, these problems cannot be easily parametrized and flexibly manipulated for investigating the performance of mixed discrete-continuous optimization algorithms in a systematic way. Therefore, a set of mixed discrete-continuous benchmark functions

[3]Note that the special case of mixed discrete-continuous optimization problems, where the variables can be either continuous or categorical, is also called mixed-variable programming problem [Abramson, 2002, Audet and Dennis, 2001].

that allow the definition of a controlled environment for the investigation of algorithm performance and tuning of algorithm parameters are required. One way how to construct mixed-variable benchmark functions was originally proposed by Socha [2008]. Following these ideas, we propose a set of artificial mixed discrete-continuous benchmark functions (See Section 5.1) that any researcher can use to evaluate the performance of new algorithms and compare algorithms.

2.3 Basic Algorithms

In this section, we introduce the two basic heuristic optimization techniques on which large parts of this thesis rely on: $\text{ACO}_{\mathbb{R}}$ and CMA-ES.

2.3.1 $\text{ACO}_{\mathbb{R}}$

ACO algorithms for combinatorial optimization problems make use of a so-called pheromone model in order to probabilistically construct solutions. A pheromone model consists of a set of numerical values, called pheromones, that are a function of the search experience of the algorithm. The pheromone model is used to bias the solution construction towards regions of the search space containing high quality solutions. As such, ACO algorithms follow a model-based search paradigm [Zlochin et al., 2004] as, for example, also estimation of distribution algorithms [Larrañaga and Lozano, 2002] do. In ACO for combinatorial optimization problems, the pheromone values are associated with a finite set of discrete components. This is not possible if continuous variables are involved. Therefore, Socha and Dorigo [2008] replaced the discrete probability distribution with probability density functions (PDFs) in the solution construction for continuous domains and proposed an ACO algorithm for continuous domains, called $\text{ACO}_{\mathbb{R}}$. $\text{ACO}_{\mathbb{R}}$ won the 2012 European Journal of Operational Research Top Cited Article Award and had more than 360 citations according to google scholar as of March 2013. $\text{ACO}_{\mathbb{R}}$ uses a solution archive [Guntsch and Middendorf, 2002] for the derivation of these PDFs over the search space. Additionally, $\text{ACO}_{\mathbb{R}}$ uses sums of weighted Gaussian functions to generate multimodal PDFs.

$\text{ACO}_{\mathbb{R}}$ initializes the solution archive with k solutions that are generated uniformly at random. Each solution is a D-dimensional vector with real-valued components $x_i \in [A, B]$, $i = 1, \ldots, D$. We assume that the optimization problems are unconstrained except possibly for bound constraints of the D real-valued variables x_i. The k solutions of the archive are kept sorted according to their quality (from

best to worst) and each solution S_j has associated a weight ω_j. This weight is calculated using a Gaussian function as:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2k^2}}, \tag{2.1}$$

where $rank(j)$ is the rank of solution S_j in the sorted archive, and q is a parameter of the algorithm. The result of computing $rank(j) - 1$ is that the best solution receives the highest weight.

The weights are used to choose probabilistically a guiding solution around which a new candidate solution is generated. The probability of choosing solution S_j as a guiding solution is given by $\omega_j / \sum_{a=1}^{k} \omega_a$ so that the better the solution, the higher the chances of choosing that solution to guide the search. Once a guiding solution S_{guide} is chosen, the algorithm samples the neighborhood of the i-th real-valued component of the guiding solution S^i_{guide} using a Gaussian PDF with $\mu^i_{\text{guide}} = S^i_{\text{guide}}$, and σ^i_{guide} equal to

$$\sigma^i_{\text{guide}} = \xi \sum_{r=1}^{k} \frac{|S^i_r - S^i_{\text{guide}}|}{k-1}, \tag{2.2}$$

which is the average distance between the value of the i-th component of S_{guide} and the values of the i-th components of the other solutions in the archive, multiplied by a parameter ξ. The process of choosing a guiding solution and generating a candidate solution is repeated a total of Na times (corresponding to the number of "ants") per iteration. Before the next iteration, the algorithm updates the solution archive keeping only the best k of the $k+Na$ solutions that are available after the solution construction process.

An outline of $\text{ACO}_{\mathbb{R}}$ is given in Algorithm 1. The structure of the solution archive and the Gaussian functions used to generate PDFs in $\text{ACO}_{\mathbb{R}}$ are shown in Figure 2.1. Thanks to the pheromone representation used in $\text{ACO}_{\mathbb{R}}$ (that is, the solution archive), it is possible to take into account the correlation between the decision variables. A non-deterministic adaptive method for doing so is presented in [Socha and Dorigo, 2008]. This method rotates coordinate axes and recalculates all the current coordinates of all the solutions in the archive to construct new temporary solutions according to an adaptively generated orthogonal base. After the solution construction process, this method converts those temporary solutions back into the original coordinate system. More details are given in [Socha and Dorigo, 2008].

Algorithm 1 Outline of $\text{ACO}_{\mathbb{R}}$

Input: k, Na, D, q, ξ, and termination criterion.
Output: The best solution found

```
Initialize and evaluate k solutions
/* Sort solutions and store them in the archive */
T = Sort(S_1 ··· S_k)
while Termination criterion is not satisfied do
  /* Generate Na new solutions */
  for l = 1 to Na do
    /* Construct solution */
    Select S_guide according to weights
    for i = 1 to D do
      Sample Gaussian (μ^i_guide, σ^i_guide)
    end for
    Store and evaluate newly generated solution
  end for
  /* Sort solutions and select the best k */
  T = Best(Sort(S_1 ··· S_{k+Na}), k)
end while
```

2.3.2 CMA-ES

Evolution Strategies primarily apply mutations based on a multivariate normal distribution and rank-based selection to a population of individuals to iteratively search better solutions. (μ, λ)-ES is one canonical version of ES. μ and λ denote the number of parents and offspring, respectively. In (μ, λ)-ES, the old μ parents are discarded and the new, elite μ parents for the next iteration are only selected from λ offspring, $\lambda > \mu$. Hansen and Ostermeier [1996, 2001], Hansen et al. [2003] proposed a particularly successful ES algorithm for continuous optimization, called Covariance Matrix Adaptation Evolution Strategy (CMA-ES), which is a (μ, λ)-evolution strategy that iteratively uses a multivariate normal distribution to sample new solutions. CMA-ES adapts the full covariance matrix of this normal distribution to guide the sampling of new solutions. The search mechanism is invariant against linear transformations of the search space, which clearly is a desirable property for a continuous optimization algorithm

The core steps of CMA-ES in each iteration are sampling of a new population, selection, and adaption of the step size and the covariance matrix. At each iteration t, CMA-ES samples λ individuals according to a multi-variate normal distribution $S_i = \boldsymbol{m}^t + \sigma^t \times \mathcal{N}_i\big(\mathbf{0}, \boldsymbol{C}^t\big), i = 1 \ldots \lambda$, where $\mathcal{N}\big(\mathbf{0}, \boldsymbol{C}^t\big)$ denotes a normally distributed random vector with mean $\mathbf{0}$ and covariance matrix $\boldsymbol{C}^t$. σ^t

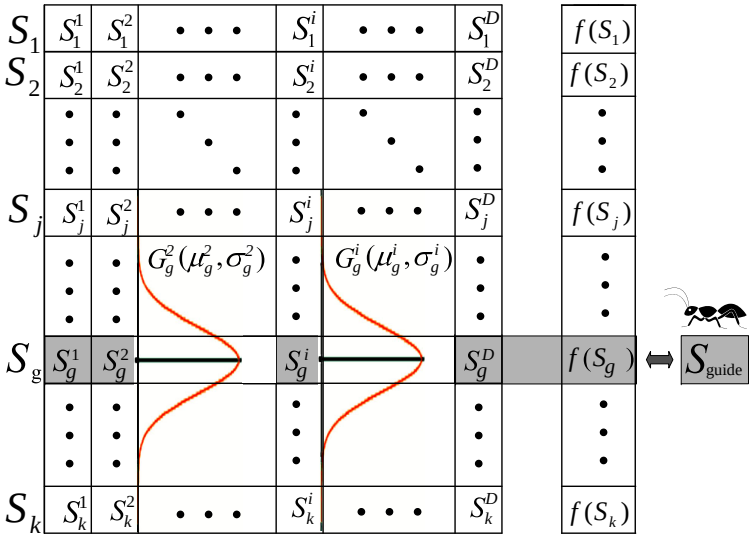


Figure 2.1: This figure indicates the structure of the solution archive and the Gaussian functions used to generate PDFs in $\text{ACO}_{\mathbb{R}}$.

is the step size, $\sigma^t > 0$. All λ individuals S_i use the same distribution with mean $\boldsymbol{m}^t$. Then these individuals are evaluated and ranked according to their objective function values. Using the ranked population, the distribution parameters $\boldsymbol{m}^t$, σ^t and $\boldsymbol{C}^t$ are updated for a new iteration. The new mean of the distribution $\boldsymbol{m}^{t+1}$ is set to the weighted sum of the best μ individuals ($\boldsymbol{m} = \sum_{i=1}^{\mu} w_i \boldsymbol{x}_{i_\lambda}^t$, with $w_i > 0$ for $i = 1 \ldots \mu$ and $\sum_{i=1}^{\mu} w_i = 1$, where index i_λ denotes the i-th best individual). Then two evolution paths called $\boldsymbol{p}_\sigma^{t+1}$ and $\boldsymbol{p}_c^{t+1}$ are computed. An evolution path $\boldsymbol{p}$ is expressed by a sum of consecutive steps of the distribution mean, the strategy takes over a number of generations. Specifically, if consecutive steps are taken in a similar (opposite) direction, the evolution paths become long (short). $\boldsymbol{p}_\sigma$ makes consecutive movements of the distribution mean orthogonal in expectation; $\boldsymbol{p}_c$ facilitates a much faster variance increase of favorable directions. Finally $\boldsymbol{p}_\sigma^{t+1}$ and $\boldsymbol{p}_c^{t+1}$ are used to updated σ^{t+1} and $\boldsymbol{C}^{t+1}$. For a detailed explanation of the CMA-ES equations for computing $\boldsymbol{p}_\sigma^{t+1}$ and $\boldsymbol{p}_c^{t+1}$ and updating σ^{t+1} and $\boldsymbol{C}^{t+1}$, we refer to Hansen [2010].

The popularity of CMA-ES is illustrated by the more than 1100 citations to Hansen and Ostermeier [2001] according to google scholar as of March 2013. CMA-ES can be used as a stand-alone algorithm but it is often used as a local optimizer in other metaheuristics [Ghosh et al., 2012, Molina et al., 2010a, Müller et al., 2009]. IPOP-CMA-ES embeds CMA-ES into a restart mechanism that increases the population size between successive runs of CMA-ES. IPOP-CMA-ES is a current state-of-the-art algorithm for continuous optimization and it was the best per-

Algorithm 2 Outline of IPOP-CMA-ES

Input: An initial candidate solution S and termination criterion
Output: The best found solution

```
while termination criterion is not satisfied do
  while stopping criterion for restart is not satisfied do
    /* CMA-ES iterations */
    Sample a new population (λ, σ^0)
    Selection Best(Sort(S_1 ··· S_λ), μ)
    Adapt step size and covariance matrix
  end while
  Increase population size and uniformly sample an initial solution
end while
```

forming algorithm of the special session on real parameter optimization at CEC'05. Whether and how much IPOP-CMA-ES could be further improved therefore becomes very interesting. An outline of the IPOP-CMA-ES algorithm is given in Algorithm 2. The default parameter settings of IPOP-CMA-ES are the following. The initial population size is $\lambda = 4 + \lfloor 3 \ln(D) \rfloor$ and the number of search points selected for the parent population is $\mu = \lfloor 0.5\lambda \rfloor$. The initial step-size is $\sigma^0 = 0.5(B - A)$. At each restart, IPOP-CMA-ES increases the population size by a factor $d = 2$. Restarts are triggered using the three paramaters *stopTolFunHist*, *stopTolFun* and *stopTolX*; they refer to the improvement of the best objective function value in the last $10 + \lceil 30D/\lambda \rceil$ generations, the function values of the recent generation, and the standard deviation of the normal distribution in all coordinates, respectively.

Recently, Hansen [2009] proposed BIPOP-CMA-ES. It is a CMA-ES variant that couples two restart regimes, one with an increasing population size as in IPOP-CMA-ES and another with small population sizes. At each restart of BIPOP-CMA-ES, by comparing the budget of function evaluations used so far in the corresponding regime, the regime with smaller budget value is applied. The first regime with an increasing population size is applied for the first restart. BIPOP-CMA-ES and IPOP-CMA-ES obtained the best performance on the BBOB benchmark set [Hansen et al., 2009a] in 2009 and 2010. Alternative restart strategies were proposed by Loshchilov et al. [2012a]. Various IPOP-CMA-ES variants [Brockhoff et al., 2012, Hansen, 2009, Loshchilov et al., 2012b,c,d, 2013] were also tested in the BBOB benchmark set.

2.4 Automatic algorithm configuration

Assigning appropriate values for the parameters of optimization algorithms is an important task [Birattari, 2009]. Although algorithm designers have spent a considerable effort in the design choices and certainly also in the definition of its parameters, over the last few years evidence has arisen that many algorithms' performance can be improved by considering automatic algorithm configuration and tuning tools [Adenso-Diaz and Laguna, 2006, Balaprakash et al., 2007, Bartz-Beielstein, 2006, Birattari et al., 2002, Hutter et al., 2007, 2009a,b, Nannen and Eiben, 2007]. Many successful studies involve tuning discrete optimization algorithms [Balaprakash et al., 2007, Birattari et al., 2002, Hutter et al., 2009b] and also continuous optimization algorithms [Hutter et al., 2009a, Montes de Oca et al., 2011, Smit and Eiben, 2009, Yuan et al., 2010].

In the design of algorithm frameworks, the entries of algorithmic components can be designed as tunable parameters. The combination of algorithmic components via automatic algorithm configuration tools has also shown its extraordinary potential for obtaining new state-of-the-art algorithms. For instance, KhudaBukhsh et al. [2009] proposed the SATenstein framework and instantiated a new state-of-the-art local search algorithm for the SAT problem. López-Ibáñez and Stützle [2010], López-Ibáñez and Stützle [2012] automatically configured a multi-objective ACO algorithm that outperformed previously proposed multi-objective ACO algorithms for the bi-objective traveling salesman problem. Dubois-Lacoste et al. [2011] configured new state-of-the-art algorithms for five variants of multi-objective flow-shop problems. More recently, the ideas behind the combination of algorithm frameworks and automatic algorithm configuration techniques have been extended to the programming by optimization paradigm [Hoos, 2012].

2.4.1 Iterated F-Race

In this thesis, we employ Iterated F-Race [Birattari et al., 2010], a racing algorithm for algorithm configuration that is included in the `irace` package [López-Ibáñez et al., 2011] for automatic algorithm configuration and parameter tuning. Iterated F-Race is an algorithm that repeatedly applies F-Race [Birattari et al., 2002] to a set of candidate configurations that are generated via a sampling mechanism that intensifies the search around the best found configurations. The generated candidate configurations then perform a "race". At each step of the race, each surviving candidate configuration is run on one training problem. Poor performing candi-

date configurations are eliminated from the race based on the result of statistical tests. To this aim, the results of each surviving configuration on the same training problem is ranked. Note that this ranking corresponds to blocking in statistical tests since ranks are determined on a same training problem. In fact, ranking is useful in the context of continuous and mixed-variable function optimization to account for the different ranges of the values of the benchmark functions. Without ranking, few functions with large values would dominate the evaluation of the algorithm performance. Based on the obtained ranks, the Friedman two-way analysis of variance by ranks checks whether sufficient statistical evidence is gathered that indicates that some configurations behave differently from the rest. If the null hypothesis of the Friedman test is rejected, Friedman post-tests are used to eliminate the statistically worse performing candidates. The `irace` automatic configuration tool handles four parameter types: continuous (r), integer (i), ordinal (o) and categorical (c).

With Iterated F-race, algorithm parameters are tuned using a machine learning approach in which an algorithm is trained on a set of problem instances and later tested on another set. In this thesis, the performance measure for tuning is the error of the objective function value obtained by the tuned algorithm after a certain number of function evaluations. The error value is defined as $f(S) - f(S^*)$, where S is a candidate solution and S^* is an optimal solution. In [López-Ibáñez et al., 2012], we used the hypervolume as performance measure for tuning, a well-known quality measure in multi-objective optimization, to assign a single numerical value to the anytime behavior of an algorithm's run.

2.4.2 Tuning methodology

This thesis is not the first to try to further tune continuous optimization algorithm using automatic algorithm configuration tools. A PSO algorithm was used by Bartz-Beielstein [2006] as a benchmark algorithm to be tuned for evaluating Sequential Parameter Optimization (SPO). CMA-ES was used by Hutter et al. [2009a] as a benchmark algorithm to be tuned for evaluating SPO$^+$, their improved variant of SPO [Bartz-Beielstein, 2006]. Following earlier work on SPO, they tuned CMA-ES only on individual functions, thus, in this sense "overtuning" CMA-ES on individual functions. (One has to remark, however, that the interest of Hutter et al. [2009a] was to evaluate SPO and the improved variant SPO$^+$ rather than proposing a new, generally improved parameter setting for CMA-ES.) Another attempt of tuning a variant of CMA-ES was made by Smit and Eiben

[2010]. From a tuning perspective, it should be mentioned that they tuned their algorithm on multiple functions. For the tuning, they used the CMA-ES variant they used on each ten dimensional function, then they were running the tests with the tuned algorithm on the same functions of dimension ten. Same was done by Montes de Oca et al. [2011] for re-designing and tuning a PSO algorithm for large scale continuous function optimization.

In the methodological approach to tuning continuous optimization algorithms, this thesis tries to avoid a bias of the results obtained due to potentially overtuning [Birattari, 2009] the algorithm on the same benchmark functions as those on which the algorithm is tested. As such, this gives a better assessment of the potential for what concerns the tuning of continuous optimizers as we have a separation between tuning and test set. In this thesis, the tuning and test sets involve three degrees of separation as follows:

- A separation of dimensionality between tuning and test set. For instance, we tuned algorithms on small dimensional functions and later tested them on (much) larger dimensional versions of the *same* functions. [Liao et al., 2011c, Montes de Oca et al., 2011].

- A separation of the functions used in the tuning and test set. For instance, we tuned algorithms on small dimensional functions and later tested them on a *different* set of functions. [Liao and Stützle, 2013, Liao et al., 2013a, López-Ibáñez et al., 2012].

- A separation between tuning functions and real world problems. For instance, we tuned algorithms on artificial benchmark functions and later tested them on *real-word* optimization problems. [Liao et al., 2013b]

The differences in the applied approaches to tuning with respect to the separation of tuning and test sets is summarized in Figure 2.2.

2.5 Summary

In this chapter, we described the models of continuous optimization problems, and reviewed the main optimization techniques (i.e., local search algorithms and metaheuristic based algorithms) and discussed the main benchmark function sets that are relevant for this thesis. We also described the models of mixed discrete-continuous optimization problems, and classified the main approaches to tackle mixed variable problems from the literature. We explained in some more detail

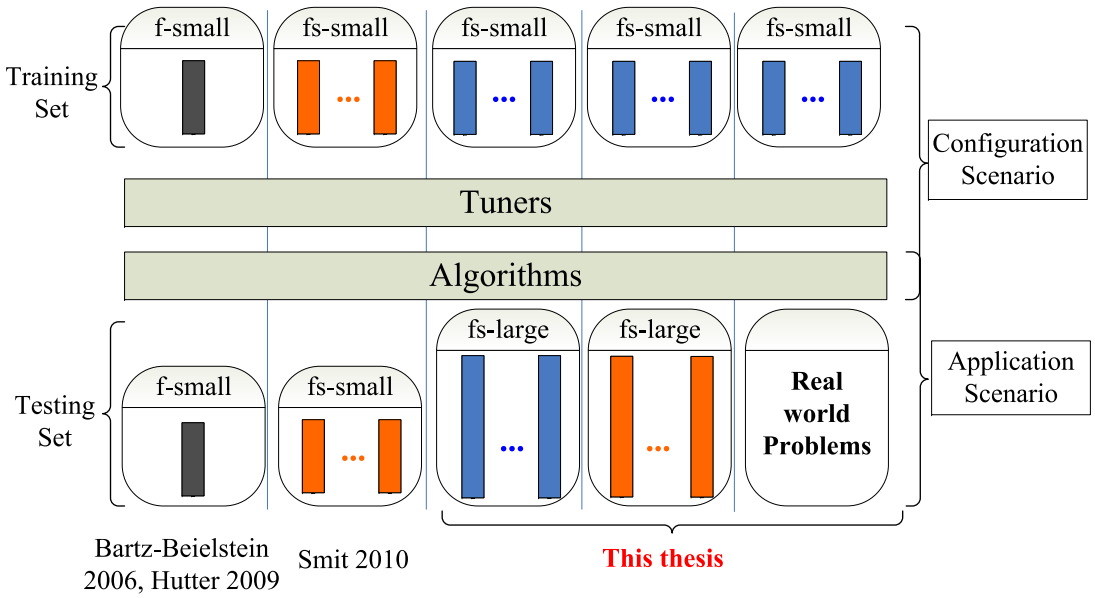


Figure 2.2: Summary of the methodological approach to tuning continuous optimization algorithms from few recent articles [Bartz-Beielstein, 2006, Hutter et al., 2009a, Smit and Eiben, 2010] and this thesis. The approaches differ in the usage of a single versus multiple functions and the degree of separation between tuning and test set. The difference of colors between tuning and test set indicates a separation of the functions used in the tuning and test set.

the two basic algorithms $ACO_{\mathbb{R}}$ and CMA-ES, on which large parts of this thesis rely on. We described an automatic algorithm configuration technique and the tuning methodologies we used in the thesis.

Chapter 3

UACOR: A unified ACO algorithm for continuous optimization

Metaheuristics are a family of optimization techniques that have seen increasingly rapid development and have been applied to numerous problems over the past few years. A prominent metaheuristic is ant colony optimization (ACO). ACO is inspired by the ants' foraging behavior and it was first applied to solve discrete optimization problems [Dorigo and Stützle, 2004, Dorigo et al., 1991, 1996]. Recently, adaptations of ACO to continuous optimization problems were introduced. Socha and Dorigo [2008] proposed one of the now most popular ACO algorithms for continuous domains, called $\text{ACO}_{\mathbb{R}}$. It uses a solution archive as a form of pheromone model for the derivation of a probability distribution over the search space. Leguizamón and Coello [2010] proposed an extension of $\text{ACO}_{\mathbb{R}}$, called $\text{DACO}_{\mathbb{R}}$, that had the goal of better maintaining diversity during the search. Subsequently, we proposed $\text{IACO}_{\mathbb{R}}$-LS, an incremental ant colony algorithm with local search for continuous optimization [Liao et al., 2011b]. $\text{IACO}_{\mathbb{R}}$-LS uses a growing solution archive as an extra search diversification mechanism and a local search to intensify the search.

In this chapter, we propose an ACO algorithm for continuous optimization, which combines algorithmic components from $\text{ACO}_{\mathbb{R}}$, $\text{DACO}_{\mathbb{R}}$ and $\text{IACO}_{\mathbb{R}}$-LS. We call this algorithm UACOR. From UACOR, one can instantiate the original $\text{ACO}_{\mathbb{R}}$, $\text{DACO}_{\mathbb{R}}$ and $\text{IACO}_{\mathbb{R}}$-LS algorithms by using specific combinations of the available algorithmic components and parameter settings. However, one also can obtain combinations of the algorithmic components that are different from any of the already proposed ones, that is, one may instantiate new, continuous ACO algorithms. UACOR's design makes the automatic generation of high performance continuous ACO algorithms possible through the use of automatic algorithm configuration tools.

The combination of algorithmic components via automatic algorithm configuration tools has already shown its high potential for obtaining new state-of-the-art algorithms for combinatorial optimization problems. For example, Dubois-Lacoste et al. [2011], KhudaBukhsh et al. [2009], López-Ibáñez and Stützle [2012] configured new state-of-the-art algorithms for the SAT problem, the bi-objective traveling salesman problem and the multi-objective flow-shop problem, respectively.

UACOR is a highly configurable algorithm, and thus it can also be considered a framework from which new state-of-the-art ACO algorithms for continuous optimization can be derived. In this chapter, we show how this can be done through the use of an automatic configuration tool. In particular, we use Iterated F-race [Birattari et al., 2010] as implemented in the `irace` package [López-Ibáñez et al., 2011]. As the set of training benchmark functions we use low dimensional versions of the functions used in the SOCO and CEC'05 benchmark sets. We configure two new ACO variants; UACOR-s is configured on the SOCO benchmark (the -s suffix stands for SOCO) set and UACOR-c on the CEC'05 benchmark set (the -c suffix stands for CEC). UACOR-s and UACOR-c are then tested on higher dimensional versions of the SOCO and CEC'05 benchmark functions. The results show that (i) UACOR-s performs superior or competitive to all the 16 algorithms benchmarked on the SOCO function set and that (ii) UACOR-c performs competitive to IPOP-CMA-ES [Auger and Hansen, 2005] and superior to other five recent state-of-the-art algorithms benchmarked on the CEC'05 function set. These experimental results illustrate the high potential of ACO algorithms for continuous optimization. To the best of our knowledge, this is also the first trial where a continuous optimizer framework is automatically configured. Previous applications of automatic configuration for continuous optimization were applied to already fully designed algorithms. Finally we re-design UACOR by implementing CMA-ES [Hansen and Ostermeier, 1996, 2001, Hansen et al., 2003] as an alternative local search procedure, which further improves the performance of UACOR.

The chapter is organized as follows. Section 3.1 introduces ACO for continuous domains, reviews the three continuous ACO algorithms underlying UACOR, and identifies their algorithmic components in a component-wise view. Section 3.2 describes UACOR. In Section 3.3, we automatically configure UACOR to instantiate UACOR-s and UACOR-c and in Section 3.4, we evaluate their performance. The performance of a re-designed and improved UACOR is shown in Section 3.5 and conclude in Section 3.6.

3.1 ACO algorithms for continuous optimization

The UACOR framework build upon and extends the $\text{ACO}_{\mathbb{R}}$ algorithm for continuous optimization. In fact, from UACOR we can also instantiate directly the $\text{ACO}_{\mathbb{R}}$ algorithm. The main details of the $\text{ACO}_{\mathbb{R}}$ algorithm have already been described in Section 2.3.1 and we refer the reader to the details described there. Next we describe $\text{DACO}_{\mathbb{R}}$ [Leguizamón and Coello, 2010] and $\text{IACO}_{\mathbb{R}}$-LS [Liao et al., 2011b], two more recent ACO algorithms for continuous optimization.

$\text{DACO}_{\mathbb{R}}$

Different from $\text{ACO}_{\mathbb{R}}$, $\text{DACO}_{\mathbb{R}}$ keeps the number of ants (Na) equal to the solution archive size k and each of the Na ants constructs at each algorithm iteration a new solution. A further difference of $\text{DACO}_{\mathbb{R}}$ with respect to $\text{ACO}_{\mathbb{R}}$ is the specific choice rule for the guiding solution S_{guide}. With a probability $Q_{best} \in [0, 1]$, ant j chooses as S_{guide} the best solution, S_{best}, in the archive; with a probability $1-Q_{best}$, it chooses as S_{guide} the solution S_j. A new solution is generated in the same way as in $\text{ACO}_{\mathbb{R}}$, and then compared to S_j (independently of whether S_{best} or S_j was chosen as guiding solution). If the newly generated solution is better than S_j, it replaces S_j in the archive; otherwise it is discarded. This replacement strategy is different from the one used in $\text{ACO}_{\mathbb{R}}$ in which all the solutions in the archive and all the newly generated ones compete.

$\text{IACO}_{\mathbb{R}}$-LS

$\text{IACO}_{\mathbb{R}}$-LS's main distinctive features are a solution archive whose size increases over time to enhance the algorithm's search diversification, and a local search procedure to enhance its search intensification [Liao et al., 2011b]. Additionally, $\text{IACO}_{\mathbb{R}}$-LS uses a different choice rule for the guiding solution than $\text{ACO}_{\mathbb{R}}$. At each algorithm iteration of $\text{IACO}_{\mathbb{R}}$-LS, the best solution in the archive S_{best} is chosen as the guiding solution S_{guide} with a probability equal to the value of a parameter $EliteQ_{best} \in [0, 1]$; with a probability of $1-EliteQ_{best}$, each solution in the archive is used as S_{guide} to generate a new solution. With this choice rule, either only one new solution is constructed by an "elite" guiding solution or k new solutions are constructed by k ants at each algorithm iteration. This latter aspect makes the action choice roles of $\text{IACO}_{\mathbb{R}}$-LS also different from the one used in $\text{DACO}_{\mathbb{R}}$. Each new solution is constructed in the same way as in $\text{ACO}_{\mathbb{R}}$. Finally, S_{guide} and the newly generated solution are compared. If the newly generated solution is better

than S_{guide}, it replaces it in the archive; otherwise it is discarded.

IACO$_{\mathbb{R}}$-LS initializes the archive with *InitAS* solutions. Every *GrowthIter* iterations a new solution is added to the archive until a maximum archive size is reached. The new solution is initialized as follows:

$$S_{\text{new}} = S_{\text{rand}} + \text{rand}(0,1)(S_{\text{best}} - S_{\text{rand}})\,, \tag{3.1}$$

where S_{rand} is a random solution and $\text{rand}(0,1)$ is a random number uniformly distributed in $[0,1)$.

In IACO$_{\mathbb{R}}$-LS, the local search procedure is called at each iteration and for *LsIter* iterations. If the local search succeeds in improving the solution from which it is called, this improved solution replaces the original solution in the archive. The maximum number of times the local search procedure is called from the same initial solution is limited to *LsFailures* calls. The initial solution for the local search is chosen as follows. The best solution is deterministically chosen as the initial solution if it has been called less than *LsFailures* times. Otherwise a random solution from the archive is chosen as the initial solution, excluding all those solutions for which the number of times they have been chosen as initial solutions is equal to *LsFailures*.

The step size to be used in the local search procedure is set as follows. First, a solution different from the best one is chosen randomly in the archive. The step size is then set to the maximum norm ($||\cdot||_\infty$) of the vector that separates this random solution from the best solution. As a result, step sizes tend to decrease upon convergence of the algorithm and, in this sense, the step sizes are chosen adaptively to focus the local search around the best-so-far solution. In our previous experiments, Powell's conjugate directions set [Powell, 1964] and Lin-Yu Tseng's Mtsls1 [Tseng and Chen, 2008] local search methods (see also Section 2.1.1) have shown very good performance.

IACO$_{\mathbb{R}}$-LS uses a default restart mechanism that restarts the algorithm and re-initializes the archive of size *InitAS* with the best-so-far solution S_{best} and *InitAS*-1 random solutions. The restart criterion is the number of consecutive iterations, *StagIter*, with a relative solution improvement lower than a threshold ϵ. IACO$_{\mathbb{R}}$-LS also integrates a second restart mechanism, which consists in restarting and initializing a new initial archive of size *RestartAS* (*RestartAS* is a parameter different from *InitAS*) with S_{best} in the current archive and *RestartAS*-1 solutions that are initialized at positions biased around S_{best}; these positions are defined by $S_{\text{best}} + 10^{Shakefactor} \cdot (S_{\text{best}} - S_{\text{rand}})$. The restart criterion is the number of consec-

utive iterations, *StagIter*, with a relative solution improvement percentage lower than a certain threshold $10^{StagThresh}$.

3.1.1 Algorithmic components

We define several algorithmic components for UACOR by abstracting the particular design alternatives taken in $ACO_{\mathbb{R}}$, $DACO_{\mathbb{R}}$ and $IACO_{\mathbb{R}}$-LS. This results in seven main groups of algorithmic components, which are described next, before detailing the outline of UACOR.

1. **Mode.** Two alternative UACOR modes, called *DefaultMode* and *EliteMode*, are identified. *DefaultMode* consists in deploying a number of ants in each algorithm iteration to construct solutions. *EliteMode* allows in each algorithm iteration to deploy only one "elite" ant with a probability of $EliteQ_{best} \in [0, 1]$. The "elite" ant selects S_{best} in the archive as S_{guide} to construct a new solution.

2. **Number of ants.** Two design choices for defining the number of ants deployed are identified. *Na* defines the number of ants as an independent parameter ($Na \leq k$) while *NaIsAS* defines the number of ants to be equal to k, the size of the solution archive.

3. **Choice of guiding solution.** This algorithmic component chooses how to select S_{guide} to sample new solutions. Three design choices are identified: (i) S_{best} is selected as S_{guide} with a probability $Q_{best} \in [0, 1]$; (ii) S_{guide} is probabilistically selected from the solutions in the archive depending on their weight; (iii) solution S_l is selected as S_{guide}, where l is the index of the currently deployed ant.

4. **Update of solution archive.** The update of the solution archive concerns the replacement of solutions in the archive. We identified three design choices. A parameter *RmLocalWorse* defines whether UACOR globally removes the *Na* worst solutions among all $k+Na$ solutions, or whether UACOR makes the decision about the acceptance of S_l locally. In the latter case, we use a parameter *SnewvsGsol* to decide whether the solution generated by ant l is compared with S_{guide} or with the previous l-th solution to remove the worse one.

5. **Local search.** We consider three options for the use of a local search procedure. If parameter *LsType* is set to F (for *false*), no local search procedure

is used. Otherwise, *LsType* invokes either Powell's conjugate directions set [Powell, 1964] or Mtsls1 [Tseng and Chen, 2008]. Both local search procedures use a dynamic calling strategy and an adaptive step size, which follow the choices taken for $\text{IACO}_{\mathbb{R}}$-LS.

6. **Incremental archive size.** The possibility of incrementing the archive size is considered. If parameter *IsIncrement* is set to F, the incremental archive mechanism is not used. Otherwise, if *IsIncrement* is set to T (for *true*), UACOR invokes the incremental archive mechanism.

7. **Restart mechanism.** Three options for the restart mechanism are identified. If parameter *RestartType* is set to F, the restart mechanism is not used. Otherwise, *RestartType* invokes either of the two restart mechanisms, which are introduced in $\text{IACO}_{\mathbb{R}}$-LS. They are labeled as 1st and 2nd, respectively.

Table 3.1 summarizes the algorithmic components defined above and their options. Some algorithmic components are only significant for a specific value of other components. We discuss the connection between these algorithmic components in Section 3.2.

Table 3.1: Algorithmic components of UACOR

Algorithm Components	Options	Description
Mode	{*DefaultMode*, *EliteMode*}	Definition of UACOR mode
AntsNumber	{*Na*, *NaIsAS*}	Definition of the number of ants deployed
SolutionConstructions	sample the neighborhood of the solution component of S_{guide}	Using a Gaussian PDF
	select S_{best} in a proportion of $Q_{best} \in [0, 1]$, select probabilistically by weights, select S_l for the ant l	How S_{guide} is selected from the solution archive
SolutionArchiveUpdate	remove by globally ranking, remove by comparing with S_{guide}, remove by comparing with S_l	How Na worse solutions are removed from the archive
LocalSearch	{F, Powell, Mtsls1}	Definition of a local search procedure
IncrementalArchive	{F, True}	Definition of an incremental archive mechanism
RestartMechanism	{F, 1st, 2st}	Definition of a restart mechanism

3.2 UACOR

The three ACO algorithms described in the previous section as well as many others that may result from the combination of their components are subsumed under the general algorithmic structure provided by UACOR. In this section, we describe the connections of the algorithmic components of UACOR by a flowchart and show how from UACOR we can instantiate the algorithms $\text{ACO}_{\mathbb{R}}$, $\text{DACO}_{\mathbb{R}}$ and $\text{IACO}_{\mathbb{R}}$-LS. The flowchart of UACOR is given in Fig. 3.1. The related parameters are given in Table 3.2. Some settings take effect in the context of certain values of other settings.

UACOR starts by randomly initializing and evaluating the solution archive of size *InitAS*. Next, UACOR selects a mode, which can be either the default mode or an elite mode.

We first describe the default mode, which is invoked if parameter *DefaultMode* is set to T (*true*). At each iteration, *Na* new solutions are probabilistically constructed by *Na* ants (recall that an ant in our case is the process through which a solution is generated). If the parameter *NaIsAS* is set to T, the number of ants is kept equal to the size of the solution archive. If the parameter *NaIsAS* is set to F (*false*), a parameter *Na*, $Na \leq k$, is activated. Each ant uses a choice rule for the guiding solution. The parameter $Q_{best} \in [0, 1]$ controls the probability of using S_{best} as S_{guide}. With a probability $1 - Q_{best}$, S_{guide} is selected in one of two different ways. If parameter *WeightGsol* is T, S_{guide} is probabilistically selected from the solutions in the archive by their weights as defined by Equation 2.1. Otherwise, solution S_l (l is associated with the index of the current ant to be deployed) is chosen as S_{guide}. Once S_{guide} is selected, a new solution is generated. This process is repeated for each of the *Na* ants. Next, UACOR updates the solution archive by removing *Na* solutions. If parameter *RmLocalWorse* is F, UACOR removes the *Na* worst solutions among all the $k + Na$ solutions as in $\text{ACO}_{\mathbb{R}}$. If parameter *RmLocalWorse* is T, one of two possibilities is considered. If parameter *SnewvsGsol* is T, each newly generated solution is compared to the corresponding S_{guide} to remove the worse one; otherwise, it is compared to the corresponding S_l to remove the worse one. Finally, a new solution archive is generated.

The elite mode is invoked if parameter *DefaultMode* is set to F. The elite mode at each algorithm iteration deploys only one "elite" ant. With a probability $EliteQ_{best}$, $0 \leq EliteQ_{best} \leq 1$, it selects S_{best} in the archive as S_{guide}. If the newly generated solution is better than this S_{best}, it replaces it in the solution archive; with a probability $1 - EliteQ_{best}$ the solution construction follows the default mode.

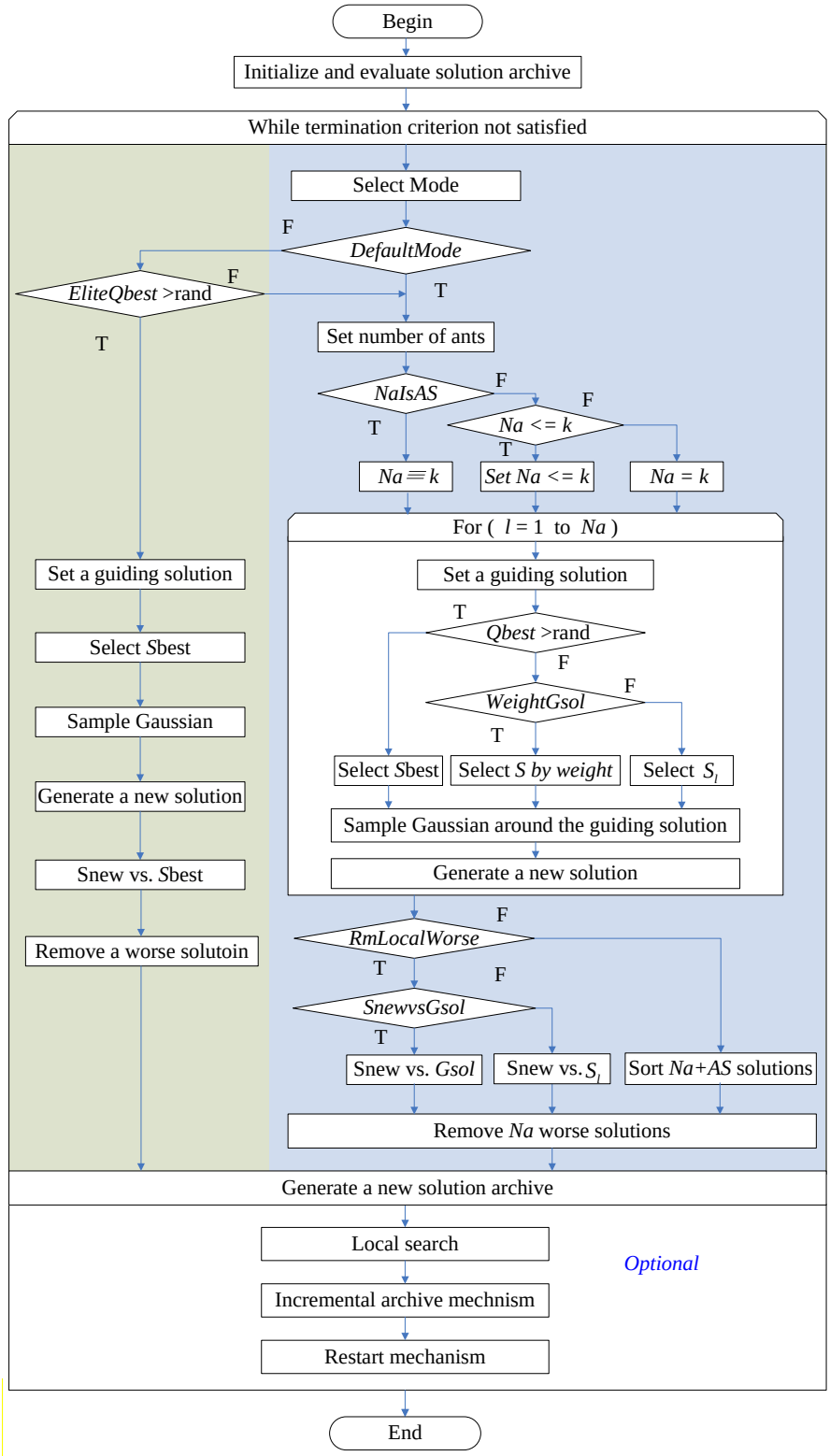


Figure 3.1: A flowchart for UACOR. For an explanation of the parameters we refer to the text.

After updating the solution archive, UACOR sequentially considers three procedures. These are a local search procedure, a mechanism for increasing the archive size and a restart mechanism, respectively. Recall that the options of these procedures were described in Section 3.1.

We use a simple penalty mechanism to handle bound constraints for UACOR. We use

$$P(\boldsymbol{x}) = fes \cdot \sum_{i=1}^{D} Bound(x_i)\,, \tag{3.2}$$

where $Bound(x_i)$ is defined as

$$Bound(x_i) = \begin{cases} 0, & \text{if } x_{\min} \leq x_i \leq x_{\max} \\ (x_{\min} - x_i)^2, & \text{if } x_i < x_{\min} \\ (x_{\max} - x_i)^2, & \text{if } x_i > x_{\max} \end{cases} \tag{3.3}$$

and x_{min} and x_{max} are the minimum and maximum limits of the search range, respectively, and *fes* is the number of function evaluations that have been used so far. For avoiding that the final solution is outside the bounds, the bound constraints are enforced by clamping the final solution S to the nearest solution on the bounds, resulting in solution S', if S violates some bound constraints. If S' is worse than the best feasible solution found in the optimization process, S' is replaced by it.

3.3 Automatic algorithm configuration

We automatically configure UACOR before evaluating its performance on benchmark functions. As the benchmark functions, we employ the 19 functions from the SOCO benchmark set [Herrera et al., 2010] (f_{soco1}-f_{soco19}) and the 25 functions from the CEC05 benchmark set [Suganthan et al., 2005] (f_{cec1}-f_{cec25}). Note that in both benchmark sets, the functions allow for different dimensionalities. These two benchmark sets have been chosen as they have become standard benchmark sets for testing continuous optimizers. The SOCO benchmark set was used in a special issue of the journal Soft Computing and it extends the benchmark sets of earlier benchmarking studies on the scaling behavior of continuous optimizers such as the one held at the CEC'08 conference. The CEC'05 benchmark set was used for a comparison of evolutionary optimizers at the special session on real parameter optimization of CEC'05 conference [Suganthan et al., 2005]. Classified by

function characteristics, the SOCO benchmark set consists of seven unimodal and 12 multimodal functions, or, four separable and 15 non-separable functions. The CEC'05 benchmark set consists of five unimodal and 20 multimodal functions, or, two separable and 23 non-separable functions. For a detailed description of the benchmark functions, we refer the reader to [Herrera et al., 2010, Suganthan et al., 2005], and to the description in Section 2.1.3.

In our experiments, we follow the termination conditions suggested for the SOCO and CEC benchmarks [Herrera et al., 2010, Suganthan et al., 2005] to make our results comparable to those of other papers. In particular, we use a maximum of $5\,000 \times D$ function evaluations for the SOCO functions, and $10\,000 \times D$ for the CEC'05 functions, where D is the dimensionality of a function.

For automatically configuring UACOR, we employ Iterated F-Race [Birattari et al., 2010], a method for automatic algorithm configuration that is included in the irace package [López-Ibáñez et al., 2011] and that was described in Section 2.4.1. In a nutshell, Iterated F-Race repeatedly applies F-Race to a set of candidate configurations. F-Race is a racing method that at each iteration applies all surviving candidate configurations to an instance of a combinatorial problem or a function in the continuous optimization case. If a candidate configuration is found to perform statistically worse than others (as determined by the Friedman two-way analysis of variance by ranks and its associated post-tests), it is eliminated from the race. F-race finishes when only one candidate survive or the allocated computation budget to the race is used. Iterated F-Race then samples new candidate configurations around the best candidate configurations found so far. The whole process is repeated for a number of iterations (hence the name Iterated F-Race).

The automatic configuration tool handles all parameter types of UACOR: continuous (r), integer (i) and categorical (c). The performance measure used for tuning is the error of the objective function value obtained by the tuned algorithm after a certain number of function evaluations. The error value is defined as $f(S) - f(S^*)$, where S is a candidate solution and S^* is the optimal solution. In the automatic tuning process, the maximum budget is set to 5 000 runs of UACOR. The settings of Iterated F-Race that we used in our experiments are the default [Birattari et al., 2010, López-Ibáñez et al., 2011]. We conduct automatic configuration for UACOR in two stages. In the first stage, we tuned UACOR on the SOCO training instances to instantiate UACOR-s. 19 SOCO benchmark functions of dimension 10 were sampled as training instances in a random order. In the second stage, we tuned UACOR on the CEC'05 training instances to instantiate UACOR-c. 25 CEC'05 benchmark functions of dimension 10 were sampled as

training instances in a random order.

The tuned parameter settings for both, UACOR-s and UACOR-c, are presented in the central and right part of Table 3.2. This table also gives the parameter settings for the UACOR's instantiations of $ACO_{\mathbb{R}}$, $DACO_{\mathbb{R}}$ and $IACO_{\mathbb{R}}$-Mtsls1. Their parameters were also automatically tuned as mentioned above for the SOCO and CEC'05 benchmark sets, respectively, and for these specific parameter configurations we again use the extensions '-s' and '-c' depending on the benchmark functions used for automatic configuration. Considering that UACOR-s does not use the restart mechanisms of UACOR after tuning and UACOR-c does, when tuning these three ACO algorithms on the SOCO training instances, we deploy them as proposed in the original literature; when tuning them on CEC'05 training instances, we extend them to use the restart mechanisms of UACOR to improve performance.

As a further illustration of the respective algorithm structures, we highlight UACOR-s and UACOR-c in the flowcharts of UACOR in Fig. 3.2 and 3.3. Both use *DefaultMode*, select S_{best} as S_{guide} with a probability $Q_{best} \in [0, 1]$, use Mtsls1 local search and the incremental archive mechanism. The parameter settings in which they differ, imply a more explorative search behavior of UACOR-c than that of UACOR-s. In fact, (i) UACOR-c sets the number of ants equal to the size of the solution archive while UACOR-s defines it as an independent parameter ($Na \leq k$); (ii) UACOR-c frequently chooses all solutions of the archive as S_{guide} (as in $DACO_{\mathbb{R}}$), while UACOR-s probabilistically selects S_{guide} based on its weight; (iii) UACOR-c makes a local acceptance decision comparing S_l to S_{guide}, while UACOR-s globally removes the Na worst solutions among all $k+Na$ solutions; (iv) UACOR-c uses a restart mechanism for diversifying the search while UACOR-s does not. Considering parameter values, UACOR-c has larger initial archive size and less iterations of local search exploitation, which is consistent with the idea of a strong search exploration than UACOR-s; the larger values of Q_{best} and *GrowthIter* would imply UACOR-c and UACOR-s differ. Similar remarks hold also for the settings of the '-c' and '-s' variants of $ACO_{\mathbb{R}}$, $DACO_{\mathbb{R}}$ and $IACO_{\mathbb{R}}$-Mtsls1. Note that the more explorative settings on the CEC'05 benchmark set are somehow in accordance with the perceived higher difficulty of this benchmark set than the SOCO set. In fact, in the CEC'05 benchmark set the best available algorithms fail to find quasi-optimal solutions much more frequently than in the SOCO benchmark function set.

Table 3.2: The left part of the table gives the list of parameter settings and their domains. Some settings are only significant for certain values of other settings. The parameter settings of the automatically configured algorithms are given in the central part and the right part, depending on which training set of benchmark functions was used for tuning.

Module	Para Name	Type	Domain	Tuning on SOCO				Tuning on CEC'05			
				$ACO_{\mathbb{R}}$-s	$DACO_{\mathbb{R}}$-s	$IACO_{\mathbb{R}}$-Mtsls1-s	UACOR-s	$ACO_{\mathbb{R}}$-c	$DACO_{\mathbb{R}}$-c	$IACO_{\mathbb{R}}$-Mtsls1-c	UACOR-c
Mode	*DefaultMode*	c	{T, F}	T	T	F	**T**	T	T	F	**T**
	EliteQ$_{best}$	r	[0, 1]	*	*	**0.0508**	*	*	*	**0.7974**	*
DefNants	*InitAS*	i	[20, 100]	**87**	**40**	**6**	**48**	**92**	**81**	**54**	**66**
	NaIsAS	c	{T, F}	F	T	T	**F**	F	T	T	**T**
	Na	i	[2, 20]	**2**	*	*	**16**	**14**	*	*	*
SolConstr	Q_{best}	r	[0, 1]	0	**0.1193**	0	**0.1895**	0	**0.1287**	0	**0.5351**
	WeightGsol	c	{T, F}	T	F	F	**T**	T	F	F	**F**
	q	r	(0, 1)	**0.2869**	*	*	**0.2591**	**0.09401**	*	*	*
	ξ	r	(0, 1)	**0.7187**	**0.6705**	**0.8782**	**0.6511**	**0.6998**	**0.7357**	**0.9164**	**0.6945**
SAUpdate	*RmLocalWorse*	c	{T, F}	F	T	T	**F**	F	T	T	**T**
	SnewvsGsol	c	{T, F}	*	F	T	*	*	F	T	**T**
LS	*LsType*	c	{F, Powell, Mtsls1}	F	F	Mtsls1	**Mtsls1**	F	F	Mtsls1	**Mtsls1**
	LsIter	i	[1, 100]	*	*	**85**	**84**	*	*	**39**	**28**
	LsFailures	i	[1, 20]	*	*	**1**	**8**	*	*	**3**	**7**
IncArch	*IsIncrement*	c	{T, F}	F	F	T	**T**	F	F	T	**T**
	GrowthIter	i	[1, 30]	*	*	**4**	**4**	*	*	**10**	**13**
RestartMech	*RestartType*	c	{F, 1st, 2nd}	F	F	1st	**F**	**2nd**	**2nd**	**2nd**	**2nd**
	StagIter	r	[1, 1000]	*	*	**18**	*	**939**	**313**	**6**	**11**
	StagThresh	r	[-15, 0]	*	*	*	*	**-3.386**	**-2.302**	**-3.041**	**-2.539**
	Shakefactor	r	[-15, 0]	*	*	*	*	**-4.993**	**-4.163**	**-0.04979**	**-0.02061**
	RestartAS	i	[2, 100]	*	*	*	*	**66**	**71**	**3**	**10**

* denotes the value of the parameter is not relevant for the corresponding algorithm.

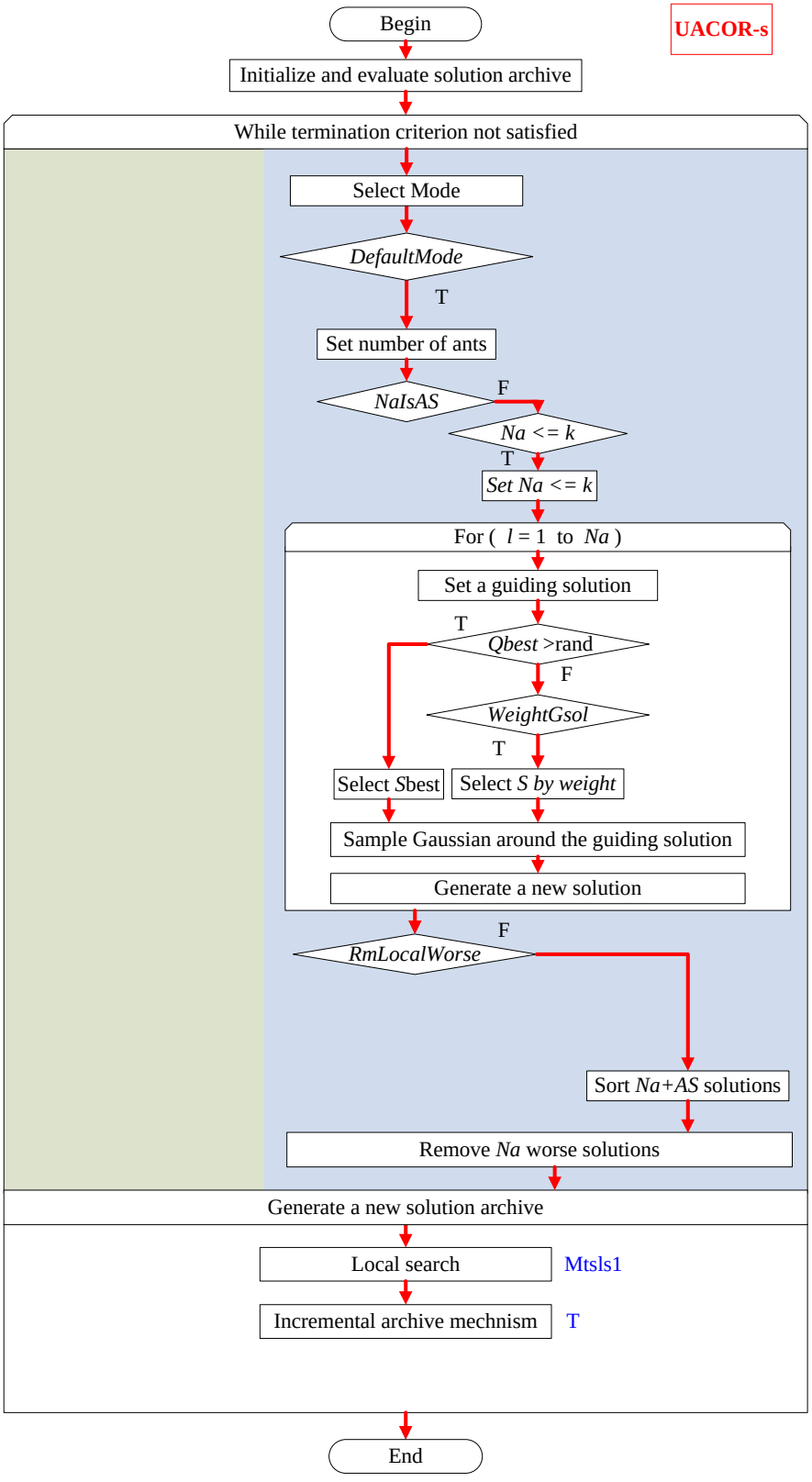


Figure 3.2: UACOR-s is highlighted in the flowchart of UACOR.

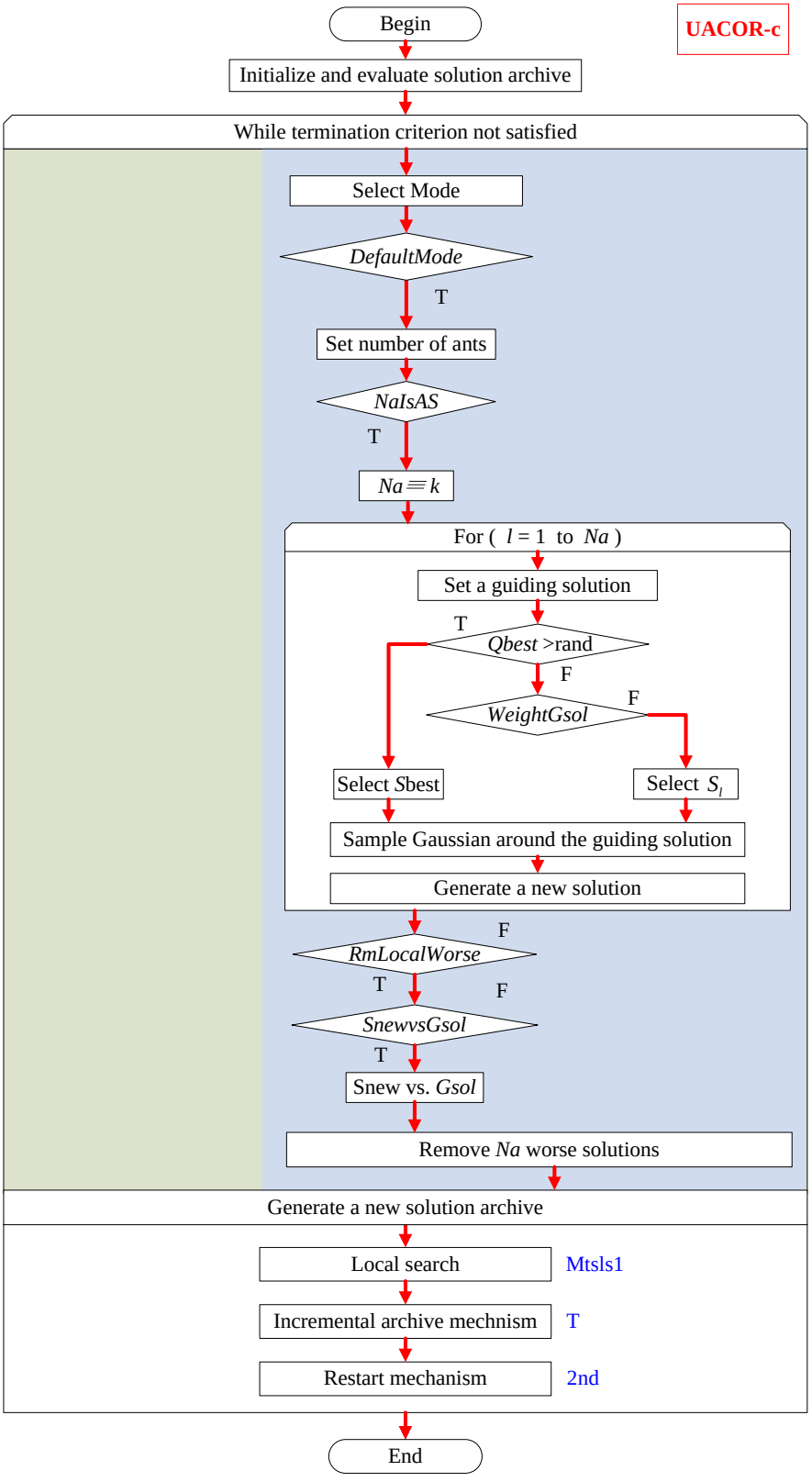


Figure 3.3: UACOR-c is highlighted in the flowchart of UACOR.

3.4 Algorithm evaluation

In this section, we evaluate UACOR-s and UACOR-c on the 19 SOCO benchmark functions of dimension 100 and 25 CEC'05 benchmark functions of dimensions 30 and 50. Each algorithm was independently run 25 times on each function. Whenever a run obtains a new best error value, we record the number of function evaluations used, and the new best error value. Following the rules of the SOCO algorithm comparison, error values lower than 10^{-14} are approximated to 10^{-14} (10^{-14} is referred as optimum threshold for SOCO functions). For CEC'05 functions, error values lower than 10^{-8} are approximated to 10^{-8} (10^{-8} is the optimum threshold for CEC'05 functions). On each benchmark function of each dimensionality we compute the average error obtained by an algorithm. These average errors on all test functions in each benchmark set (SOCO or CEC'05) are then used to compare the algorithms' performance. To analyze the results we first use a Friedman test at the 0.05 α-level to determine whether there are significant differences among the algorithms compared [Conover, 1998]. In fact, in all cases the null hypothesis of equal performance is rejected and we then determine the significance of the difference between the algorithms of interest based on the computed minimum difference between the sum of the ranks that is statistically significant.

Experiments on SOCO benchmark set

First, we compare UACOR-s with the three ACO algorithms, $\text{ACO}_{\mathbb{R}}$-s, $\text{DACO}_{\mathbb{R}}$-s and $\text{IACO}_{\mathbb{R}}$-Mtsls1-s. The left plot of Figure 3.4 shows that UACOR-s improves upon these ACO algorithms on the distribution of average errors across the 19 SOCO benchmark functions. In particular, UACOR-s performs statistically significantly better than $\text{ACO}_{\mathbb{R}}$-s and $\text{DACO}_{\mathbb{R}}$-s. This test is based on the average error values that are reported in Table 3.3. In fact, on 14 of the 19 functions the average error obtained by UACOR-s is below the optimum threshold, while for $\text{ACO}_{\mathbb{R}}$-s, $\text{DACO}_{\mathbb{R}}$-s and $\text{IACO}_{\mathbb{R}}$-Mtsls1-s such low average error values are only obtained 0, 1, and 8 times, respectively. (The main responsible for the large differences between the performance of $\text{ACO}_{\mathbb{R}}$-s and $\text{DACO}_{\mathbb{R}}$-s on one side and UACOR-s and $\text{IACO}_{\mathbb{R}}$-Mtsls1-s on the other side is due to the usage or not of a local search procedure to improve candidate solutions.) The larger number of optimum thresholds reached also is the reason why UACOR-s has a lower 75th percentile than $\text{ACO}_{\mathbb{R}}$-Mtsls1-s. Only on three functions, on which UACOR-s does not reach the optimum threshold, it obtains slightly worse average errors than $\text{IACO}_{\mathbb{R}}$-Mtsls1-s.

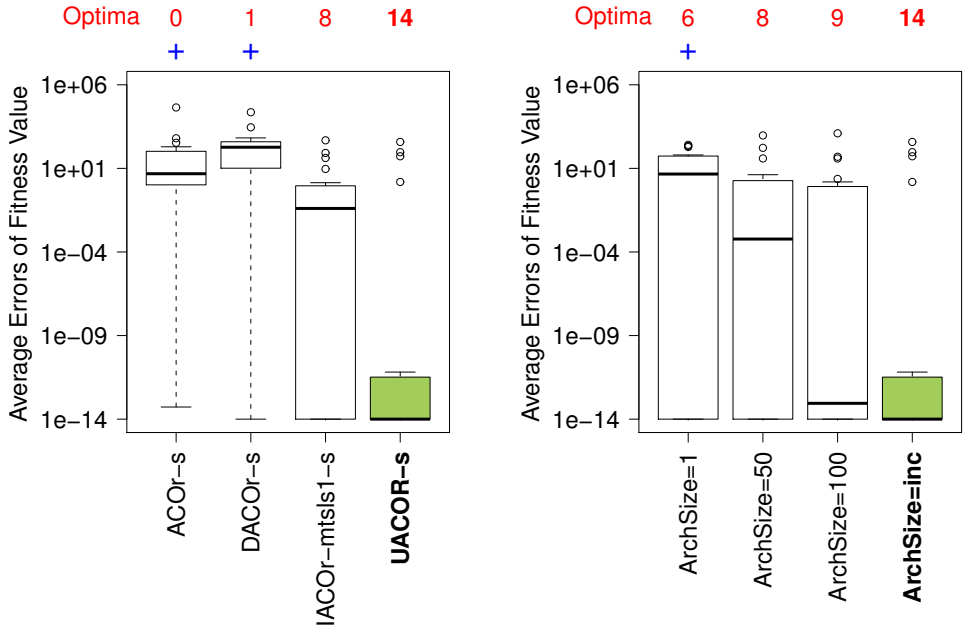


Figure 3.4: The box-plots show the distribution of the average errors obtained on the 19 SOCO benchmark functions of dimension 100. The left plot compares the performance of UACOR-s with $ACO_{\mathbb{R}}$-s, $DACO_{\mathbb{R}}$-s and $IACO_{\mathbb{R}}$-Mtsls1-s. The right plot shows the benefit of the incremental archive size used in UACOR-s. A + symbol on top of each box-plot denotes a statistically significant difference at the 0.05 α-level between the results obtained by the indicated algorithm and those obtained with UACOR-s. The absence of a symbol means that the difference is not statistically significant. The numbers on top of a box-plot denote the number of the averages below the optimum threshold 10^{-14} found by the indicated algorithms.

As a next step, we investigate the benefit of the incremental archive mechanism used by UACOR-s when compared to a fixed archive size. The right boxplot of Figure 3.4 shows that UACOR-s performs more effective than with archive sizes fixed to 1, 50 and 100, respectively. (Note that for an archive size one, the resulting algorithm is actually an iterated Mtsls1 local search algorithm [Tseng and Chen, 2008].) The differences are statistically significant for the archive sizes 1 and the average errors of UACOR-s obtain the largest number of times the optimum threshold number (14 versus 6, 8 and 9, respectively).

Finally, we compare UACOR-s with all 13 candidate algorithms published in the SOCO special issue and to the three algorithms that were chosen as reference algorithms.[1] Figure 3.5 shows that UACOR-s performs statistically significantly better than 10 other algorithms. Recall that IPOP-CMA-ES [Auger and Hansen,

[1]Information about these 16 algorithms is available at `http://sci2s.ugr.es/eamhco/CFP.php`

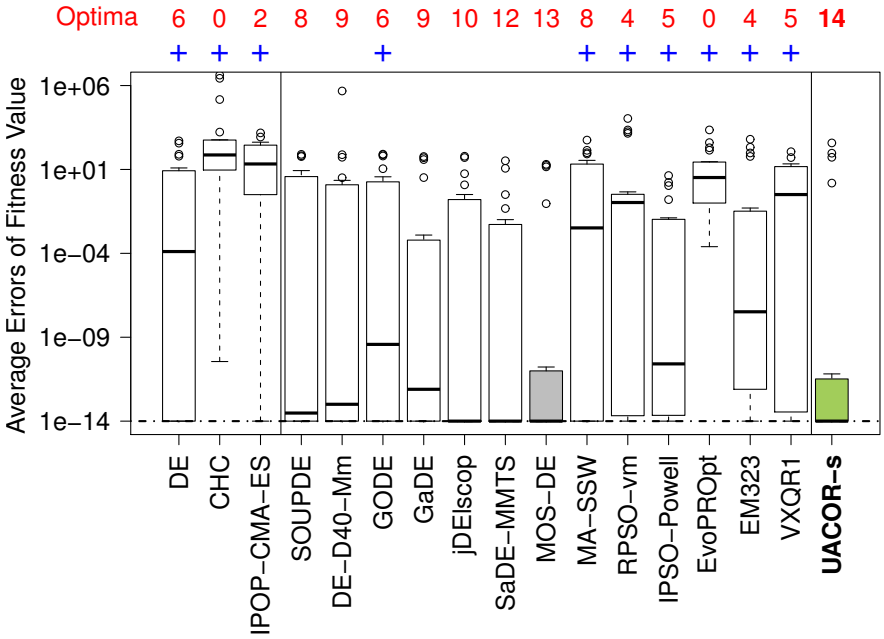


Figure 3.5: The box-plots show the distribution of the average errors obtained on the 19 SOCO benchmark functions of dimension 100. The results obtained by the three reference algorithms (left), 13 algorithms (middle) published in SOCO and UACOR-s (right) are shown on the left plot. The line at the bottom of the boxplot represents the optimum threshold (10^{-14}). A + symbol on top of the two box-plot denotes a statistically significant difference at the 0.05 α-level between the results obtained with the indicated algorithm and those obtained with UACOR-s detected with a Friedman test and its associated post test on the 17 algorithms. The absence of a symbol means that the difference is not significant. The numbers on top of a box-plot denote the number of averages below the optimum threshold 10^{-14} found by the indicated algorithms.

2005] is considered to be a representative of the state-of-the-art for continuous optimization and MA-SSW [Molina et al., 2010b, 2011] was the best performing algorithm at the CEC'2010 competition on high-dimensional numerical optimization. UACOR-s performs statistically significantly better than these two algorithms. The best performing algorithm from the SOCO competition is MOS-DE [LaTorre et al., 2011], an algorithm that combines differential evolution and the Mtsls1 local search algorithm. It is noteworthy that UACOR-s performs competitive to MOS-DE. Although UACOR-s does not obtain on more functions lower average errors than MOS-DE than vice versa, UACOR-s gives on more functions the zero threshold (14 versus 13).

Experiments on the CEC'05 benchmark set

We next evaluate UACOR-c on the CEC'05 benchmark set of dimension 30 and 50. Tables 3.4 and 3.5 show the average error values across the 25 CEC'05 benchmark functions obtained by UACOR-c, $ACO_{\mathbb{R}}$-c, $DACO_{\mathbb{R}}$-c, $IACO_{\mathbb{R}}$-Mtsls1-c, IPOP-CMA-ES [Auger and Hansen, 2005] and other five recent state-of-the-art algorithms.

Table 3.4 shows that UACOR-c gives across the 30 and 50 dimensional problems, on more functions lower average errors than $ACO_{\mathbb{R}}$-c, $DACO_{\mathbb{R}}$-c and $IACO_{\mathbb{R}}$-Mtsls1-c than vice versa. Considering the average error values across all these CEC'05 benchmark functions, UACOR-c performs statistically significantly better than $DACO_{\mathbb{R}}$-c and $IACO_{\mathbb{R}}$-Mtsls1-c. It is important to highlight the following two observations. First, UACOR-c significantly outperforms $IACO_{\mathbb{R}}$-Mtsls1-c on the CEC'05 benchmarks while UACOR-s was superior to $IACO_{\mathbb{R}}$-Mtsls1-s but without statistical significance. Second, the opposite happens with $ACO_{\mathbb{R}}$: $ACO_{\mathbb{R}}$-c performs roughly on par with UACOR-c but $ACO_{\mathbb{R}}$-s is significantly outperformed by UACOR-s on the SOCO benchmark set. (Recall that also $ACO_{\mathbb{R}}$ and $IACO_{\mathbb{R}}$-Mtsls1 were tuned for each of the benchmark sets.) In fact, the flexibility of UACOR makes it adaptable to each of the benchmark sets and allows it to outperform other available ACO algorithms.

Of particular interest is the comparison between UACOR-c and IPOP-CMA-ES, the data of which are taken from the literature [Auger and Hansen, 2005]. The latter is an acknowledged state-of-the-art algorithm on the CEC'05 benchmark set. UACOR-c shows competitive performance to IPOP-CMA-ES and it gives on slightly more functions lower average errors than IPOP-CMA-ES than vice versa. The average error values that correspond to a better result between UACOR-c and IPOP-CMA-ES are highlighted in Table 3.4.

As a final step, we compare UACOR-c with five recent state-of-the-art continuous optimization algorithms published since 2011. These reference algorithms include HDDE [Dorronsoro and Bouvry, 2011], Pro-JADE [Epitropakis et al., 2011], Pro-SaDE [Epitropakis et al., 2011], Pro-DEGL [Epitropakis et al., 2011] and ABC-MR [Akay and Karaboga, 2012]. In the original literature, these algorithms were tested on the CEC'05 benchmark set for which the parameter values of the algorithms were either set by experience or they were manually tuned. We directly obtain the data of the five algorithms on the CEC'05 benchmark set from the original papers. Table 3.5 shows that UACOR-c gives on the 30 and 50 dimensional problems on more functions lower average errors than each of these five

state-of-the-art algorithms. For each algorithm, Table 3.6 summarizes the average ranking, the number of times the optimum thresholds is reached and the number of lowest average error values obtained across all six algorithms that are compared. Although with the exception of ABC-MR the differences to these algorithms are not found to be statistically significant (neither are the differences among these algorithms statistically significant), it is a noteworthy result that UACOR-c obtains the best average ranking, the highest number of optimum thresholds and that it is the best performing algorithm for most functions.

Table 3.3: The average errors obtained by $ACO_{\mathbb{R}}$-s, $DACO_{\mathbb{R}}$-s, $IACO_{\mathbb{R}}$-Mtsls1-s, MOS-DE and UACOR-s for each SOCO function. The numbers in parenthesis at the bottom of the table represent the number of times an algorithm is better, equal or worse, respectively, than UACOR-s. Error values lower than 10^{-14} are approximated to 10^{-14}. The average errors that correspond to a better result between MOS-DE and UACOR-c are highlighted.

Dim	f_{soco}	$ACO_{\mathbb{R}}$-s	$DACO_{\mathbb{R}}$-s	$IACO_{\mathbb{R}}$-Mtsls1-s	MOS-DE	UACOR-s
100	f_{soco1}	5.32E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
	f_{soco2}	2.77E+01	3.82E+01	5.27E−12	2.94E−12	6.53E−12
	f_{soco3}	1.96E+02	2.86E+03	4.77E+02	2.03E+01	3.81E+02
	f_{soco4}	6.34E+02	3.89E+02	1.00E−14	1.00E−14	1.00E−14
	f_{soco5}	2.96E−04	4.96E−01	1.00E−14	1.00E−14	1.00E−14
	f_{soco6}	2.04E−08	3.49E+00	1.00E−14	1.00E−14	1.00E−14
	f_{soco7}	9.94E−10	5.01E−01	1.00E−14	1.00E−14	1.00E−14
	f_{soco8}	4.39E+04	2.28E+04	1.39E+00	9.17E−02	1.54E+00
	f_{soco9}	4.78E+00	1.84E+02	1.62E−01	1.00E−14	1.00E−14
	f_{soco10}	2.75E+00	2.09E+01	1.00E−14	1.00E−14	1.00E−14
	f_{soco11}	3.96E+00	1.90E+02	1.67E−01	1.00E−14	1.00E−14
	f_{soco12}	1.13E+01	2.39E+02	4.01E−02	1.00E−14	1.00E−14
	f_{soco13}	1.47E+02	3.92E+02	4.19E+01	1.75E+01	9.31E+01
	f_{soco14}	3.40E+02	2.40E+02	9.23E+00	1.68E−11	1.00E−14
	f_{soco15}	5.89E−01	4.41E+00	1.00E−14	1.00E−14	1.00E−14
	f_{soco16}	1.61E+00	4.18E+02	4.24E−01	1.00E−14	1.00E−14
	f_{soco17}	6.27E+01	6.65E+02	8.40E+01	1.43E+01	5.30E+01
	f_{soco18}	1.57E+01	1.17E+02	1.07E−01	1.00E−14	1.00E−14
	f_{soco19}	1.48E+00	1.61E+01	1.00E−14	1.00E−14	1.00E−14
By f_{soco}		(1, 0, 18)†	(0, 1, 18)†	(3, 8, 8)	(5, 13, 1)	

† denotes a significant difference between the corresponding algorithm and UACOR-s by a Friedman test at the 0,05 α-level over the distribution of average errors of $ACO_{\mathbb{R}}$-s, $DACO_{\mathbb{R}}$-s, $IACO_{\mathbb{R}}$-Mtsls1-s and UACOR-s.

Table 3.4: The average errors obtained by $ACO_{\mathbb{R}}$-c, $DACO_{\mathbb{R}}$-c, $IACO_{\mathbb{R}}$-Mtsls1-c, IPOP-CMA-ES and UACOR-c for each CEC'05 function. The numbers in parenthesis at the bottom of the table represent the number of times an algorithm is better, equal or worse, respectively, compared to UACOR-c. Error values lower than 10^{-8} are approximated to 10^{-8}. The average errors that correspond to a better result between IPOP-CMA-ES and UACOR-c are highlighted.

Dim	f_{cec}	$ACO_{\mathbb{R}}$-c	$DACO_{\mathbb{R}}$-c	$IACO_{\mathbb{R}}$-Mtsls1-c	IPOP-CMA-ES	UACOR-c
30	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.00E−08	4.74E+00	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	3.88E+05	4.21E+06	2.19E+05	1.00E−08	2.92E+05
	f_{cec4}	2.75E−04	2.62E+02	9.45E+03	1.11E+04	4.11E+03
	f_{cec5}	1.00E−08	1.00E−08	6.79E−08	1.00E−08	1.00E−08
	f_{cec6}	8.99E+00	2.50E+01	1.64E+02	1.00E−08	2.92E+01
	f_{cec7}	1.80E−02	1.54E−02	1.00E−02	1.00E−08	8.96E−03
	f_{cec8}	2.00E+01	2.02E+01	2.00E+01	2.01E+01	2.00E+01
	f_{cec9}	2.50E+01	6.35E+01	1.00E−08	9.38E−01	1.00E−08
	f_{cec10}	4.51E+01	6.58E+01	1.19E+02	1.65E+00	1.06E+02
	f_{cec11}	3.75E+01	3.19E+01	2.36E+01	5.48E+00	2.15E+01
	f_{cec12}	3.59E+03	1.92E+04	7.89E+03	4.43E+04	1.16E+04
	f_{cec13}	3.67E+00	4.57E+00	1.28E+00	2.49E+00	1.46E+00
	f_{cec14}	1.25E+01	1.34E+01	1.32E+01	1.29E+01	1.29E+01
	f_{cec15}	3.40E+02	3.28E+02	2.48E+02	2.08E+02	2.26E+02
	f_{cec16}	1.33E+02	1.93E+02	2.49E+02	3.50E+01	2.29E+02
	f_{cec17}	1.49E+02	1.81E+02	3.35E+02	2.91E+02	2.62E+02
	f_{cec18}	9.12E+02	9.07E+02	9.02E+02	9.04E+02	8.77E+02
	f_{cec19}	9.11E+02	9.07E+02	8.92E+02	9.04E+02	8.82E+02
	f_{cec20}	9.12E+02	9.07E+02	8.97E+02	9.04E+02	8.78E+02
	f_{cec21}	5.38E+02	5.00E+02	5.12E+02	5.00E+02	5.00E+02
	f_{cec22}	9.08E+02	8.70E+02	9.90E+02	8.03E+02	9.80E+02
	f_{cec23}	5.75E+02	5.35E+02	5.66E+02	5.34E+02	5.34E+02
	f_{cec24}	2.27E+02	7.85E+02	1.26E+03	9.10E+02	8.30E+02
	f_{cec25}	2.19E+02	2.31E+02	5.60E+02	2.11E+02	4.74E+02
50	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.80E−04	2.61E+03	4.08E−07	1.00E−08	1.00E−08
	f_{cec3}	6.68E+05	6.72E+06	5.60E+05	1.00E−08	6.06E+05
	f_{cec4}	8.36E+03	4.69E+04	5.33E+04	4.68E+05	4.15E+04
	f_{cec5}	2.23E−05	2.02E−01	7.95E−07	2.85E+00	1.00E−08
	f_{cec6}	2.92E+01	5.18E+01	1.73E+02	1.00E−08	5.00E+01
	f_{cec7}	9.93E−03	1.08E−02	4.53E−03	1.00E−08	7.68E−03
	f_{cec8}	2.00E+01	2.02E+01	2.00E+01	2.01E+01	2.00E+01
	f_{cec9}	4.82E+01	1.15E+02	1.00E−08	1.39E+00	1.00E−08
	f_{cec10}	9.77E+01	1.42E+02	2.83E+02	1.72E+00	2.63E+02
	f_{cec11}	7.30E+01	5.81E+01	4.63E+01	1.17E+01	4.57E+01
	f_{cec12}	2.74E+04	1.07E+05	1.47E+04	2.27E+05	5.26E+04
	f_{cec13}	7.24E+00	1.06E+01	2.13E+00	4.59E+00	2.38E+00
	f_{cec14}	2.24E+01	2.29E+01	2.26E+01	2.29E+01	2.24E+01
	f_{cec15}	3.26E+02	3.61E+02	2.64E+02	2.04E+02	3.00E+02
	f_{cec16}	1.10E+02	1.64E+02	2.89E+02	3.09E+01	2.74E+02
	f_{cec17}	1.62E+02	2.45E+02	5.65E+02	2.34E+02	4.64E+02
	f_{cec18}	9.33E+02	9.26E+02	9.31E+02	9.13E+02	8.83E+02
	f_{cec19}	9.34E+02	9.27E+02	9.18E+02	9.12E+02	8.83E+02
	f_{cec20}	9.36E+02	9.27E+02	9.19E+02	9.12E+02	8.95E+02
	f_{cec21}	5.39E+02	9.82E+02	5.12E+02	1.00E+03	5.00E+02
	f_{cec22}	9.48E+02	9.07E+02	1.06E+03	8.05E+02	1.06E+03
	f_{cec23}	5.56E+02	1.02E+03	5.53E+02	1.01E+03	5.39E+02
	f_{cec24}	2.94E+02	9.06E+02	1.40E+03	9.55E+02	1.30E+03
	f_{cec25}	2.63E+02	3.39E+02	9.52E+02	2.15E+02	7.59E+02
By f_{cec}		(19, 7, 24)	(14, 4, 32)†	(8, 8, 34)†	**(20, 8, 22)**	

† denotes a significant difference between the corresponding algorithm and UACOR-s by a Friedman test at the 0,05 α-level over the distribution of average errors of $ACO_{\mathbb{R}}$-c, $DACO_{\mathbb{R}}$-c, $IACO_{\mathbb{R}}$-Mtsls1-c and UACOR-c.

Table 3.5: The average errors obtained by HDDE, Pro-JADE, Pro-SaDE, Pro-DEGL, ABC-MR and UACOR-c for for each CEC'05 function. The numbers in parenthesis at the bottom of the table represent the number of times an algorithm is better, equal or worse, respectively, compared to UACOR-c. Error values lower than 10^{-8} are approximated to 10^{-8}. The lowest average errors values are highlighted.

Dim	f_{cec}	HDDE	Pro-JADE	Pro-SaDE	Pro-DEGL	ABC-MR	UACOR-c
30	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	8.13E+00	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	2.31E+06	1.85E+04	2.28E+06	4.20E+04	2.20E+05	2.92E+05
	f_{cec4}	1.33E+02	1.00E−08	2.00E−05	1.00E−08	1.00E−08	4.11E+03
	f_{cec5}	7.66E+02	5.90E+01	5.51E+01	1.91E+01	6.02E+03	1.00E−08
	f_{cec6}	3.19E+01	1.89E+01	1.36E+00	1.20E+00	1.38E+02	2.92E+01
	f_{cec7}	4.70E+03	4.70E+03	4.70E+03	4.69E+03	1.49E−02	8.96E−03
	f_{cec8}	2.09E+01	2.09E+01	2.10E+01	2.09E+01	2.09E+01	2.00E+01
	f_{cec9}	3.91E+00	1.00E−08	1.00E−08	3.58E+01	6.60E+01	1.00E−08
	f_{cec10}	6.01E+01	8.18E+01	1.01E+02	5.18E+01	2.01E+02	1.06E+02
	f_{cec11}	2.57E+01	3.01E+01	3.37E+01	2.01E+01	3.56E+01	2.15E+01
	f_{cec12}	7.86E+03	2.63E+04	1.48E+03	2.35E+04	9.55E+04	1.16E+04
	f_{cec13}	2.04E+00	3.32E+00	2.95E+00	3.40E+00	1.07E+01	1.46E+00
	f_{cec14}	1.27E+01	1.29E+01	1.31E+01	1.24E+01	1.88E−01	1.29E+01
	f_{cec15}	3.17E+02	3.71E+02	3.86E+02	3.53E+02	2.88E+02	2.26E+02
	f_{cec16}	8.58E+01	1.14E+02	6.97E+01	1.76E+02	3.06E+02	2.29E+02
	f_{cec17}	1.01E+02	1.45E+02	7.20E+01	1.60E+02	3.01E+02	2.62E+02
	f_{cec18}	9.03E+02	8.60E+02	8.56E+02	9.09E+02	8.12E+02	8.77E+02
	f_{cec19}	9.04E+02	8.90E+02	8.67E+02	9.10E+02	8.17E+02	8.82E+02
	f_{cec20}	9.04E+02	8.96E+02	8.52E+02	9.10E+02	8.23E+02	8.78E+02
	f_{cec21}	5.00E+02	5.06E+02	5.00E+02	6.79E+02	6.42E+02	5.00E+02
	f_{cec22}	8.76E+02	8.95E+02	9.09E+02	8.94E+02	9.04E+02	9.80E+02
	f_{cec23}	5.34E+02	5.00E+02	5.00E+02	6.77E+02	8.20E+02	5.34E+02
	f_{cec24}	2.00E+02	2.00E+02	2.00E+02	7.77E+02	2.01E+02	8.30E+02
	f_{cec25}	1.28E+03	1.67E+03	1.63E+03	1.64E+03	2.00E+02	4.74E+02
50	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	3.30E+02	1.00E−08	7.40E−04	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	4.44E+06	4.30E+04	7.82E+05	1.93E+05	1.13E+06	6.06E+05
	f_{cec4}	2.94E+03	3.19E−01	6.64E+01	1.08E−01	3.82E+02	4.15E+04
	f_{cec5}	3.22E+03	1.83E+03	1.95E+03	2.23E+03	1.03E+04	1.00E−08
	f_{cec6}	5.74E+01	1.04E+01	1.15E+01	8.77E−01	2.47E+03	5.00E+01
	f_{cec7}	6.20E+03	6.20E+03	6.20E+03	6.20E+03	8.10E−01	7.68E−03
	f_{cec8}	2.11E+01	2.10E+01	2.11E+01	2.11E+01	2.11E+01	2.00E+01
	f_{cec9}	1.87E+01	2.77E+01	6.61E−01	7.94E+01	2.59E+02	1.00E−08
	f_{cec10}	1.13E+02	1.99E+02	6.23E+01	9.24E+01	4.58E+02	2.63E+02
	f_{cec11}	5.19E+01	6.03E+01	6.61E+01	6.14E+01	7.03E+01	4.57E+01
	f_{cec12}	3.84E+04	9.45E+04	7.34E+03	6.32E+04	8.88E+05	5.26E+04
	f_{cec13}	4.36E+00	9.14E+00	6.90E+00	5.41E+00	3.50E+01	2.38E+00
	f_{cec14}	2.23E+01	2.26E+01	2.28E+01	2.26E+01	2.32E+01	2.24E+01
	f_{cec15}	2.62E+02	3.80E+02	3.96E+02	3.44E+02	2.52E+02	3.00E+02
	f_{cec16}	1.05E+02	1.44E+02	4.85E+01	1.63E+02	3.39E+02	2.74E+02
	f_{cec17}	1.15E+02	1.92E+02	9.36E+01	1.94E+02	3.08E+02	4.64E+02
	f_{cec18}	9.17E+02	9.26E+02	9.05E+02	9.28E+02	9.85E+02	8.83E+02
	f_{cec19}	9.16E+02	9.32E+02	8.97E+02	9.28E+02	9.70E+02	8.83E+02
	f_{cec20}	9.16E+02	9.33E+02	9.11E+02	9.29E+02	9.70E+02	8.95E+02
	f_{cec21}	7.37E+02	5.00E+02	5.00E+02	9.50E+02	8.34E+02	5.00E+02
	f_{cec22}	9.01E+02	9.49E+02	9.60E+02	9.28E+02	8.75E+02	1.06E+03
	f_{cec23}	7.85E+02	5.00E+02	5.06E+02	9.35E+02	5.71E+02	5.39E+02
	f_{cec24}	2.00E+02	2.00E+02	2.00E+02	6.81E+02	2.01E+02	1.30E+03
	f_{cec25}	1.37E+03	1.71E+03	1.69E+03	1.67E+03	2.01E+02	7.59E+02
By f_{cec}		(17, 4, 29)	(19, 7, 24)	(21, 6, 23)	(18, 4, 28)	(15, 4, 31)†	

† denotes a significant difference between the corresponding algorithm and UACOR-c by a Friedman test at the 0,05 α-level over the distribution of average errors of HDDE, Pro-JADE, Pro-SaDE, Pro-DEGL, ABC-MR and UACOR-c.

Table 3.6: Given are the average rank, the number of optimum thresholds reached, and the number of times the lowest average errors reached by each algorithm presented in Table 3.5. In addition, we give the publication source for each reference algorithm.

Algorithms	Average Ranking	Num of Optima	Num of lowest average error values	Publication Sources
UACOR-c	**3.05**	**8**	**21**	
Pro-SaDE	3.19	4	16	IEEE TEC, 2011
Pro-JADE	3.40	6	13	IEEE TEC, 2011
HDDE	3.44	2	7	IEEE TEC, 2011
Pro-DEGL	3.69	5	10	IEEE TEC, 2011
ABC-MR	4.23	5	13	Information Sciences, 2012

3.5 UACOR$^+$: Re-designed UACOR

Recently, a number of high-performing algorithms have been proposed that integrate CMA-ES as a local search algorithm into another metaheuristic. Examples of such approaches are a recent memetic algorithm [Molina et al., 2010a], the integration of CMA-ES into a PSO algorithm [Müller et al., 2009], or a DE algorithm [Ghosh et al., 2012]. Such an approach is particularly relevant in case functions are rotated because CMA-ES is invariant against linear transformations of the search space. Clearly, this may also help the UACOR framework to improve results on rotated functions, thus, mainly on the CEC benchmark set.

In this section, we re-design the UACOR algorithm by implementing CMA-ES as a alternative local search procedure. We called this updated version UACOR$^+$. The initial population size of the CMA-ES is set to a random size between $\lambda = 4 + \lfloor 3\ln(D) \rfloor$ and $2^3 \times \lambda = 4 + \lfloor 3\ln(D) \rfloor$. The initial step size is kept the same adaptive way as other local search methods. There are no fixed number of iterations that are given to CMA-ES local search procedure. The CMA-ES local search procedure run until one of the three stopping criteria [Auger and Hansen, 2005] is triggered. The three stopping criteria use three parameters *stopTolFunHist*($= 10^{-20}$), *stopTolFun*($= 10^{-12}$) and *stopTolX*($= 10^{-12}$); they refer to the improvement of the best objective function value in the last $10 + \lceil 30D/\lambda \rceil$ generations, the function values of the recent generation, and the standard deviation of the normal distribution in all coordinates, respectively.

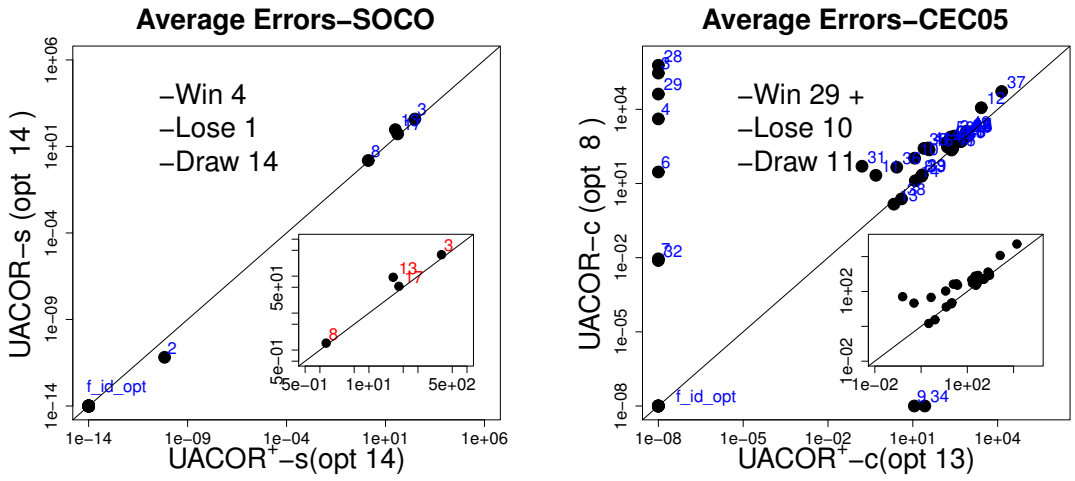
We repeat the automatic algorithm configuration for UACOR$^+$ as in Section 3.3. The tuned parameter settings of UACOR$^+$'s instantiations, UACOR$^+$-s and UACOR$^+$-c are given in the appendix in Table A.1. The flowcharts of UACOR$^+$-s and UACOR$^+$-c are given in the appendix in Fig. A.1 and A.2, page 133 and 134. A sensible change is that UACOR$^+$-s still uses Mtsls1 as local search procedure and UACOR$^+$-c uses CMA-ES as local search procedure. We repeat the same experimental study for UACOR$^+$ as those for UACOR. The experimental results of UACOR$^+$ are shown in Figures A.3 and A.4, Tables A.2, A.3, A.4 and A.5. We summarize the main changes of the results of UACOR$^+$ with respect to those of UACOR as follows.

1. In the SOCO benchmark functions, the results of UACOR$^+$-s becomes significantly better than those of ACO$_{\mathbb{R}}$-s, DACO$_{\mathbb{R}}$-s and IACO$_{\mathbb{R}}$-Mtsls1-s, while the results of UACOR-s are only significantly better than those of ACO$_{\mathbb{R}}$-s and DACO$_{\mathbb{R}}$-s ; UACOR$^+$-s performs significantly better than 12 other algorithms in SOCO, two more than UACOR-s does. The performance dif-

ference between UACOR-s and UACOR$^+$-s is mainly caused by their slightly different parameter values on the same UACOR instantiation, which were obtained from a stochastic tuning procedure.

2. In the CEC'05 benchmark functions, the results of UACOR$^+$-c are significantly better than those of ACO$_\mathbb{R}$-c, DACO$_\mathbb{R}$-c and IACO$_\mathbb{R}$-Mtsls1-c, while the results of UACOR-c are only significantly better than those of DACO$_\mathbb{R}$-c and IACO$_\mathbb{R}$-Mtsls1-c; UACOR$^+$-c also more often obtains the lower average error values than IPOP-CMA-ES than UACOR-c; UACOR$^+$-c obtains significantly better resuls than those obtained by HDDE , Pro-JADE, Pro-SaDE , Pro-DEGL and ABC-MR, while UACOR-c only matches their results when aggregated across the whole benchmark set.

Fig. 3.6 shows correlation plots that illustrate the relative performance for UACOR-s and UACOR$^+$-s on the SOCO benchmark set and for UACOR-c and UACOR$^+$-c on the CEC'05 benchmark set. There is no statistically significant difference between the results of UACOR-s and UACOR$^+$-s. However, UACOR$^+$-c reaches statistically significantly better performance than UACOR-c. This comparison confirms our expectation of using CMA-ES as a alternative local search procedure to improve results on rotated functions.



(a) (b)

Figure 3.6: Correlation plots of UACOR-s and UACOR$^+$-s on 19 SOCO function of dimension 100, UACOR-c and UACOR$^+$-c and 25 CEC'05 benchmark functions over dimensions 30 and 50 (the indexes of 50 dimensional functions are labeled from 26 to 50). Each point represents the average error value obtained by either of the two algorithms. A point on the upper triangle delimited by the diagonal indicates better performance for the algorithm on the x-axis; a point on the lower right triangle indicates better performance for the algorithm on the y-axis. The number labeled beside some outstanding points represent the index of the corresponding function. The comparison is conducted based on average error values and the comparison results of the algorithm on the x-axis are presented in the form of -win, -draw, -lose, respectively. We marked with a + symbol those cases in which there is a statistically significant difference at the 0.05 α-level between the algorithms checked by a two-sided Wilcoxon matched-pairs signed-ranks test between UACOR-s and UACOR$^+$-s. The number of opt on the axes shows the number of means that is lower than the zero threshold, obtained by the corresponding algorithm.

3.6 Summary

In this chapter, we proposed UACOR, a unified ant colony framework, that integrates algorithmic components from three previous ACO algorithms for continuous optimization problems, $ACO_{\mathbb{R}}$ [Socha and Dorigo, 2008], $DACO_{\mathbb{R}}$ [Leguizamón and Coello, 2010] and $IACO_{\mathbb{R}}$-LS [Liao et al., 2011b]. The UACOR framework is flexible and allows the instantiation of new ACO algorithms for continuous optimization through the exploitation of automatic algorithm configuration techniques. In fact, in this way we can obtain from the available algorithmic components new ACO algorithms that have not been considered or tested before.

In the experimental part of the chapter, we have shown that the combination of a flexible, unified algorithm for continuous optimization and automatic algorithm configuration tools can be instrumental for generating new, very high performing algorithms for continuous optimization. We have configured UACOR using Iterated F-Race [Birattari et al., 2010], an automatic algorithm configuration technique implemented in the irace package [López-Ibáñez et al., 2011], on small dimensional training functions taken from two well-known benchmark sets, the SOCO and the CEC'05 benchmark sets, for continuous optimization. The computational results showed that the tuned UACOR algorithms obtain better performance on each of the benchmark sets than the tuned variants of the three ACO algorithms that underly the UACOR framework, namely $ACO_{\mathbb{R}}$, $DACO_{\mathbb{R}}$ and $IACO_{\mathbb{R}}$-LS. Moreover, when UACOR is automatically configured for the SOCO benchmark set, it performs better or competitive to all the recent 16 algorithms benchmarked on this benchmark set; when configured for the CEC'05 benchmark set, it performs competitive to IPOP-CMA-ES, the acknowledged state-of-the-art algorithm on this benchmark set and also competitive or superior to other five recent high-performance continuous optimizers that were evaluated on this benchmark set.

Finally we proposed $UACOR^{+}$, a re-designed and improved UACOR that now also includes the option of using CMA-ES as a local search. The computational results showed that the tuned $UACOR^{+}$ algorithms statistically significantly improve $ACO_{\mathbb{R}}$, $DACO_{\mathbb{R}}$ and $IACO_{\mathbb{R}}$-LS on each of the benchmark sets. When $UACOR^{+}$ is automatically configured for the SOCO benchmark set, it performs statistically significantly better than 12 of the recent 16 algorithms benchmarked on this benchmark set. When configured for the CEC'05 benchmark set, it performs superior to IPOP-CMA-ES and statistically significantly better than other five recent high-performance continuous optimizers that were evaluated on this

benchmark set.

In summary, in this chapter we have proven the high potential ACO algorithms have for continuous optimization and that automatic algorithm configuration has a high potential also for the development of continuous optimizers out of algorithm components. This should encourage also other researchers to apply such automatic algorithm configuration techniques in the design of continuous optimizers.

Chapter 4

iCMAES-ILS: A cooperative competitive hybrid algorithm for continuous optimization

The development of algorithms for tackling difficult, high-dimensional continuous optimization problems (e.g. high-dimensional CEC'05 benchmark functions [Suganthan et al., 2005]) is a main research theme in population based heuristic algorithms. While initially pure algorithm strategies have been tested extensively [Hansen and Ostermeier, 1996, 2001, Karaboga and Basturk, 2007, Kennedy and Eberhart, 1995, Socha and Dorigo, 2008, Storn and Price, 1997], in the past few years most newly proposed algorithms that reach state-of-the-art performance combine elements from different algorithmic techniques, that is, they are hybrid algorithms. A popular class of hybrid approaches embeds local optimizers into global optimizers. Several examples that follow this approach use the covariance matrix adaptation evolution strategy (CMA-ES) [Hansen and Ostermeier, 1996, 2001, Hansen et al., 2003] as a local optimizer and embed it into a real-coded steady state genetic algorithm (SSGA) [Molina et al., 2010a], particle swarm optimization (PSO) [Müller et al., 2009], differential evolution (DE) [Ghosh et al., 2012] or into the UACOR framework as done in the previous chapter, Section 3.5. Alternative approaches include the definition of algorithm portfolios [Gomes and Selman, 2001, Huberman et al., 1997] or the exploitation of algorithm selection techniques [Bischl et al., 2012, Rice, 1976]. Peng et al. [Peng et al., 2010] define a portfolio of four algorithms and do report, on few classical benchmark functions, some improvements over IPOP-CMA-ES [Auger and Hansen, 2005]. More recently, algorithm selection is being explored as one possible way to improve upon state-of-the-art algorithms. One common way is to define problem features and based on the measured feature values to choose an algorithm for final execution. This approach has been followed by [Bischl et al., 2012], where features for continuous optimization problems taken from [Mersmann et al., 2011] have been exploited.

However, they conclude that the feature computation gives a significant overhead leading to a situation that it may make this approach not advantageous.

In this chapter, we propose iCMAES-ILS, a structurally simple, hybrid algorithm that loosely couples IPOP-CMA-ES with an iterated local search (ILS) algorithm. The hybrid iCMAES-ILS algorithm consists of an initial competition phase, in which IPOP-CMA-ES and the ILS algorithm compete for further execution in the deployment phase, where only one of the two algorithms is run until the budget is exhausted. The initial competition phase features also a cooperative aspect between IPOP-CMA-ES and the ILS algorithm. In fact, first IPOP-CMA-ES is run for a specific initial budget and the best solution found by IPOP-CMA-ES is used to bias the acceptance decision in the ILS algorithm, which is run next for a same initial budget as IPOP-CMA-ES.

We consider iCMAES-ILS to be a simple algorithm since the two algorithms it is based upon can be seen as black boxes and the main design issue is on how to interleave the execution of the two component algorithms. iCMAES-ILS is also related to approaches for algorithm selection [Rice, 1976]. However, instead of extracting sometimes complex problem features and then deciding upon which algorithm to execute [Bischl et al., 2012], we use the performance of the two component algorithms in an initial competition phase as the feature and then directly employ the better one, thus, making again the design of iCMAES-ILS very simple.

We also show that iCMAES-ILS is an effective algorithm that reaches (and actually statistically significantly surpasses) state-of-the-art performance. As our testbed, we use all 25 benchmark functions of 30 and 50 dimensions from the CEC'05 benchmark [Suganthan et al., 2005]. We use this benchmark set to compare iCMAES-ILS to its component algorithms, IPOP-CMA-ES and ILS, and also to a number of alternative hybrids between IPOP-CMA-ES and ILS based on using portfolios, interleaving the execution of IPOP-CMA-ES and ILS, and using ILS as an improvement method between restarts of IPOP-CMA-ES. These comparisons indicate that iCMAES-ILS reaches statistically significantly better performance than almost all competitors and, thus, establishes our hybrid design as the most performing one. Additional comparisons to two state-of-the-art CMA-ES hybrid algorithms (MA-LSCh-CMA [Molina et al., 2010a] and PS-CMA-ES [Müller et al., 2009]) further show statistically significantly better performance of iCMAES-ILS over these competitors. In a final step, we tune iCMAES-ILS by applying `irace` [López-Ibáñez et al., 2011] to examine the further performance improvements that can be expected from a more careful fine-tuning of the algorithm parameters.

Overall, our experimental results establish iCMAES-ILS as a new state-of-the-art algorithm for continuous optimization.

The chapter is organized as follows. Section 4.1 describes iCMAES-ILS. In Section 4.2, we analyze, evaluate and compare iCMAES-ILS to results from the literature. In Section 4.3, we further fine-tune iCMAES-ILS and examine the further performance improvements. We end by some discussion and concluding remarks in Sections 4.4 and 4.5.

4.1 iCMAES-ILS algorithm

The iCMAES-ILS algorithm build upon and extends IPOP-CMA-ES for continuous optimization. The main details of the IPOP-CMA-ES algorithm have already been described in Section 2.3.2 and we refer the reader to the details described there. Next, we describe the ILS algorithm and then describe the hybrid iCMAES-ILS algorithm.

4.1.1 ILS

Iterated local search [Lourenço et al., 2010] is a stochastic local search method [Hoos and Stützle, 2005] that iterates between improvements through local search and solution perturbations that are used to define new promising starting solutions for a local search. An additional acceptance criterion decides from which candidate solution the search is continued. Recently, iterated local search has been applied to solve continuous optimization problems [Gimmler, 2005, Kramer, 2010] showing very promising results. For this chapter, we present a new iterated local search algorithm (labeled as ILS), where a high-performing derivative-free local search algorithm for continuous optimization, Mtsls1 [Tseng and Chen, 2008], is used. The Mtsls1 local search was chosen since it is a crucial component of several high-performing algorithms for continuous optimization [LaTorre et al., 2011, Liao et al., 2011b]. Mtsls1 iteratively searches along dimensions one by one in a certain step size. If one Mtsls1 iteration does not find an improvement in any of the dimensions, the next Mtsls1 iteration halves the search step size. For more details, we refer back to Chapter 2, Section 2.1.1, page 17. Algorithm 3 gives an outline of the proposed ILS algorithm. For purposes that are explained later, we use two input solutions, S and S_{best}; the behavior of a standard ILS algorithm [Lourenço et al., 2010] would be obtained by setting $S = S_{\text{best}}$. In our ILS algorithm, local search is first executed generating a new solution S_{new}. If S_{new} is better than the best-so-far

Algorithm 3 Outline of the ILS algorithm

Input: Candidate solution S, S_{best} and termination criterion
Output: the best found solution
  while termination criterion is not satisfied **do**
    $S_{\text{new}} \leftarrow$ LS (S, ss, *LSIterations*) /* Local search procedure */
    if $f(S_{\text{new}}) < f(S_{\text{best}})$ **then**
      $S \leftarrow S_{\text{new}}$
      $S_{\text{best}} \leftarrow S_{\text{new}}$
    else
      $S \leftarrow S_{\text{rand}} + r \times (S_{\text{best}} - S_{\text{rand}})$ /* Perturbation */
    end if
  end while

solution S_{best}, S_{new} undergoes a refinement local search (that is, no perturbation is applied before calling the next time the local search procedure). Otherwise, the local search continues from a solution obtained by perturbation. The perturbation uses information from S_{best}. It is biased towards the best-so-far solution S_{best} using $S = s_{\text{rand}} + r \times (S_{\text{best}} - S_{\text{rand}})$, where S_{rand} is a random solution and r is a random number chosen uniformly at random in $[0, 1)$.

4.1.2 iCMAES-ILS

iCMAES-ILS is a hybrid of IPOP-CMA-ES and ILS. iCMAES-ILS consists of two phases, competition and deployment. In the competition phase, IPOP-CMA-ES and ILS start from the same initial solution and are sequentially executed with the same budget of function evaluations, *CompBudget*. After the competition phase, the one of the two that found the better solution is deployed for the remaining budget. Specifically, if ILS finds a better solution than IPOP-CMA-ES, iCMAES-ILS applies ILS for the remaining budget. Otherwise, iCMAES-ILS restarts IPOP-CMA-ES from the best-so-far solution with its initial default parameter settings. The restart of IPOP-CMA-ES is triggered by the end of the competition phase, that is, not due to the internal termination criterion of IPOP-CMA-ES; as we will show in Section 4.2, this external restart trigger is beneficial to iCMAES-ILS' performance.

In iCMAES-ILS, the input S_{best} of ILS is set to the best-so-far solution returned by the IPOP-CMA-ES. Thus, the competition phase in iCMAES-ILS includes also some aspect of cooperation between the two algorithms implemented through the transfer of a candidate solution from IPOP-CMA-ES to ILS. The transferred candidate solution influences the search behavior of ILS by biasing the acceptance decision and the perturbation. In an independent ILS algorithm, which we will use as a reference for comparison, the input S_{best} is set to the same as the input

Algorithm 4 Outline of iCMAES-ILS

Input: Candidate solution S and termination criterion ($TotalBudget$)
Output: the best found solution

```
CompBudget ← comp_r × TotalBudget
S_best ← IPOP-CMA-ES (S, CompBudget)
S'_best ← ILS (S, S_best, CompBudget)
/* Heuristic deployment */
if f(S'_best < f(S_best) then
   ILS (S'_best, S'_best, TotalBudget−2×CompBudget) /* ILS found better solution than IPOP-CMA-ES */
else
   IPOP-CMA-ES (S_best, TotalBudget−2×CompBudget) /* IPOP-CMA-ES found better solution than ILS */
end if
```

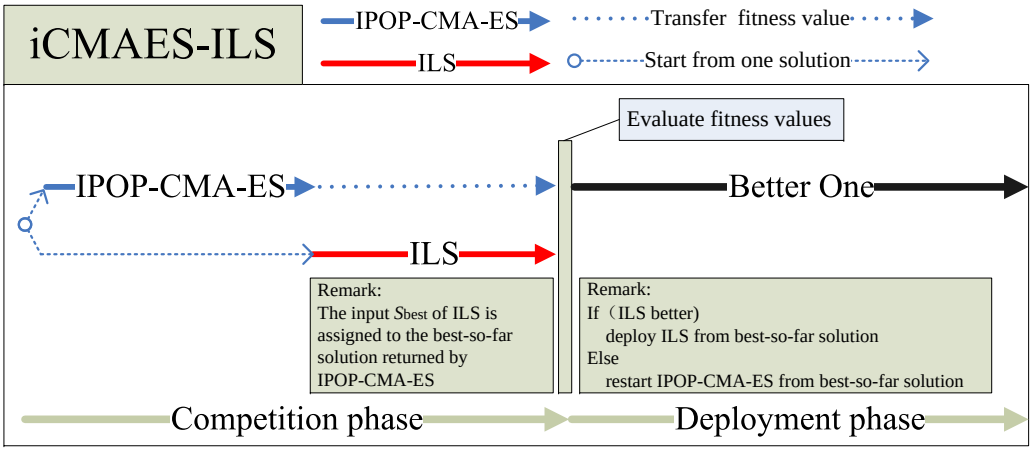


Figure 4.1: A schematic view of iCMAES-ILS in the perspective of optimization process.

candidate solution S. We set $CompBudget = comp_r \times TotalBudget$, where $TotalBudget$ is the maximum number of function evaluations in each algorithm run. As default, we use $comp_r = 0.1$. An outline and a schematic view of iCMAES-ILS is given in Algorithm 4 and Fig. 4.1, respectively.

4.2 Algorithm analysis and evaluation

Experimental setup

Our test-suite consists of 25 CEC'05 benchmark functions (functions labeled as f_{cec*}) of dimensions 30 and 50. We followed the protocol described in [Suganthan et al., 2005] for the CEC'05 test-suite, that is, the maximum number of function evaluations is $10\,000 \times D$. The investigated algorithms were independently run 25

Table 4.1: Given are the default parameter settings of iCMAES-ILS. $(B - A)$ is the size of the initial search interval. $TotalBudget$ is the maximum number of function evaluations.

Algorithm Components	**Parameter settings**		
IPOP-CMA-ES	Init pop size: λ_0	=	$4 + \lfloor 3\ln(D) \rfloor$
	Parent size: μ	=	$\lfloor \lambda/2 \rfloor$
	Init step size: σ_0	=	$0.5 \times (B - A)$
	IPOP factor: *ipop*	=	2
	stopTolFun	=	10^{-12}
	stopTolFunHist	=	10^{-20}
	stopTolX	=	10^{-12}
ILS	*LSIterations*	=	$1.5 \times D$
Competition Design	$comp_r$	=	0.1

times on each function. We report error values defined as $f(S) - f(S^*)$, where S is a candidate solution and S^* is the optimal solution. Error values lower than 10^{-8} are clamped to 10^{-8}, which is used as the zero threshold. To ensure that the final solution obtained by iCMAES-ILS is inside the bounds, the bound constraints of the benchmark functions are enforced by clamping the variable of each generated solution that violates the bound constraints to the nearest value on the bounds before evaluating a solution. Our analysis considers the average errors on each function, and the distribution of the average errors on the CEC'05 benchmark function set over dimensions 30 and 50. We apply the Friedman test at the 0.05 α-level to check whether the differences in the average rank values obtained by multiple algorithms over the CEC'05 benchmark functions is statistically significant. The parameter settings used for iCMAES-ILS are listed in Table 4.1. The parameter settings of IPOP-CMA-ES and ILS in iCMAES-ILS are the same as the default IPOP-CMA-ES and ILS algorithms, respectively.

4.2.1 Algorithm analysis: the role of ILS

As a first indication of performance, we present in Table 4.2 the average errors by ILS, IPOP-CMA-ES, and iCMAES-ILS for 25 CEC'05 function of dimension 50. Interestingly, iCMAES-ILS obtains lower average errors than ILS and IPOP-CMA-ES on most of the functions: iCMAES-ILS finds 19 lowest average error values while ILS and IPOP-CMA-ES find 7 and 10, respectively. These results indicate that iCMAES-ILS finds better results than its component algorithms, that is, it shows some synergetic effect through the hybrid design.

Next, we analyze the role of ILS in iCMAES-ILS on the 18 benchmark functions

of dimension 50, where iCMAES-ILS obtained either a better or a worse average error than IPOP-CMA-ES alone. On a per function basis, Fig. 4.2 shows the development of the average error for IPOP-CMA-ES and iCMAES-ILS over the number of function evaluations (also called SQT curves [Hoos and Stützle, 2005] for solution quality over time). The ILS phase in iCMAES-ILS is shown between the two vertically dotted lines in the SQT curves for iCMAES-ILS. It is observed that in functions f_{cec8}, f_{cec12}, f_{cec13}, f_{cec14}, f_{cec15}, f_{cec21}, f_{cec23}, f_{cec25}, where the average error after the ILS execution is lower than after executing IPOP-CMA-ES, iCMAES-ILS strongly improves upon IPOP-CMA-ES w.r.t. final average solution quality. Only for functions f_{cec6}, f_{cec9}, the improvement by ILS does not lead to a final improvement of iCMAES-ILS over IPOP-CMA-ES. For functions f_{cec4}, f_{cec11}, f_{cec16}, f_{cec17}, f_{cec18}, f_{cec22}, ILS does not find any further improvement over IPOP-CMA-ES. However, the external restart triggered through the end of the ILS phase seems to be helpful, as indicated by the very good performance of iCMAES-ILS w.r.t. IPOP-CMA-ES on these functions. Specially in functions f_{cec4} and f_{cec17}, we can observe that the external restart trigger obviously helps. In fact, on these two functions IPOP-CMA-ES would not restart according to its internal restart criterion and, thus, stagnate on high average error values. Only for functions f_{cec5}, f_{cec10}, the restart trigger does not help. Table 4.3 summarizes our observations.

Table 4.2: The average errors obtained by ILS, IPOP-CMA-ES, and iCMAES-ILS for each CEC'05 benchmark function of dimension 50. Error values lower than 10^{-8} are approximated to 10^{-8}. The lowest average error values are highlighted.

f_{cec}	ILS	IPOP-CMA-ES	iCMAES-ILS
f_{cec1}	1.00E−08	1.00E−08	1.00E−08
f_{cec2}	1.00E−08	1.00E−08	1.00E−08
f_{cec3}	2.56E+05	1.00E−08	1.00E−08
f_{cec4}	9.28E+04	1.72E+04	1.11E+04
f_{cec5}	1.48E+04	6.25E−02	6.40E−01
f_{cec6}	8.19E+01	1.00E−08	1.04E+01
f_{cec7}	4.63E−03	1.00E−08	1.00E−08
f_{cec8}	2.00E+01	2.09E+01	2.00E+01
f_{cec9}	1.99E−01	4.37E+00	4.41E+00
f_{cec10}	4.58E+02	2.26E+00	5.35E+00
f_{cec11}	4.86E+01	9.24E−03	1.00E−08
f_{cec12}	1.43E+04	4.25E+04	2.25E+04
f_{cec13}	2.28E+00	4.44E+00	2.50E+00
f_{cec14}	2.36E+01	2.28E+01	2.27E+01
f_{cec15}	2.26E+02	2.00E+02	1.40E+02
f_{cec16}	3.80E+02	1.21E+01	1.15E+01
f_{cec17}	9.87E+02	2.12E+02	1.80E+02
f_{cec18}	1.05E+03	9.13E+02	9.12E+02
f_{cec19}	1.01E+03	9.14E+02	9.14E+02
f_{cec20}	1.03E+03	9.15E+02	9.15E+02
f_{cec21}	5.48E+02	6.55E+02	5.00E+02
f_{cec22}	1.31E+03	8.20E+02	8.16E+02
f_{cec23}	6.09E+02	6.97E+02	5.39E+02
f_{cec24}	2.00E+02	2.00E+02	2.00E+02
f_{cec25}	2.19E+02	2.14E+02	2.13E+02
No.lowest values	7	10	19

Table 4.3: Summary of the impact of ILS on iCMAES-ILS performance over 25 independent runs. Success (Failure) denotes that iCMAES-ILS does (not) finally improve upon IPOP-CMA-ES. The percentage of runs in which ILS obtains an improvement over IPOP-CMA-ES is given in the parenthesis.

ILS improvement⇒Success	$f_{cec8}(100\%)$, $f_{cec12}(12\%)$, $f_{cec13}(80\%)$, $f_{cec14}(28\%)$, $f_{cec15}(48\%)$, $f_{cec21}(24\%)$, $f_{cec23}(72\%)$, $f_{cec25}(4\%)$
ILS stagnation ⇒Success	f_{cec4}, f_{cec11}, f_{cec16}, f_{cec17}, f_{cec18}, f_{cec22}
ILS improvement⇒Failure	$f_{cec6}(56\%)$, $f_{cec9}(44\%)$
ILS stagnation ⇒Failure	f_{cec5}, f_{cec10}
Remark:	if ILS improves, iCMAES-ILS deploys ILS for the remaining budget. If ILS does not improve over the best solution found by IPOP-CMA-ES so far, iCMAES-ILS restarts IPOP-CMA-ES for the remaining budget.

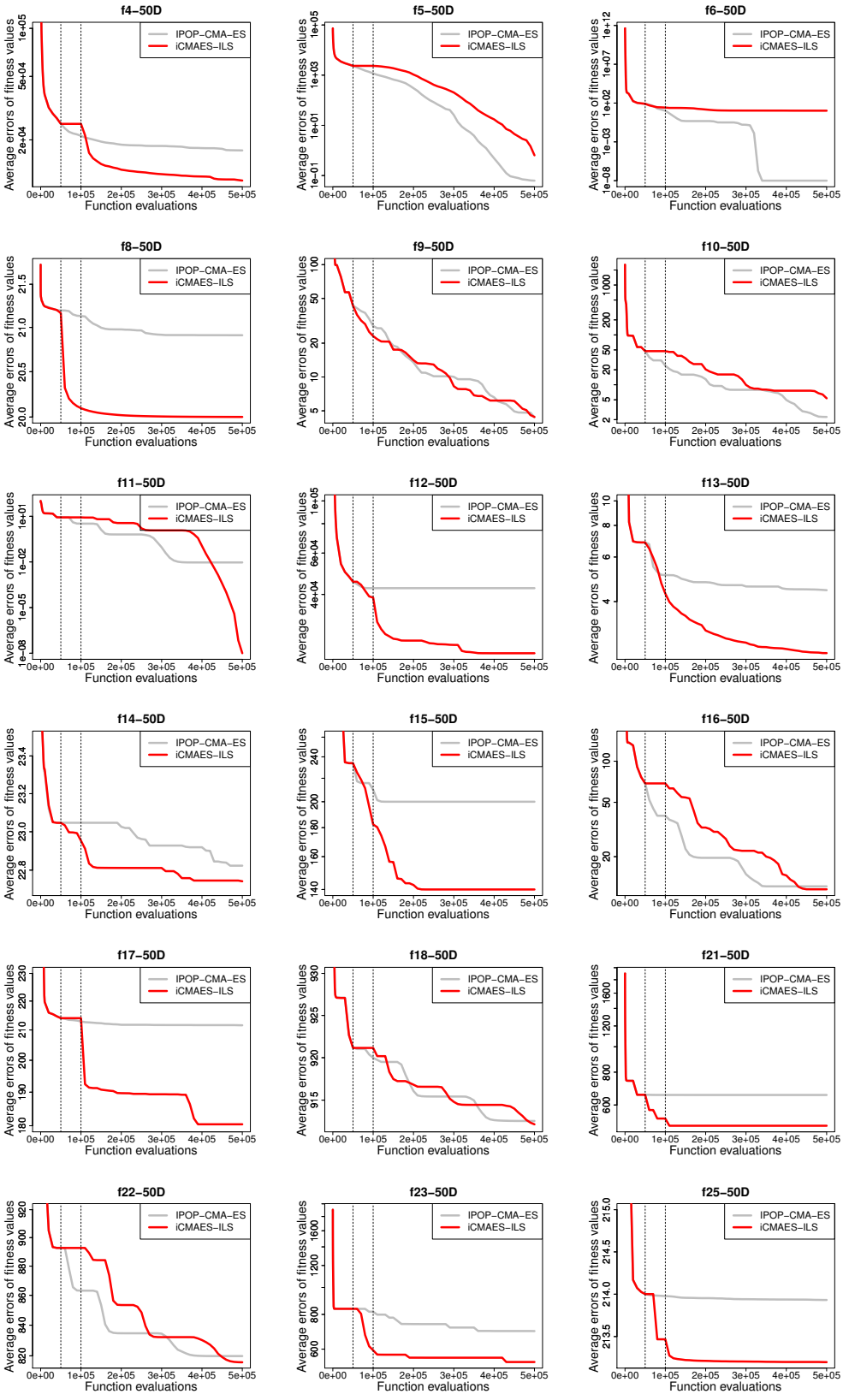


Figure 4.2: SQT curves of IPOP-CMA-ES and iCMAES-ILS. The ILS phase in iCMAES-ILS is shown between the two vertically dotted lines.

4.2.2 Performance evaluation of iCMAES-ILS

As a next step, we compare the performance of iCMAES-ILS to a number of alternative possibilities that include using ILS and IPOP-CMA-ES as standalone algorithms, defining combinations between IPOP-CMA-ES and ILS, and algorithms that integrate CMA-ES into other global optimization methods. We next introduce the other combinations of ILS and IPOP-CMA-ES and the CMA-ES hybrids.

1. We designed three different combinations between IPOP-CMA-ES and ILS. They are labeled as iCMAES-ILS-portfolio, iCMAES-ILS-relay and iCMAES-LTH-ILS, respectively. iCMAES-ILS-portfolio is a portfolio-based algorithm, in which ILS and IPOP-CMA-ES run in parallel each with half of *TotalBudget*. iCMAES-ILS-relay follows a high-level relay hybrid scheme [Talbi, 2002], in which ILS relays IPOP-CMA-ES in a number of cycles, in which ILS and IPOP-CMA-ES use in each run $relayP \times TotalBudget$ evaluations until the total budget is exhausted. iCMAES-LTH-ILS follows a low-level teamwork hybrid (LTH) scheme [Talbi, 2002], where ILS is embedded as a local search procedure before IPOP-CMA-ES restarts; ILS uses in each of its runs $lsP \times TotalBudget$ evaluations. We test values of $relayP$ and $lsP \in \{0.01, 0,025, 0.05, 0.1, 0.25, 0.5\}$. A schematic view of iCMAES-ILS-portfolio, iCMAES-ILS-relay and iCMAES-LTH-ILS is shown in Fig.4.3.

2. We compare iCMAES-ILS also to results obtained from two state-of-the-art algorithms, MA-LSCh-CMA [Molina et al., 2010a] and PS-CMA-ES [Müller et al., 2009], which use CMA-ES as a local search method and to the results of IPOP-CMA-ES-cec05 [Auger and Hansen, 2005]. In MA-LSCh-CMA [Molina et al., 2010a], CMA-ES is used as a local search procedure, and SSGA [Herrera et al., 1998] is specifically designed for diversification. In PS-CMA-ES [Müller et al., 2009], multiple instances of CMA-ES concurrently explore different search space regions and exchange information as the particle swarm does, then adapt the search direction and distribution in each CMA-ES instance. Recall that IPOP-CMA-ES-cec05 are the results for the CEC'05 benchmark functions taken from [Auger and Hansen, 2005]; IPOP-CMA-ES-cec05 is a Matlab version of IPOP-CMA-ES and it handles bound constraints by an approach based on penalty functions, which is described in [Hansen et al., 2009c].

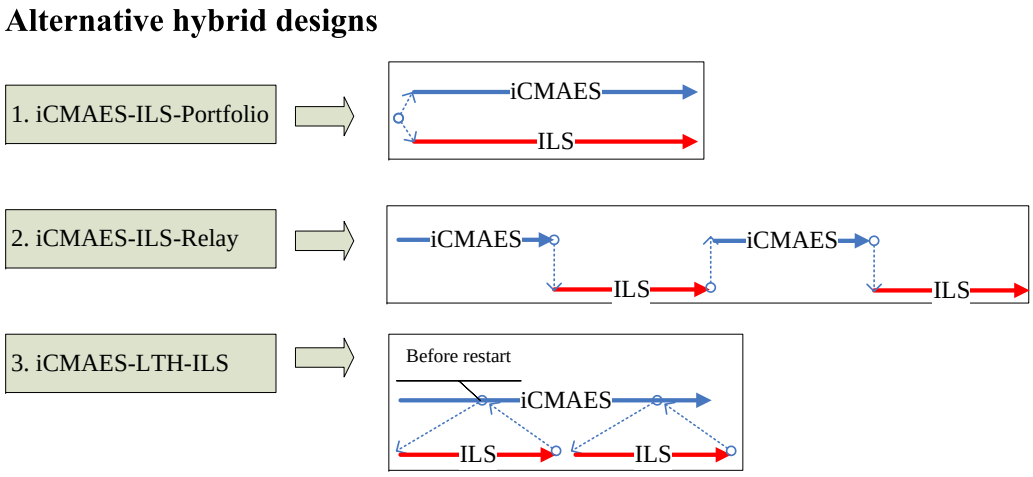


Figure 4.3: A schematic view of three combinations of IPOP-CMA-ES and ILS we used as comparison to our hybrid design. The empty points denote solutions that are transferred between IPOP-CMA-ES and ILS.

Table 4.4 gives average algorithm rankings on the CEC'05 benchmark function set over dimensions 30 and 50. We carry out a Friedman test at the 0.05 α-level to check for the significance of the differences between the best ranked algorithm and the others. The results of the Friedman tests are given in the last column of Table 4.4. ΔR_α is the minimum significant difference between the algorithms. The numbers in parenthesis are the differences of the sum of ranks relative to the best algorithm. Algorithms that are significantly worse than the best ranked algorithm are indicated in bold face. The experimental results identify iCMAES-ILS as the best ranked algorithm. In particular, iCMAES-ILS performs statistically significantly better than (i) its component algorithms IPOP-CMA-ES and ILS; (ii) most other hybrid algorithms including iCMAES-ILS-portfolio, the worst three ranked iCMAES-ILS-relay variants and all variants of iCMAES-LTH-ILS; (iii) various algorithms from literature, namely MA-LSCh-CMA [Molina et al., 2010a], PS-CMA-ES [Müller et al., 2009] and IPOP-CMA-ES-cec05 [Auger and Hansen, 2005]. No statistically significant difference was found between iCMAES-ILS and the best three ranked iCMAES-ILS-relay variants, which indicates that iCMAES-ILS-relay with few cycles reaches promising performance.

The detailed average error values of MA-LSCh-CMA, PS-CMA-ES, IPOP-CMA-ES-cec05 and iCMAES-ILS are given in Table 4.8 (results for the other algorithms are given in the supplementary material `http://iridia.ulb.ac.be/supp/IridiaSupp2012-017`. It is observed that on the uni-model functions (f_{cec1} to f_{cec5}) over dimension 30 and 50, iCMAES-ILS performs better than IPOP-

CMA-ES-cec05 on more functions than vice-versa. It indicates that iCMAES-ILS improves the performance of IPOP-CMA-ES-cec05 on both multi-modal and uni-model functions. In contrast, MA-LSCh-CMA and PS-CMA-ES cannot match such a good performance. Both perform worse than IPOP-CMA-ES on the uni-model functions. In particular, we compare iCMAES-ILS and IPOP-CMA-ES-cec05. A two-sided Wilcoxon matched-pairs signed-rank test at the 0.05 α-level is used to check whether the differences in the distribution of the average results obtained by the two algorithms are statistically significant. Table 4.5 shows that iCMAES-ILS statistically significantly improves upon IPOP-CMA-ES-cec05 on the CEC'05 benchmark set over dimensions 30 and 50 and on each of them. To the best of our knowledge, no other algorithm in the literature of the affine fields such as Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995], Differential Evolution (DE) [Storn and Price, 1997] and Artificial Bee Colony (ABC) [Karaboga and Basturk, 2007] has obtained statistically significantly better results than IPOP-CMA-ES-cec05 across the whole (that is, not only across a subset of the) CEC'05 benchmark set.

4.3 A tuned version of iCMAES-ILS

In this section, we examine how much iCMAES-ILS further profits from a fine-tuning of its parameters through automatic algorithm configuration techniques.

4.3.1 Automatic algorithm configuration

For the fine-tuning, we again employ iterated F-race [Birattari et al., 2010], a racing algorithm for algorithm configuration that is included in the publicly available `irace` package [López-Ibáñez et al., 2011].[1] For tuning iCMAES-ILS, we consider different variants of ILS, in particular, a tunable perturbation mechanism that biases towards the best solution. This mechanism follows the original equation $S = S_{\text{rand}} + r \times (S_{\text{best}} - S_{\text{rand}})$, but r is now a random number uniformly distributed in $[\mathit{BiasExtent}, 1)$, where *BiasExtent* is a tunable continuous parameter, $0 \leq \mathit{BiasExtent} < 1$. If $\mathit{BiasExtent} = 0$, the perturbation mechanism is same as the default in iCMAES-ILS. As the value of *BiasExtent* increases, an increasingly strong bias towards the best solution is induced. We also consider a tunable initial step size of ILS. This size is set to ss_r times the initial search range, where ss_r is a

[1] For details on the working of iterated F-race we refer to [Birattari et al., 2010] and the user guide in [López-Ibáñez et al., 2011]. See also Section 2.4.1 for some more details.

Table 4.4: The average ranks with respect to the average error values are given. The Friedman test at significance level $\alpha = 0.05$ is used. ΔR_α is the minimum significant difference 95.05. The numbers in parenthesis are the difference of the sum of ranks relative to the best algorithm iCMAES-ILS. Algorithms that are significantly different from iCMAES-ILS are indicated in bold face.

	Algorithms	Average Rank	(ΔR)
Proposed	iCMAES-ILS	7.28	(0)
Component algorithms	**IPOP-CMA-ES**	9.79	**(125.5)**
	ILS	14.39	**(355.5)**
Other hybrids	**iCMAES-ILS-portfolio**	9.29	**(100.5)**
	iCMAES-ILS-relay-025	8.15	(43.5)
	iCMAES-ILS-relay-01	8.52	(62.0)
	iCMAES-ILS-relay-05	8.69	(70.5)
	iCMAES-ILS-relay-0025	9.68	**(120.0)**
	iCMAES-ILS-relay-005	9.86	**(129.0)**
	iCMAES-ILS-relay-001	11.63	**(217.5)**
	iCMAES-LTH-ILS-025	9.46	**(109.0)**
	iCMAES-LTH-ILS-01	9.70	**(121.0)**
	iCMAES-LTH-ILS-005	9.78	**(125.0)**
	iCMAES-LTH-ILS-05	9.90	**(131.0)**
	iCMAES-LTH-ILS-0025	10.28	**(150.0)**
	iCMAES-LTH-ILS-001	10.33	**(152.5)**
Literature	**PS-CMA-ES**	9.95	**(133.5)**
	IPOP-CMA-ES-cec05	11.53	**(212.5)**
	MA-LSCh-CMA	11.79	**(225.5)**

tunable continuous parameter, $0 < ss_r \leq 1$. In total, we expose eleven parameters that directly control internal parameters of iCMAES-ILS. These eleven parameters are given in Table 4.7, together with the internal parameter of iCMAES-ILS controlled by each of them, their default value and the range considered for tuning.

The setting of iterated F-race we used are the default [López-Ibáñez et al., 2011]. The budget of each run of iterated F-race is set to 5 000 runs of iCMAES-ILS. The performance measure is the fitness error value of each instance. Iterated F-race handles two parameter types of iCMAES-ILS, continuous (r) and ordinal (o). The inputs are the parameter ranges given in Table 4.7 and a set of training instances.

As the training instances, we follow [Liao et al., 2013a] and consider a separation between training and test set to prevent the bias of the results due to potentially overtuning iCMAES-ILS. We use the 10-dimensional benchmark functions from the recent special issue of the Soft Computing journal (labeled as SOCO) [Herrera et al., 2010, Lozano et al., 2011]. This SOCO benchmark set consists of

Table 4.5: The numbers in parenthesis represent the number of times IPOP-CMA-ES-cec05 is better, equal or worse, respectively, compared to iCMAES-ILS. Error values lower than 10^{-8} are approximated to 10^{-8}.

IPOP-CMA-ES-cec05 vs iCMAES-ILS		
Dim 30 by f_{cec}: (4, 6, 15)†	Dim 50 by f_{cec}: (6, 4, 15)†	Total by f_{cec}: (10, 10, 30)†

† denotes a significant difference over the distribution of average errors between IPOP-CMA-ES-05 and iCMAES-ILS by a two-sided Wilcoxon matched-pairs signed-ranks test at the 0,05 α-level.

19 functions. Four of these functions are the same as in the CEC'05 benchmark set and are therefore removed from the training set. The training functions are then sampled in a random order from all possible such functions. The number of function evaluations of each run is equal to $5\,000 \times D$ according to the termination criteria in the definition of the SOCO benchmark set.

The tuned settings of iCMAES-ILS are presented in the last column of Table 4.7 and we label the tuned iCMAES-ILS as iCMAES-ILSt. The tuned parameter settings of the IPOP-CMA-ES component imply a larger exploration at the beginning of the search than in the default settings; the tuned parameter setting of the ILS component imply a shorter local search, a larger initial step size and a slightly stronger bias by the perturbation than in the default version.

4.3.2 Performance evaluation of iCMAES-ILSt

We apply iCMAES-ILSt to the CEC'05 benchmark functions of 30 and 50 dimensions as described in the experimental setup in Section 4.2 and compare its results to iCMAES-ILS, a tuned ILS (labeled as ILSt) and a tuned IPOP-CMA-ES [Liao et al., 2013a] (labeled as IPOP-CMA-ESt). The parameters and the parameter ranges considered for tuning ILS and IPOP-CMA-ES as well as the tuning setup and effort is the same as for tuning iCMAES-ILS. The parameters of ILSt and IPOP-CMA-ESt are given in the supplementary information pages `http://iridia.ulb.ac.be/supp/IridiaSupp2012-017`. Table 4.6 summarizes the average ranking obtained by iCMAES-ILS, ILSt, IPOP-CMA-ESt and iCMAES-ILSt. A Friedman test at the 0.05 α-level is used to check for the significance of the differences between the best ranked algorithm iCMAES-ILSt and the other algorithms. We find that iCMAES-ILSt is statistically significantly better than ILSt and IPOP-CMA-ESt; iCMAES-ILSt also statistically significantly improves upon iCMAES-ILS. The detailed average error values obtained by ILSt, IPOP-CMA-ESt, iCMAES-ILS and iCMAES-ILSt are listed in Table 4.9. These

results further indicate the high performance that is reached by the design of iCMAES-ILS and they also show the further improvement obtained through automatic algorithm configuration.

Table 4.6: The average ranks with respect to the average error values are given. The Friedman test at significance level $\alpha = 0.05$ is used. ΔR_α are the minimum significant difference 13.24 for dimension 30 (left table) and 13.74 for dimension 50 (right table), respectively. The numbers in parenthesis are the difference of the sum of ranks relative to the best algorithm iCMAES-ILSt. Algorithms that are significantly different from iCMAES-ILSt are indicated in bold face.

Dim=30	Algorithms	Average Rank	(ΔR)
	iCMAES-ILSt	1.82	(0)
	iCMAES-ILS	2.36	**(13.5)**
	IPOP-CMA-ESt	2.38	**(14.0)**
	ILSt	3.44	**(40.5)**

Dim=50	Algorithms	Average Rank	(ΔR)
	iCMAES-ILSt	1.80	(0)
	IPOP-CMA-ESt	2.38	**(14.5)**
	iCMAES-ILS	2.44	**(16.0)**
	ILSt	3.38	**(39.5)**

Table 4.7: Parameters that have been considered for tuning. Given are the default values of the parameters and the ranges we considered for tuning. The last column is the tuned parameter settings.

Algorithm Components	**Parameter**		**Internal parameter**	**Default**	**Range**	**Tuned**
IPOP-CMA-ES	a	Init pop size: λ_0	$= 4 + \lfloor a \ln(D) \rfloor$	3	r $[1, 10]$	9.687
	b	Parent size: μ	$= \lfloor \lambda / b \rfloor$	2	r $[1, 5]$	1.614
	c	Init step size: σ_0	$= c \cdot (B - A)$	0.5	r $(0, 1]$	0.6825
	d	IPOP factor: *ipop*	$= d$	2	r $[1, 4]$	3.245
	e	*stopTolFun*	$= 10^e$	-12	r $[-20, -6]$	-9.023
	f	*stopTolFunHist*	$= 10^f$	-20	r $[-20, -6]$	-10.82
	g	*stopTolX*	$= 10^g$	-12	r $[-20, -6]$	-16.26
ILS	i_r	*LSIterations*	$= i_r \times D$	1.5	o $(1, 1.25, 1.5, 1.75, 2)$	1
	ss_r	*ss*	$= ss_r \times (B - A)$	0.5	r $(0, 1]$	0.6703
	$bias_e$	*BiasExtent*	$= bias_e$	0	r $[0, 1)$	0.01910
Competition Design	$comp_r$	*CompBudget*	$= comp_r \times TotalBudget$	0.1	o $(0.05, 0.1, 0.15, 0.2, 0.25, 0.3)$	0.15

Table 4.8: The average errors obtained by IPOP-CMA-ES-CEC05, MA-LSch-CMA, PS-CMA-ES and iCMAES-ILS for each CEC'05 function. Error values lower than 10^{-8} are approximated to 10^{-8}. The lowest average errors values are highlighted.

Dim	f_{cec}	IPOP-CMA-ES-CEC05	MA-LSch-CMA	PS-CMA-ES	iCMAES-ILS
30 Dim	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	1.00E−08	2.75E+04	2.96E+04	1.00E−08
	f_{cec4}	1.11E+04	3.02E+02	4.56E+03	1.74E+02
	f_{cec5}	1.00E−08	1.26E+03	2.52E+01	1.00E−08
	f_{cec6}	1.00E−08	1.12E+00	1.15E+01	8.67E+00
	f_{cec7}	1.00E−08	1.75E−02	1.00E−08	1.00E−08
	f_{cec8}	2.01E+01	2.00E+01	2.00E+01	2.00E+01
	f_{cec9}	9.38E−01	1.00E−08	8.76E−01	7.16E−01
	f_{cec10}	1.65E+00	2.25E+01	5.57E−01	3.10E+00
	f_{cec11}	5.48E+00	2.15E+01	7.10E+00	1.87E−02
	f_{cec12}	4.43E+04	1.67E+03	8.80E+02	2.60E+03
	f_{cec13}	2.49E+00	2.03E+00	2.05E+00	1.41E+00
	f_{cec14}	1.29E+01	1.25E+01	1.24E+01	1.30E+01
	f_{cec15}	2.08E+02	3.00E+02	1.37E+02	1.36E+02
	f_{cec16}	3.50E+01	1.26E+02	1.59E+01	1.48E+01
	f_{cec17}	2.91E+02	1.83E+02	9.15E+01	2.11E+02
	f_{cec18}	9.04E+02	8.98E+02	9.05E+02	8.96E+02
	f_{cec19}	9.04E+02	9.01E+02	8.85E+02	8.96E+02
	f_{cec20}	9.04E+02	8.96E+02	9.05E+02	8.96E+02
	f_{cec21}	5.00E+02	5.12E+02	5.00E+02	5.00E+02
	f_{cec22}	8.03E+02	8.80E+02	8.43E+02	8.12E+02
	f_{cec23}	5.34E+02	5.34E+02	5.34E+02	5.33E+02
	f_{cec24}	9.10E+02	2.00E+02	2.00E+02	2.00E+02
	f_{cec25}	2.11E+02	2.14E+02	2.10E+02	2.03E+02
50 Dim	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.00E−08	3.06E−02	7.36E−06	1.00E−08
	f_{cec3}	1.00E−08	3.21E+04	9.10E+04	1.00E−08
	f_{cec4}	4.68E+05	3.23E+03	2.17E+04	1.11E+04
	f_{cec5}	2.85E+00	2.69E+03	1.79E+03	6.40E−01
	f_{cec6}	1.00E−08	4.10E+00	2.91E+01	1.04E+01
	f_{cec7}	1.00E−08	5.40E−03	1.00E−08	1.00E−08
	f_{cec8}	2.01E+01	2.00E+01	2.00E+01	2.00E+01
	f_{cec9}	1.39E+00	1.00E−08	5.45E+00	4.41E+00
	f_{cec10}	1.72E+00	5.01E+01	5.33E+00	5.35E+00
	f_{cec11}	1.17E+01	4.13E+01	1.59E+01	1.00E−08
	f_{cec12}	2.27E+05	1.39E+04	6.90E+03	2.25E+04
	f_{cec13}	4.59E+00	3.15E+00	4.15E+00	2.50E+00
	f_{cec14}	2.29E+01	2.22E+01	2.15E+01	2.27E+01
	f_{cec15}	2.04E+02	3.72E+02	1.25E+02	1.40E+02
	f_{cec16}	3.09E+01	6.90E+01	1.62E+01	1.15E+01
	f_{cec17}	2.34E+02	1.47E+02	9.13E+01	1.80E+02
	f_{cec18}	9.13E+02	9.41E+02	8.70E+02	9.12E+02
	f_{cec19}	9.12E+02	9.38E+02	9.13E+02	9.14E+02
	f_{cec20}	9.12E+02	9.28E+02	9.09E+02	9.15E+02
	f_{cec21}	1.00E+03	5.00E+02	6.62E+02	5.00E+02
	f_{cec22}	8.05E+02	9.14E+02	8.63E+02	8.16E+02
	f_{cec23}	1.01E+03	5.39E+02	8.12E+02	5.39E+02
	f_{cec24}	9.55E+02	2.00E+02	2.00E+02	2.00E+02
	f_{cec25}	2.15E+02	2.21E+02	2.14E+02	2.13E+02
No.lowest values		16	13	21	30

Table 4.9: The average errors obtained by ILSt, IPOP-CMA-ESt, iCMAES-ILS and iCMAES-ILSt for each CEC'05 function. Error values lower than 10^{-8} are approximated to 10^{-8}. The lowest average errors values are highlighted.

Dim	f_{cec}	ILSt	IPOP-CMA-ESt	iCMAES-ILS	iCMAES-ILSt
30 Dim	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	2.13E+05	1.00E−08	1.00E−08	1.00E−08
	f_{cec4}	2.25E+04	1.00E−08	1.74E+02	1.00E−08
	f_{cec5}	7.01E+03	1.00E−08	1.00E−08	1.00E−08
	f_{cec6}	7.80E+01	1.00E−08	8.67E+00	1.00E−08
	f_{cec7}	1.18E−02	1.00E−08	1.00E−08	1.00E−08
	f_{cec8}	2.00E+01	2.08E+01	2.00E+01	2.00E+01
	f_{cec9}	1.00E−08	1.99E+00	7.16E−01	5.70E−01
	f_{cec10}	1.42E+02	1.59E+00	3.10E+00	1.63E+00
	f_{cec11}	2.43E+01	5.09E−05	1.87E−02	4.08E−02
	f_{cec12}	1.91E+03	4.22E+02	2.60E+03	6.11E+02
	f_{cec13}	1.16E+00	2.53E+00	1.41E+00	1.69E+00
	f_{cec14}	1.39E+01	1.10E+01	1.30E+01	1.16E+01
	f_{cec15}	2.80E+02	2.00E+02	1.36E+02	1.16E+02
	f_{cec16}	2.76E+02	1.11E+01	1.48E+01	1.18E+01
	f_{cec17}	4.14E+02	2.08E+02	2.11E+02	1.67E+02
	f_{cec18}	9.90E+02	9.04E+02	8.96E+02	8.96E+02
	f_{cec19}	9.81E+02	9.04E+02	8.96E+02	8.96E+02
	f_{cec20}	9.92E+02	9.04E+02	8.96E+02	8.96E+02
	f_{cec21}	5.52E+02	5.00E+02	5.00E+02	4.89E+02
	f_{cec22}	1.17E+03	8.17E+02	8.12E+02	8.11E+02
	f_{cec23}	5.34E+02	5.34E+02	5.33E+02	5.33E+02
	f_{cec24}	2.00E+02	2.00E+02	2.00E+02	2.00E+02
	f_{cec25}	2.10E+02	2.09E+02	2.03E+02	2.02E+02
50 Dim	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	2.92E+05	1.00E−08	1.00E−08	1.00E−08
	f_{cec4}	8.45E+04	1.00E−08	1.11E+04	1.00E−08
	f_{cec5}	1.35E+04	1.00E−08	6.40E−01	1.00E−08
	f_{cec6}	3.75E+01	1.00E−08	1.04E+01	1.01E+01
	f_{cec7}	4.04E−03	1.00E−08	1.00E−08	1.00E−08
	f_{cec8}	2.00E+01	2.10E+01	2.00E+01	2.00E+01
	f_{cec9}	1.00E−08	4.18E+00	4.41E+00	2.11E+00
	f_{cec10}	4.06E+02	2.71E+00	5.35E+00	2.43E+00
	f_{cec11}	4.78E+01	6.03E−02	1.00E−08	1.13E−01
	f_{cec12}	9.60E+03	4.69E+03	2.25E+04	4.24E+03
	f_{cec13}	1.92E+00	4.70E+00	2.50E+00	2.72E+00
	f_{cec14}	2.37E+01	2.09E+01	2.27E+01	2.12E+01
	f_{cec15}	2.92E+02	2.00E+02	1.40E+02	1.20E+02
	f_{cec16}	3.33E+02	5.34E+00	1.15E+01	5.26E+00
	f_{cec17}	7.01E+02	6.36E+01	1.80E+02	4.42E+01
	f_{cec18}	9.89E+02	9.13E+02	9.12E+02	9.06E+02
	f_{cec19}	9.80E+02	9.13E+02	9.14E+02	9.10E+02
	f_{cec20}	1.01E+03	9.13E+02	9.15E+02	9.10E+02
	f_{cec21}	5.60E+02	7.05E+02	5.00E+02	5.32E+02
	f_{cec22}	1.33E+03	8.19E+02	8.16E+02	8.23E+02
	f_{cec23}	5.67E+02	7.30E+02	5.39E+02	5.39E+02
	f_{cec24}	2.00E+02	2.00E+02	2.00E+02	2.00E+02
	f_{cec25}	2.20E+02	2.13E+02	2.13E+02	2.13E+02
No.lowest values		12	23	22	36

4.4 Comparisions of iCMAES-ILS and UACOR$^+$

As a final step, we compare iCMAES-ILS to UACOR$^+$, the UACOR version that included CMA-ES as a local search as proposed in Chapter 3. For this analysis, we consider the average errors on each function, and the distribution of the average errors. For a fair comparison, the same tuning setup and effort is considered for obtaining the parameter settings.

We first re-tuned UACOR$^+$-c allowing this time also to tune the parameters of CMA-ES (resulting in a configuration labeled as UACOR$^+$-ct). For this, we used tuning setup and effort to be the same as tuning iCMAES-ILS to result in iCMAES-ILSt. The parameter values of the tuned CMA-ES in UACOR$^+$-ct are population size $\lambda = 4 + \lfloor 9.600 \ln(D) \rfloor$, parent size $\mu = \lfloor \lambda / 1.452 \rfloor$, $stopTolFun 10^{-8.854}$, $stopTolFunHist = 10^{-9.683}$ and $stopTolX = 10^{-12.55}$. These parameter values are derived from Liao et al. [2013a]. The initial step size of CMA-ES in UACOR$^+$-ct is defined adaptively as in UACOR$^+$, not a fixed parameter value. We then ran UACOR$^+$-ct on the CEC'05 benchmark functions of dimensions 30 and 50, and we compare the results of UACOR$^+$-ct to those of IPOP-CMA-ES-cec05, UACOR$^+$-c, and iCMAES-ILSt. From the correlation plots in the Fig. 4.4, we can clearly observe that UACOR$^+$-ct statistically significantly improves upon UACOR$^+$-c; UACOR$^+$-ct statistically significantly improves also upon IPOP-CMA-ES-cec05. Recall that UACOR$^+$-c did not improve in a statistically significant way upon IPOP-CMA-ES-cec05. These comparisons clearly show the effectiveness of the tuned parameter values. Nevertheless, iCMAES-ILSt still performs statistically significantly better than UACOR$^+$-ct.

Next we investigate the performance of iCMAES-ILSt on the SOCO benchmark set where UACOR$^+$-s has obtained a very good performance and IPOP-CMA-ES performs rather poorly. We tuned iCMAES-ILSt with the same tuning setup and effort as for tuning UACOR$^+$-s. The tuned version is labeled as iCMAES-ILSt-s. We then tested iCMAES-ILSt-s on the SOCO benchmark functions of dimensions 100 and 200. As shown in the correlation plot in the Fig. 4.5, iCMAES-ILSt-s statistically significantly improves upon a tuned IPOP-CMA-ES, labeled as iCMAESt-s, with the same tuning setup and effort as for tuning iCMAES-ILSt-s. However, the results of iCMAES-ILSt-s is statistically significant worse than those of UACOR$^+$-s. The worse results obtained by iCMAES-ILSt-s may be explained by the use of IPOP-CMA-ES that is known to perform poorly on these benchmark functions. However, it is noteworthy that iCMAES-ILSt-s has impressive results on function f_{soco8}, which is a hyperellipsoid rotated in all directions. Other

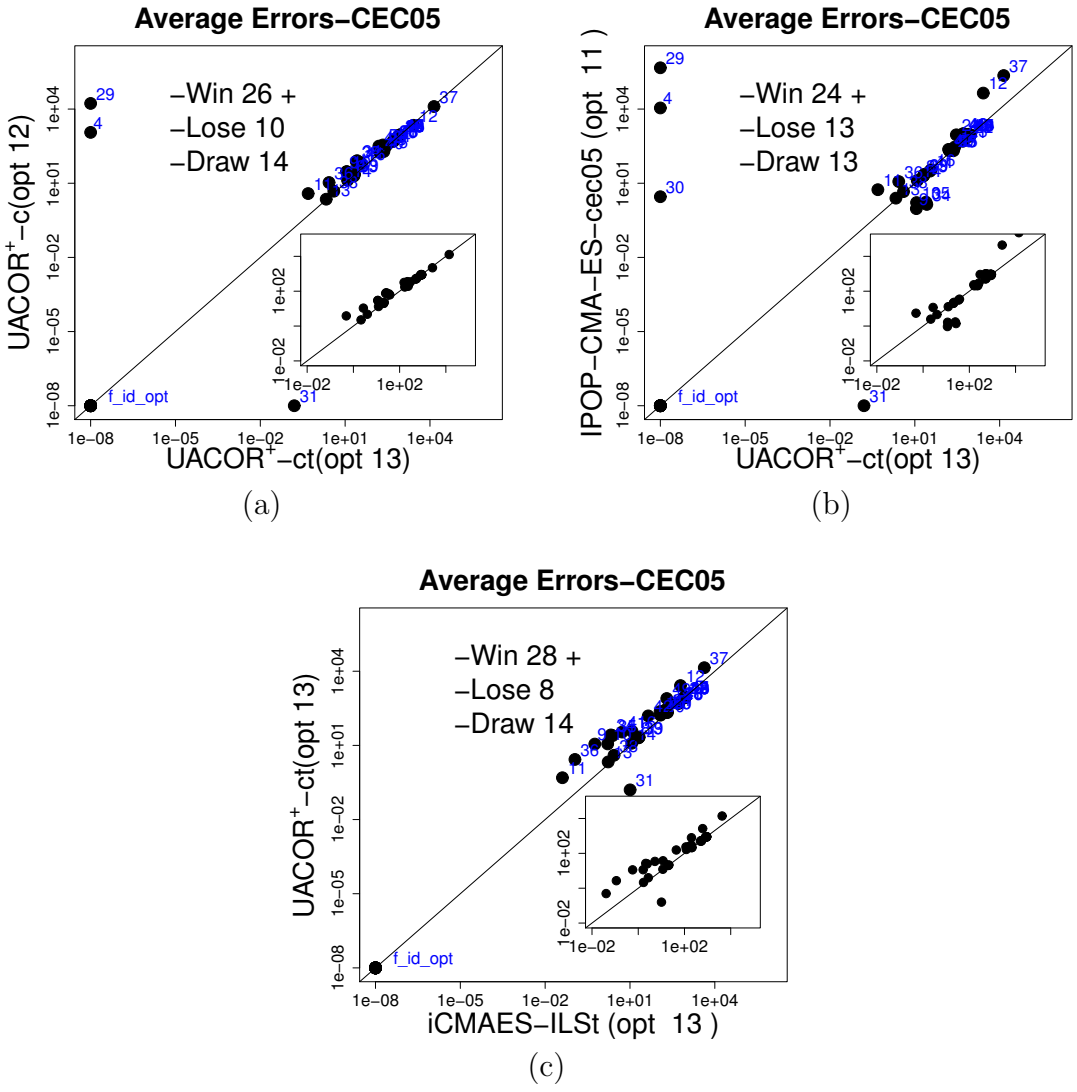


Figure 4.4: Correlation plots of UACOR$^+$-c and UACOR$^+$-ct, IPOP-CMA-ES-cec05 and UACOR$^+$-ct, UACOR$^+$-ct and iCMAES-ILSt on 25 CEC05 benchmark functions of dimensions 30 and 50 (the indexes of 50 dimensional functions are labeled from 26 to 50). Each point represents the average error value obtained by either of the two algorithms. A point on the upper triangle delimited by the diagonal indicates better performance for the algorithm on the x-axis; a point on the lower right triangle indicates better performance for the algorithm on the y-axis. The number labeled beside some outstanding points represent the index of the corresponding function. The comparison is conducted based on average error values and the comparison results of the algorithm on the x-axis are presented in the form of -win, -draw, -lose, respectively. We marked with a + symbol those cases in which there is a statistically significant difference at the 0.05 α-level between the algorithms checked by a Friedman test and its post tests over IPOP-CMA-ES-cec05, UACOR$^+$-c, UACOR$^+$-ct and iCMAES-ILSt. The number of opt on the axes shows the number of means that is lower than the zero threshold, obtained by the corresponding algorithm.

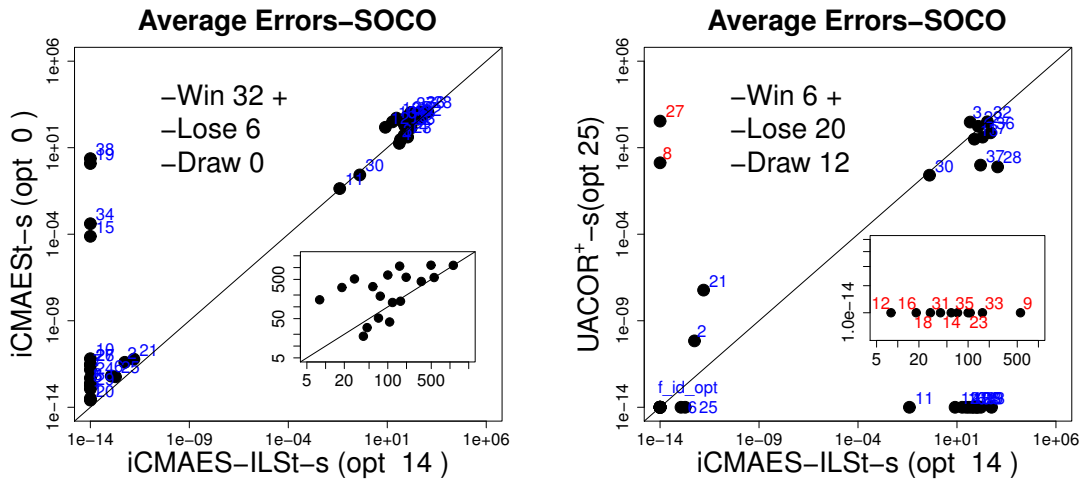


Figure 4.5: Correlation plots of iCMAESt-s and iCMAES-ILSt-s, UACOR$^+$-s and iCMAES-ILSt-s on 19 SOCO function of dimensions 100 and 200 (the indexes of 200 dimensional functions are labeled from 20 to 38); Each point represents the average error value obtained by either of the two algorithms. A point on the upper triangle delimited by the diagonal indicates better performance for the algorithm on the x-axis; a point on the lower right triangle indicates better performance for the algorithm on the y-axis. The number labeled beside some outstanding points represent the index of the corresponding function. The comparison is conducted based on average error values and the comparison results of the algorithm on the x-axis are presented in the form of -win, -draw, -lose, respectively. We marked with a + symbol those cases in which there is a statistically significant difference at the 0.05 α-level between the algorithms checked by a Friedman test and its post tests over iCMAESt-s, iCMAES-ILSt-s and UACOR$^+$-s. The number of opt on the axes shows the number of means that is lower than the zero threshold, obtained by the corresponding algorithm.

algorithms such as UACOR$^+$-s, other enhanced DE or PSO variants that have been benchmarked on the SOCO benchmark functions as well, could not find the optimum of this function.

4.5 Summary

In this chapter, we have introduced iCMAES-ILS, a hybrid algorithm where IPOP-CMA-ES and ILS compete in an initial competition phase for further deployment. The main advantages of iCMAES-ILS are its simple design and its effectiveness as demonstrated through the excellent performance on the CEC'05 benchmark functions set. The computational results with a default parameter settings and further fine-tuned parameter settings establish iCMAES-ILS as a state-of-the-art algorithm for continuous optimization. In fact, iCMAES-ILS improves statistically significantly over IPOP-CMA-ES, ILS, MA-LSCh-CMA, PS-CMA-ES and it gives better results than other possible hybrid designs such as iCMAES-ILS-portfolio, iCMAES-ILS-relay and iCMAES-LTH-ILS. At the end of the chapter, we discuss the comparisons of iCMAES-ILS and UACOR$^+$ proposed in the Chapter 3 on the CEC'05 and SOCO benchmark functions set, respectively. As a result, each of iCMAES-ILS and UACOR$^+$ shows own advantages with respect to the two different benchmark functions sets. iCMAES-ILS performs statistically significantly better than UACOR$^+$ on the CEC'05 benchmark functions set; UACOR$^+$ performs statistically significantly better than iCMAES-ILS on the SOCO benchmark functions set.

Chapter 5

Mixed discrete-continuous optimization

Many real-world optimization problems can be modeled using combinations of continuous and discrete variables. Due to the practical relevance of these discrete-continuous problems, a number of optimization algorithms for tackling them have been proposed. These algorithms are mainly based on Genetic Algorithms [Goldberg, 1989], Differential Evolution [Storn and Price, 1997], Particle Swarm Optimization [Kennedy and Eberhart, 2001] and Pattern Search [Torczon, 1997]. Mixed integer programming (linear or nonlinear) refers to mathematical programming with continuous and integer variables in the (linear or nonlinear) objective function and constraints [Bussieck and Pruessner, 2003]. In this thesis, we consider more general, mixed discrete-continuous optimization problems where the discrete variables can be ordinal or categorical. Ordinal variables exhibit a natural ordering relation (e.g., integers or $\{\text{small}, \text{medium}, \text{large}\}$) and are usually handled using a *continuous relaxation approach* [Dimopoulos, 2007, Gao and Hailu, 2010, Guo et al., 2004, Lampinen and Zelinka, 1999a,b, Mashinchi et al., 2011, Rao and Xiong, 2005, Turkkan, 2003]. Categorical variables take their values from a finite set of categories [Abramson et al., 2009], which often identify non-numeric elements of an unordered set (e.g., colors, shapes or types of material). Categorical variables do not have a natural ordering relation and therefore require the use of a *categorical optimization approach* [Abramson, 2002, 2004, Abramson et al., 2009, Audet and Dennis, 2001, Deb and Goyal, 1998, Kokkolaras et al., 2001, Ocenasek and Schwarz, 2002] that does not assume any ordering relation. To the best of our knowledge, the approaches to mixed discrete-continuous problems available in the literature are targeted to either handle mixtures of continuous and ordinal variables or mixtures of continuous and categorical variables. In other words, they do not consider the possibility that the formulation of a problem may involve at the same time the three types of variables. Hence, there is a need for algorithms that allow the explicit declaration of each variable as either continuous, ordinal or

categorical.

To tackle mixed discrete-continuous optimization problems, in this chapter, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$, an ant colony optimization and a covariance matrix adaptation evolution strategy algorithms are presented, respectively. In $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$, the decision variables of an optimization problem can be declared as continuous, ordinal, or categorical, which allows the algorithm to treat them adequately. $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ include three solution generation mechanisms: a continuous optimization mechanism, a continuous relaxation mechanism for ordinal variables, and a categorical optimization mechanism for categorical variables. Together, these mechanisms allow $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ to tackle mixed variable optimization problems. We also propose a set of artificial mixed discrete-continuous benchmark functions, which can simulate discrete variables as ordered or categorical. We use them to automatically tune $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$'s parameters and benchmark their performance. Finally, we test $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ on various real-world continuous and mixed discrete-continuous engineering optimization problems. Comparisons with results from the literature demonstrate the effectiveness and robustness of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ on mixed discrete-continuous optimization problems.

The original $\text{ACO}_{\mathbf{MV}}$ algorithm was proposed by Socha in his PhD thesis [Socha, 2008]. Starting from this initial work of Socha, we re-implemented the $\text{ACO}_{\mathbf{MV}}$ algorithm in C++, which reduced strongly the computation time when compared to the original implementation in R. In this thesis, we refine the original $\text{ACO}_{\mathbf{MV}}$ algorithm and implement $\text{ACO}_{\mathbf{MV}}$ with a restart operator. We also propose and generate a large set of new benchmark functions for mixed discrete-continuous optimization problems. We use these benchmark problems for a detailed study of specific aspects of $\text{ACO}_{\mathbf{MV}}$ and to derive high-performance parameter settings through automatic tuning tools. In addition, also methodological improvements were made. While in the original work of Socha $\text{ACO}_{\mathbf{MV}}$ was specifically tuned on each real-world engineering test problem on which $\text{ACO}_{\mathbf{MV}}$ was finally evaluated, now one single parameter setting, obtained by a tuning process on new benchmark functions, which are independent of the engineering test functions, is applied for the final test on mixed discrete-continuous engineering problems. Another contribution here is that we increased further the test bed of engineering problems that were originally considered by Socha. Finally, the development of $\text{CES}_{\mathbf{MV}}$ is an original contribution of this thesis.

The chapter is organized as follows. In Section 5.1, we first present a new set of artificial, mixed discrete-continuous benchmark functions, which can simulate

discrete variables as ordered or categorical. In section 5.2, we detail $\text{ACO}_{\mathbf{MV}}$; next in Section 5.3, we introduce three ways of handling categorical variables in a extension of CMA-ES to mixed discrete-continuous optimization problems. The three resulting variants are labeled as $\text{CES}_{\mathbf{MV}}$ CES-RoundC and CES-RelayC, respectively. Afterwards, the automatic tuning of the parameters of $\text{ACO}_{\mathbf{MV}}$, $\text{CES}_{\mathbf{MV}}$, CES-RoundC and CES-RelayC is presented in Section 5.4.1. We first examine the performance of $\text{CES}_{\mathbf{MV}}$, CES-RoundC and CES-RelayC in Section 5.4.2, and identify $\text{CES}_{\mathbf{MV}}$ as the best performing variant. Finally, we compare $\text{CES}_{\mathbf{MV}}$ and $\text{ACO}_{\mathbf{MV}}$ on mixed discrete-continuous benchmark functions. In Section 5.5, we apply $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ to mixed-variable engineering benchmark problems and compare the results of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ with those found in the literature.

5.1 Artificial mixed discrete-continuous benchmark functions

The real-world mixed-variable problems cannot be easily parametrized and flexibly manipulated for investigating the performance of mixed-variable optimization algorithms in a systematic way. In this section, we propose a set of new, artificial mixed-variable benchmark functions that allow the definition of a controlled environment for the investigation of algorithm performance and the automatic tuning of algorithm parameters [Birattari et al., 2010, Hutter et al., 2009b]. These new artificial benchmark functions will be used in the evaluation of the various algorithms for mixed discrete-continuous optimization problems that are proposed in this thesis. Our proposed artificial mixed-variable benchmark functions are defined in Table 5.1. These functions originate from some typical continuous functions of the CEC'05 benchmark set [Suganthan et al., 2005]. The functions we have chosen here represent the most widely used continuous optimization functions used in the literature. The decision variables consist of continuous and discrete variables; n is the total number of variables and $\mathbf{M}$ is a random, normalized, $n \times n$ rotation matrix. The problems' global optima $\vec{S}^*$ are shifted in order not to give an advantage to methods that may have a bias towards the origin of the search space [Eiben and Bäck, 1997]. The proposed benchmarks allow three settings for discrete variables. The first setting consists of only ordinal variables; the second setting consists of only categorical variables; and the third setting consists of both ordinal and categorical variables. *MinRange* and *MaxRange* denote the lower and upper bound of variable domains, respectively.

We use the two-dimensional, not shifted, randomly rotated Ellipsoid mixed-

Table 5.1: Artificial mixed-variable benchmark functions. In the upper part the objective functions are defined; the variables are defined in the lower part of the table.

Objective functions		
$f_{Ellipsoid_{MV}}(\vec{x}) = \sum_{i=1}^{n}(\beta^{\frac{i-1}{n-1}} z_i)^2$,		
$f_{Ackley_{MV}}(\vec{x}) = -20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n}(z_i^2)}} - e^{\frac{1}{n}\sum_{i=1}^{n}(\cos(2\pi z_i))} + 20 + e$,		
$f_{Rastrigin_{MV}}(\vec{x}) = 10n + \sum_{i=1}^{n}(z_i^2 - 10\cos(2\pi z_i^2))$,		
$f_{Rosenbrock_{MV}}(\vec{x}) = \sum_{i=1}^{n-1}[100(z_{i+1} - z_i^2)^2 + (z_i - 1)^2]$,		
$f_{Sphere_{MV}}(\vec{x}) = \sum_{i=1}^{n} z_i^2$,		
$f_{Griewank_{MV}}(\vec{x}) = \frac{1}{4\,000}\sum_{i=1}^{n} z_i^2 - \prod_{i=1}^{n}\cos(\frac{z_i}{\sqrt{i}}) + 1$,		
Definition of mixed variables		
1st setting:	$\vec{z} = \mathbf{M}(\vec{x} - \vec{S^*}) : \vec{S^*} = (R_*^1\, R_*^2 \dots R_*^r\, O_*^1\, O_*^2 \dots O_*^o)^t$,	
	$if(f_{Rosenbrock_{MV}}), \vec{z} = \vec{z} + 1$,	
	$\vec{S^*}$ is a shift vector, $n = o + r$,	
	$\vec{x} = (R^1\, R^2 \dots R^r\, O^1\, O^2 \dots O^o)^t$,	
	$R^i \in (MinRange^i, MaxRange^i)$,	$i = 1, \dots, r$
	$O^i \in \mathbf{T}, \mathbf{T} = \{\theta_1, \theta_2, ..., \theta_{t_i}\} : \forall l\ \theta_{t_l} \in (MinRange^i, MaxRange^i)$	$i = 1, \dots, o$
2nd setting:	$\vec{z} = \mathbf{M}(\vec{x} - \vec{S^*}) : \vec{S^*} = (R_*^1\, R_*^2 \dots R_*^r\, C_*^1\, C_*^2 \dots C_*^c,)^t$,	
	$if(f_{Rosenbrock_{MV}}), \vec{z} = \vec{z} + 1$,	
	$\vec{S^*}$ is a shift vector, $n = c + r$,	
	$\vec{x} = (R^1\, R^2 \dots R^r\, C^1\, C^2 \dots C^c)^t$,	
	$R^i \in (MinRange^i, MaxRange^i)$,	$i = 1, \dots, r$
	$C^i \in \mathbf{T}, \mathbf{T} = \{\theta_1, \theta_2, ..., \theta_{t_i}\} : \forall l\ \theta_{t_l} \in (MinRange^i, MaxRange^i)$	$i = 1, \dots, c$
3rd setting:	$\vec{z} = \mathbf{M}(\vec{x} - \vec{S^*}) : \vec{S^*} = (R_*^1\, R_*^2 \dots R_*^r\, O_*^1\, O_*^2 \dots O_*^o\, C_*^1\, C_*^2 \dots C_*^c)^t$,	
	$if(f_{Rosenbrock_{MV}}), \vec{z} = \vec{z} + 1$,	
	$\vec{S^*}$ is a shift vector, $n = o + c + r$,	
	$\vec{x} = (R^1\, R^2 \dots R^r\, O^1\, O^2 \dots O^o\, C^1\, C^2 \dots C^c)^t$,	
	$R^i \in (MinRange^i, MaxRange^i)$,	$i = 1, \dots, r$
	$O^i \in \mathbf{T}, \mathbf{T} = \{\theta_1, \theta_2, ..., \theta_{t_i}\} : \forall l\ \theta_{t_l} \in (MinRange^i, MaxRange^i)$	$i = 1, \dots, o$
	$C^i \in \mathbf{T}, \mathbf{T} = \{\theta_1, \theta_2, ..., \theta_{t_i}\} : \forall l\ \theta_{t_l} \in (MinRange^i, MaxRange^i)$	$i = 1, \dots, c$

variable function as an example of how to construct artificial mixed-variable benchmark functions. The way how to construct mixed-variable functions was originally proposed by Socha [2008]. We re-describe it in the following. We start with a

two-dimensional, continuous, not shifted, randomly rotated Ellipsoid function:

$$f_{EL}(\vec{x}) = \sum_{i=1}^{2} (\beta^{\frac{i-1}{2-1}} z_i)^2, \quad \begin{cases} x_1, x_2 \in [-3, 7], \\ \vec{z} = \mathbf{M}\vec{x}, \\ \beta = 5. \end{cases} \tag{5.1}$$

In order to transform this continuous function into a mixed-variable one, we discretize 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:

$$f_{EL_{MV}}(x_1, x_2) = z_1^2 + \beta \cdot z_2^2, \quad \begin{cases} x_1 \in \mathbf{T}, \\ x_2 \in [-3, 7], \\ \vec{z} = \mathbf{M}\vec{x}, \\ \beta = 5. \end{cases} \tag{5.2}$$

The set $\mathbf{T}$ is created by choosing t uniformly spaced values from the original domain $[-3, 7]$ so that $\exists_{i=1,...,t}\ \theta_i = 0$. In this way, it is always possible to find the optimum value $f_{EL_{MV}}(0, 0)^t = 0$, regardless of the chosen t discrete values.

Problems that involve ordinal variables are easy to simulate with the aforementioned procedure because the discrete points in the discretization for variable x_1 are naturally ordered. To simulate problems involving categorical variables only, the discrete points are randomly re-ordered. In this setting, a different ordering is generated for each run of the algorithm. This setting allows us to investigate how the algorithm performs when the ordering of the discrete points is not well defined or unknown.

The artificial mixed-variable benchmark functions have characteristics such as non-separability, ill-conditioning and multi-modality. Non-separable functions often exhibit complex dependencies between decision variables. Ill-conditioned functions often lead to premature convergence. Multi-modal functions have multiple local optima and require an efficient global search. Therefore, these characteristics are expected to be a challenge for different mixed-variable optimization algorithms. The flexibility in defining functions with different numbers of discrete points and the possible mixing of ordered and categorical variables enables systematic experimental studies addressing the impact of function features on algorithm performance.

5.2 $\text{ACO}_{\mathbf{MV}}$: ACO for mixed discrete-continuous optimization problems

In this section, we detail $\text{ACO}_{\mathbf{MV}}$, an ant colony optimization algorithm for tackling mixed discrete-continuous optimization problems. We start by describing the structure of $\text{ACO}_{\mathbf{MV}}$. Then, we describe the probabilistic solution construction for continuous variables, ordinal variables and categorical variables, respectively. In Section 5.2.1, we give analysis on some specific aspects of $\text{ACO}_{\mathbf{MV}}$.

$\text{ACO}_{\mathbf{MV}}$ structure

$\text{ACO}_{\mathbf{MV}}$ uses a *solution archive, SA,* as a form of pheromone model for the derivation of a probability distribution over the search space, following in this way the principle of population-based ACO [Guntsch and Middendorf, 2002] and also $\text{ACO}_{\mathbb{R}}$ [Socha and Dorigo, 2008]. The solution archive contains k complete solutions of 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.

Given an n-dimensional mixed discrete-continuous problem and k solutions, $\text{ACO}_{\mathbf{MV}}$ stores the value of the n variables and the objective function value of each solution in the solution archive. Fig. 5.1 shows the structure of the solution archive. It is divided into three groups of columns, one for continuous variables, one for ordinal variables, and one for categorical variables.

The basic flow of the $\text{ACO}_{\mathbf{MV}}$ algorithm and many of its particular choices follow $\text{ACO}_{\mathbb{R}}$. We describe the main details in what follows. The solution archive is initialized with k randomly generated solutions. Then, these k solutions are sorted according to their quality (from best to worst). A weight ω_j is associated with solution S_j. This weight is calculated using a Gaussian function defined by:

$$\omega_j = \frac{1}{qk\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2q^2k^2}}, \tag{5.3}$$

where $rank(j)$ is a function that returns the rank of solution S_j, and q is a parameter of the algorithm. By computing $rank(j)-1$, which corresponds to setting the mean of the Gaussian function to 1, the best solution receives the highest weight, while the weight of the other solutions decreases exponentially with their rank. At each iteration of the algorithm, m new solutions are probabilistically constructed by m ants, where an ant is a probabilistic solution construction procedure. The weight of a solution determines the level of attractiveness of that solution during

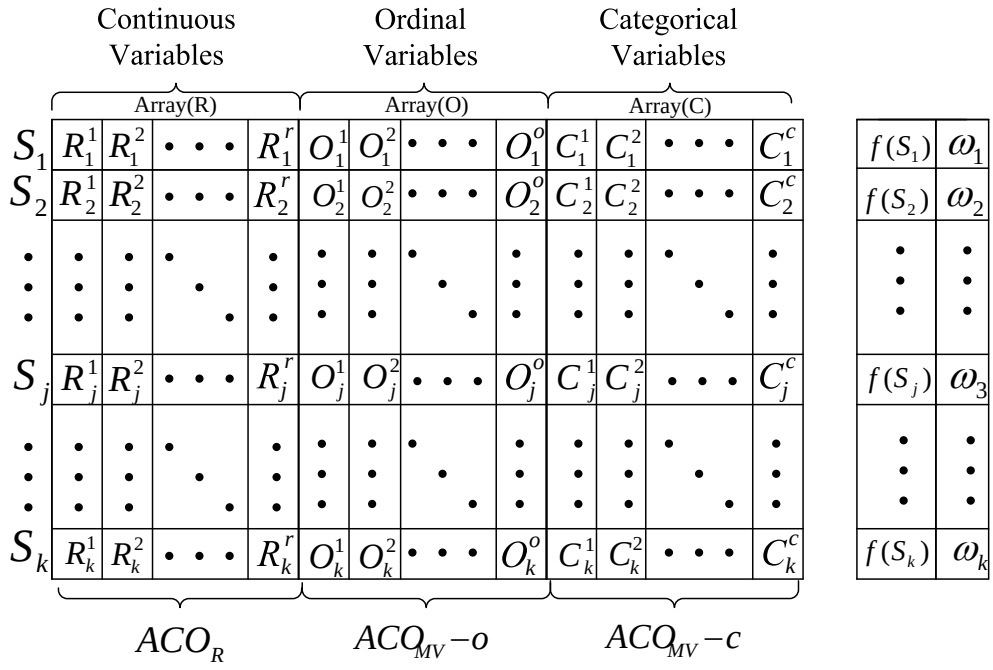


Figure 5.1: The structure of the solution archive used by ACO$_{\mathbf{MV}}$. The solutions in the archive are sorted according to their quality (i.e., the value of the objective function $f(S_j)$); hence, the position of a solution in the archive always corresponds to its rank.

the solution construction process. A higher weight means a higher probability of sampling around that solution. Once the m solutions have been generated, they are added into the solution archive. The $k + m$ solutions in the archive are then sorted and the m worst ones are removed. The remaining k solutions constitute the new solution archive. In this way, the search process is biased towards the best solutions found during the search. During the probabilistic solution construction process, an ant applies the construction mechanisms of ACO$_{\mathbb{R}}$, ACO$_{\mathbf{MV}}$-o and ACO$_{\mathbf{MV}}$-c. ACO$_{\mathbb{R}}$ handles continuous variables, while ACO$_{\mathbf{MV}}$-o and ACO$_{\mathbf{MV}}$-c handle ordinal variables and categorical variables, respectively. Their detailed description is given in the following subsection. An outline of the ACO$_{\mathbf{MV}}$ algorithm is given in Algorithm 5. The functions Sort and Best in Algorithm 5 implement the sorting of the archive and the selection of the k best solutions, respectively.

Probabilistic Solution Construction for Continuous Variables

Continuous variables are handled by ACO$_{\mathbb{R}}$ [Socha and Dorigo, 2008]. In ACO$_{\mathbb{R}}$, the construction of new solutions by the ants is accomplished in an incremental

Algorithm 5 Outline of $\text{ACO}_{\mathbf{MV}}$

```
Initialize decision variables
Initialize and evaluate k solutions
/* Sort solutions and store them in the archive SA */
SA ← Sort(S_1 ··· S_k)
while termination criterion is not satisfied do
  /* ConstructAntSolution */
  for 1 to m do
    Probabilistic Solution Construction for ACO_R
    Probabilistic Solution Construction for ACO_MV-o
    Probabilistic Solution Construction for ACO_MV-c
    Store and evaluate newly generated solutions
  end for
  /* Sort solutions and select the best k solutions */
  SA ← Best(Sort(S_1 ··· S_{k+m}), k)
end while
```

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:

$$p_j = \frac{\omega_j}{\sum_{l=1}^{k} \omega_l}, \tag{5.4}$$

where ω_j is calculated according to Equation (5.3).

An ant then constructs a new continuous variable solution around the chosen solution j. It assigns values to variables in a fixed variable order, that is, at the i-th construction step, $1 \leq i \leq r$, an ant assigns a value to continuous variable i. To assign a value to variable i, the ant samples the neighborhood around the value R_j^i of the chosen j-th solution. The sampling is done using a normal probability density function with mean μ and standard deviation σ:

$$g(x, \mu, \sigma) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{5.5}$$

When considering continuous variable i of solution j, we set $\mu = R_j^i$. Furthermore, we set

$$\sigma = \xi \sum_{l=1}^{k} \frac{|R_l^i - R_j^i|}{k-1}, \tag{5.6}$$

which is the average distance between the values of the i-th continuous variable of the solution j and the values of the i-th continuous variables of the other solutions in the archive, multiplied by a parameter ξ. This parameter has an effect similar to

that of the pheromone persistence in ACO. The higher the value of ξ, the lower the convergence speed of the algorithm. This process is repeated for each dimension by each of the m ants.

Thanks to the pheromone representation used in $\text{ACO}_{\mathbb{R}}$ (that is, the solution archive), it is possible to take into account the correlation between the decision variables. A non-deterministic adaptive method for doing so is presented in [Socha and Dorigo, 2008]. It is effective on the rotated benchmark functions proposed later (see Table 5.1) and it is also used to handle the variable dependencies of mixed discrete-continuous engineering problems in Section 5.5.

Probabilistic Solution Construction for Ordinal Variables

If the considered optimization problem includes ordinal variables, the continuous relaxation approach, $\text{ACO}_{\mathbf{MV}}$-o, is used. $\text{ACO}_{\mathbf{MV}}$-o does not operate on the actual values of the ordinal variables but on their indices in an array. The values of the indices for the new solutions are generated 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. The reason for this choice is that ordinal variables do not necessarily have numerical values; for example, an ordered variable may take as possible values $\{\text{small}, \text{medium}, \text{large}\}$. $\text{ACO}_{\mathbf{MV}}$-o otherwise works exactly as $\text{ACO}_{\mathbb{R}}$.

Probabilistic Solution Construction for Categorical Variables

While ordinal variables are relaxed and treated by the original $\text{ACO}_{\mathbb{R}}$, categorical variables are treated differently by $\text{ACO}_{\mathbf{MV}}$-c as this type of variables has no predefined ordering. At each step of $\text{ACO}_{\mathbf{MV}}$-c, an ant assigns a value to one variable at a time. For each categorical variable i, $1 \leq i \leq c$, an ant chooses probabilistically one of the t_i available values $v_l^i \in \{v_1^i, \ldots, v_{t_i}^i\}$. The probability of choosing the l-th value is given by

$$p_l^i = \frac{w_l}{\sum_{j=1}^{t_i} w_j}, \tag{5.7}$$

where w_l is the weight associated to the l-th available value. The weight w_l is calculated as

$$w_l = \begin{cases} \frac{\omega_{j_l}}{u_l^i} + \frac{q}{\eta}, & if (\eta > 0, u_l^i > 0), \\ \frac{\omega_{j_l}}{u_l^i}, & if (\eta = 0, u_l^i > 0), \\ \frac{q}{\eta}, & if (\eta > 0, u_l^i = 0), \end{cases} \tag{5.8}$$

where ω_{j_l} is calculated according to Equation (5.3) with j_l being the index of the highest quality solution that uses value v_l^i for the categorical variable i. u_l^i is the number of solutions that use value v_l^i for the categorical variable i in the archive (hence, the more common the value v_l^i is, the lower is its final weight); thus, u_l^i is a variable whose value is adapted at run-time and that controls the weight of choosing the l-th available value. $u_l^i = 0$ corresponds to the case in which the l-th available value is not used by the solutions in the archive; in this case the weight of the l-th value is equal to $\frac{q}{\eta}$. η is the number of values from the t_i available ones that are not used by the solutions in the archive; $\eta = 0$ (that is, all values are used) corresponds to the case in which $\frac{q}{\eta}$ is discarded. Again, η is a variable that is adapted at run-time and, if $\eta = 0$, it is natural to discard the second component in Equation (5.8). Note that u_l^i and η are nonnegative numbers, and their values are never equal to zero at the same time. q is the same parameter of the algorithm that was used in Equation (5.3). Here we note that Equation (5.8) here is a refined version of its corresponding equation given in the Socha's PhD thesis.

The weight w_l is therefore a sum of two components. The first component biases the choice towards values that are chosen in the best solutions but do not occur very frequently among all solutions in the archive. The second component plays the role of exploring values of the categorical decision variable i that are currently not used by any solution in the archive; in fact, the weight of such values according to the first component would be zero and, thus, this mechanism helps to avoid premature convergence (in other words, to increase diversification).

Restart strategy

$\text{ACO}_{\mathbf{MV}}$ uses a restart strategy for fighting stagnation. This strategy consists in restarting the algorithm without forgetting the best-so-far solution in the archive. A restart is triggered if the number of consecutive iterations with a relative solution improvement lower than a certain threshold ε is larger than *MaxStagIter*.

5.2.1 Algorithm analysis

Analysis of $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c

We first verify the relevance of the design choice we have taken in $\text{ACO}_{\mathbf{MV}}$, namely combining a continuous relaxation approach, $\text{ACO}_{\mathbf{MV}}$-o, and a native categorical optimization approach, $\text{ACO}_{\mathbf{MV}}$-c, in one single algorithm. We analyze the performance of $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c on two sets of the mixed discrete-continuous benchmark functions that were proposed in Section 5.1. The first set of benchmark functions involves continuous and ordinal variables. The second set of benchmark functions involves continuous and categorical variables.

For the two settings described in Section 5.1, we evaluate the performance of $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c on six benchmark functions with different numbers t of discrete points in the discretization, $t \in \{2, 5, 10, 20, 30, ..., 90, 100, 200, 300, ..., 900, 1\,000\}$, and dimensions 2, 6 and 10; this results in 18 groups of experiments (six benchmark functions and three dimensions) for the first and the second set of benchmark functions. In this study, half of the dimensions are continuous variables and the other half are discrete variables. The continuous variables in these benchmark functions are handled by $\text{ACO}_{\mathbb{R}}$, while the discrete variables are handled by $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c, respectively.

To ensure a fair comparison in every group of experiments, we tuned the parameters of $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c using Iterated F-Race [Balaprakash et al., 2007, Birattari et al., 2010] with the same tuning budget on a training set of benchmark functions. The performance measure for tuning is the objective function value of each instance after 10 000 function evaluations. The tuning budget for Iterated F-Race is set to 2 000 runs of $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c. The training set involves ordinal and categorical variables with a random number of t discrete points, $t \in \{2, 5, 10, 20, 30, ..., 90, 100, 200, 300, ..., 900, 1\,000\}$. In a test phase, we conducted experiments with benchmark functions different from those used in the training phase. The comparisons for each possible number t of discrete points were performed independently in each experiment group (defined by benchmark function and dimension). In total, we conducted $378 = 21 \times 6 \times 3$ comparisons for ordinal and categorical variables, respectively. In each experiment, we compare $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c without restart mechanism by measuring the solution quality obtained by 50 independent runs. A uniform random search (URS) method [Brooks, 1958] is included as a baseline for comparison. It consists in sampling search points uniformly at random in the search domain and keeping the best solution found.

Table 5.2: Comparison between $\text{ACO}_{\mathbf{MV}}$-o, $\text{ACO}_{\mathbf{MV}}$-c and uniform random search (URS) for two setups of discrete variables. For each comparison, we give the frequency with which the first mentioned algorithm is statistically significantly better, indistinguishable, or worse than the second one.

	1st setup Ordinal variables	2nd setup Categorical variables
$\text{ACO}_{\mathbf{MV}}$-o vs. $\text{ACO}_{\mathbf{MV}}$-c	0.63, 0.35, 0.02	0.07, 0.00, 0.93
$\text{ACO}_{\mathbf{MV}}$-o vs. URS	0.98, 0.02, 0.00	0.78, 0.12, 0.10
$\text{ACO}_{\mathbf{MV}}$-c vs. URS	0.93, 0.07, 0.00	0.96, 0.04, 0.00

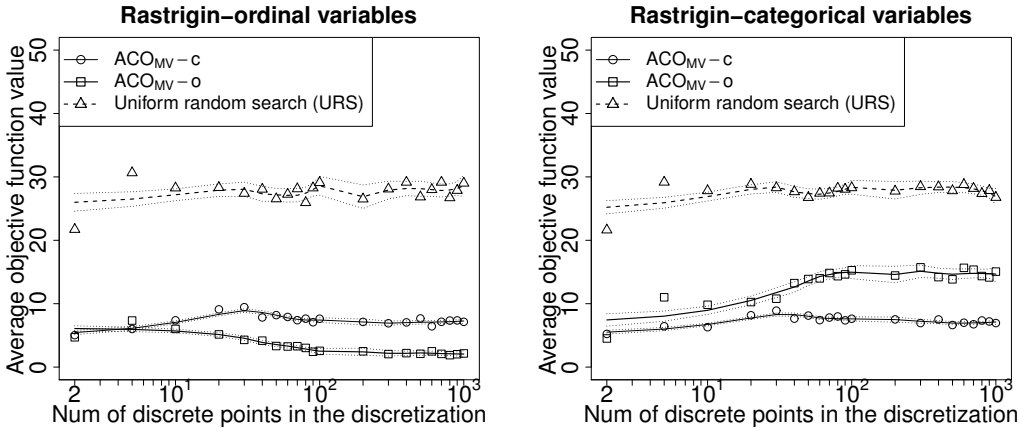


Figure 5.2: The plot shows the average objective function values obtained by $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c on the 6 dimensional function $f_{Rastrigin_{MV}}$ after 10 000 evaluations, with the number t of discrete points in the discretization $t \in \{2, 5, 10, 20, 30, ..., 90, 100, 200, 300, ..., 900, 1\,000\}$.

Table 5.2 summarizes the results of the comparison between $\text{ACO}_{\mathbf{MV}}$-o, $\text{ACO}_{\mathbf{MV}}$-c and URS for ordinal and categorical variables. The Wilcoxon rank-sum test at the 0.05 α-level is used to test the statistical significance of the differences in each of the 378 comparisons. In the case of ordinal variables, the statistical analysis revealed that in 63% of the 378 comparisons $\text{ACO}_{\mathbf{MV}}$-o reaches statistically significantly better solutions than $\text{ACO}_{\mathbf{MV}}$-c, in 2% of the experiments $\text{ACO}_{\mathbf{MV}}$-c is statistically significantly better than $\text{ACO}_{\mathbf{MV}}$-o, and in the remaining 35% of the cases there was no statistically significant difference. As expected, both $\text{ACO}_{\mathbf{MV}}$-o and $\text{ACO}_{\mathbf{MV}}$-c outperform URS: they perform significantly better in 98% and 93% of the cases, respectively, and they never obtain statistically significantly worse results than URS. In the case of categorical variables, the statistical analysis revealed that in 93% of the 378 comparisons $\text{ACO}_{\mathbf{MV}}$-c reaches

statistically significantly better solutions than ACO$_{\mathbf{MV}}$-o and in 7% of the experiments ACO$_{\mathbf{MV}}$-o is statistically significantly better than ACO$_{\mathbf{MV}}$-c. Again, both ACO$_{\mathbf{MV}}$-o and ACO$_{\mathbf{MV}}$-c outperform URS. They perform better in 96% and 78% of the cases, respectively, and ACO$_{\mathbf{MV}}$-c never obtains statistically significantly worse results than URS.

These experiments confirm our expectation that ACO$_{\mathbf{MV}}$-o is more effective than ACO$_{\mathbf{MV}}$-c on problems with ordinal variables, while ACO$_{\mathbf{MV}}$-c is more effective than ACO$_{\mathbf{MV}}$-o on problems with categorical variables. In Fig. 5.2, the comparisons on $f_{Rastrigin_{MV}}$ are shown. As seen in the figure, the categorical optimization approach, ACO$_{\mathbf{MV}}$-c, reaches approximately the same objective function values no matter whether the discrete variables are ordinal or categorical. The continuous relaxation approach ACO$_{\mathbf{MV}}$-o performs better than ACO$_{\mathbf{MV}}$-c in the case of ordinal variables, but its performance is not as good when applied to the categorical case.

Effectiveness of the restart mechanism

Here we show that ACO$_{\mathbf{MV}}$'s restart mechanism really helps in improving its performance. We conducted 50 independent runs using a maximum of 1 000 000 evaluations in each run. In Fig. 5.3, we show ACO$_{\mathbf{MV}}$'s run-length distributions (RLDs, for short) on two multi-modal functions $f_{Ackley_{MV}}$ and $f_{Griewank_{MV}}$ with continuous and categorical variables with $t = 100$ discrete points. An empirical RLD gives the estimated cumulative probability distribution for finding a solution of a certain quality as a function of the number of objective function evaluations. (For more information about RLDs, we refer the reader to [Hoos and Stützle, 2005].) As expected, ACO$_{\mathbf{MV}}$'s performance is strongly improved by the restart mechanism. For example, in the case of $f_{Ackley_{MV}}$ in two, six and ten dimensions, ACO$_{\mathbf{MV}}$ reaches a solution whose objective function value is equal to or less than 1.00E−10 with probability 1 or 100% success rate, and in the case of $f_{Griewank_{MV}}$ in two, six and ten dimensions ACO$_{\mathbf{MV}}$ reaches a solution whose objective function value is equal to or less than 1.00E−10 with probability 1, 0.82 and 0.85 respectively. Without restart, ACO$_{\mathbf{MV}}$ stagnates at much lower success rates.

Analysis of Equation (5.8)

We experimentally explored different options for the shape of Equation (5.8) to illustrate the influence of alternative choices for Equation. We perform two experiments on two multi-modal functions $f_{Ackley_{MV}}$ and $f_{Griewank_{MV}}$ with continuous and

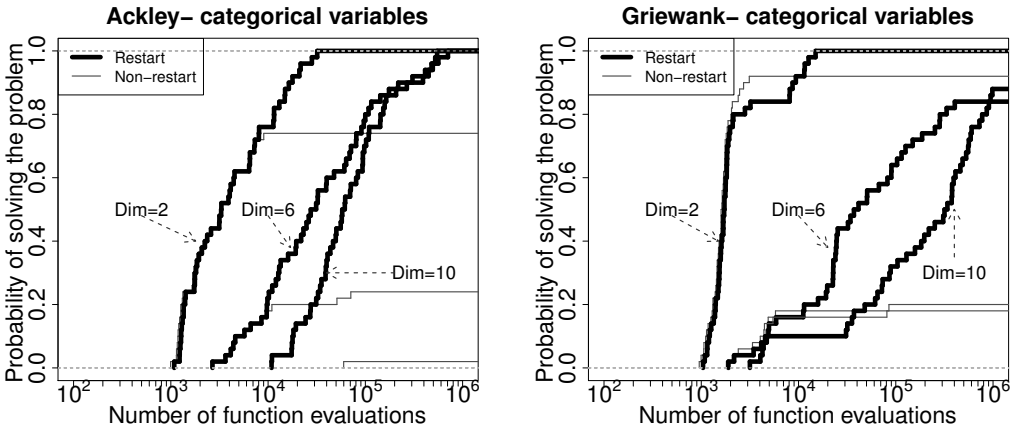


Figure 5.3: The RLDs obtained by $\text{ACO}_{\mathbf{MV}}$ with restarts and without restarts. The solution quality threshold is 1.00E−10. D indicates the dimensionality of the benchmark problem. Half of the dimensions are categorical variables and the other half are continuous variables.

categorical variables with $t = 100$ discrete points. The details of the benchmark functions and the experimental setup is explained in Sections 5.1 and 5.4.1. The two experiments are based on the following alternative choices for Equation (5.8).
(1) We modify Equation (5.8) to

$$w_l = \begin{cases} \omega_{j_l}, & if(u_l^i > 0), \\ 0, & if(u_l^i = 0). \end{cases} \tag{5.9}$$

That is, we omit the terms u_l^i and $\frac{q}{\eta}$ in Equation (5.8).
(2) We modify Equation (5.8) to

$$w_l = \begin{cases} \frac{\omega_{j_l}}{u_l^i}, & if(u_l^i > 0), \\ 0, & if(u_l^i = 0). \end{cases} \tag{5.10}$$

That is, we omit the term $\frac{q}{\eta}$ in Equation (5.8).

We tuned the parameters of two versions of $\text{ACO}_{\mathbf{MV}}$ that use the two alternative Equations (5.9) and (5.10), respectively, by the same automatic tuning procedure used for tuning the original $\text{ACO}_{\mathbf{MV}}$ with Equation (5.8) to ensure a fair comparison.

Summary information based on RLDs are given in Fig. 5.4 and 5.5. The results of experiment (1) show that the RLDs obtained by using Equation (5.8) clearly dominate those obtained by using Equation (5.9). In fact, the success rates ob-

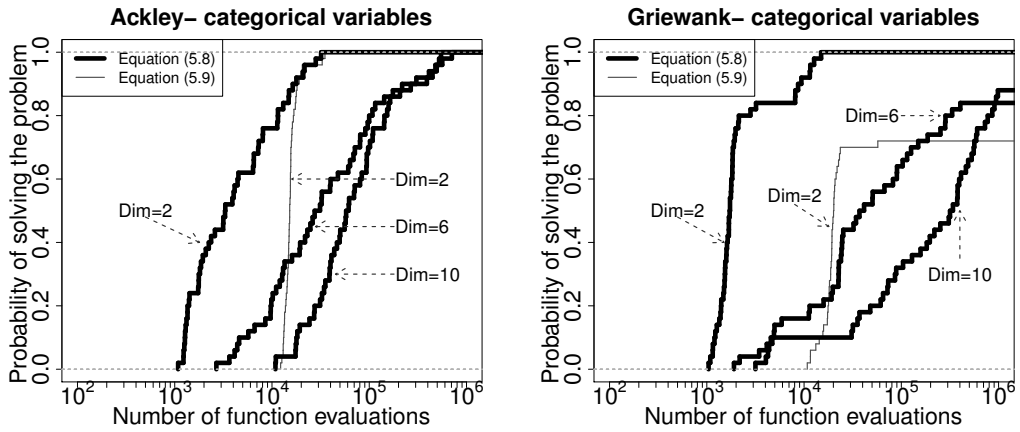


Figure 5.4: The RLDs obtained by the two $\text{ACO}_{\mathbf{MV}}$ variants with Equation (5.8) and (5.9) in 50 independent runs. The solution quality threshold is 1.00E−10. D indicates the dimensionality of the benchmark problem. Half of the dimensions are categorical variables and the other half are continuous variables.

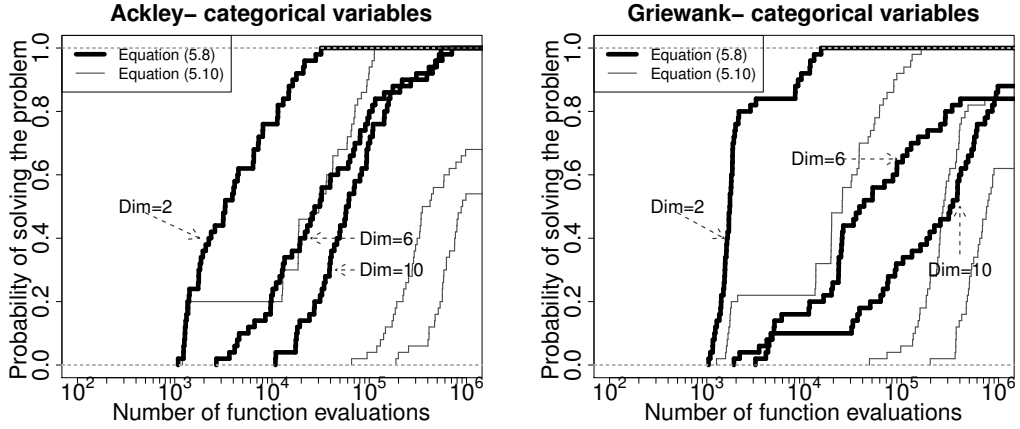


Figure 5.5: The RLDs obtained by the two $\text{ACO}_{\mathbf{MV}}$ variants with Equation (5.8) and (5.10) in 50 independent runs. The solution quality threshold is 1.00E−10. D indicates the dimensionality of the benchmark problem. Half of the dimensions are categorical variables and the other half are continuous variables. The RLDs obtained by $\text{ACO}_{\mathbf{MV}}$ with Equation (5.10) in dimensions two, six and ten are sequentially shown as the increasing number of function evaluations to solve the problem at the first time.

tained by Equation (5.9) in dimensions six and ten are zero and therefore not shown in Figure 5.4. The same conclusions hold for experiment (2): the RLDs obtained by using Equation (5.8) dominate those obtained by using Equation (5.10) in all cases, illustrating in this way the benefit of Equation (5.8). Finally we approve the shape Equation (5.8) through fair comparisons.

5.3 CMA-ES extensions for mixed discrete-continuous optimization

In this section, we present three CMA-ES extensions for mixed discrete-continuous optimization. As for $\text{ACO}_{\mathbf{MV}}$, the three CMA-ES extensions allow the user to explicitly declare each variable of a mixed-variable optimization problem as continuous, ordinal or categorical. They feature different approaches for handling categorical variables and use the same approach for continuous and ordinal variables. We start by describing CES-RoundC, a basic CMA-ES extension for handling mixed discrete-continuous optimization. There we introduce the approaches for handling continuous and ordinal variables. Then we present $\text{CES}_{\mathbf{MV}}$ and CES-RelayC focusing on the different approaches for handling categorical variables.

5.3.1 CES-RoundC

In CES-RoundC, continuous variables are handled with a continuous optimization approach (CMA-ES), and both, ordinal and categorical, variables are handled with a continuous relaxation approach through rounding. We label the continuous relaxation approach as Round. Round is the same mechanism as $\text{ACO}_{\mathbf{MV}}$ for ordinal variables on the page 97. 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. Round otherwise works exactly as CMA-ES. When the considered optimization problem includes ordinal or categorical variables, Round is used.

The CES-RoundC algorithm is described as follows. CES-RoundC initializes a population with $4 + \lfloor a \ln(D) \rfloor$ randomly generated solutions. Then, CES-RoundC applies the mechanisms of CMA-ES and Round to iteratively generate new populations. The number of search points selected for the parent population is $\mu = \lfloor \lambda / b \rfloor$. The initial step-size is $\sigma^{(0)} = c(B - A)$. CES-RoundC uses the same restart schema as in [Auger and Hansen, 2005]. At each restart, CES-RoundC increases the population size by a factor d and randomly generates a new population of solutions.

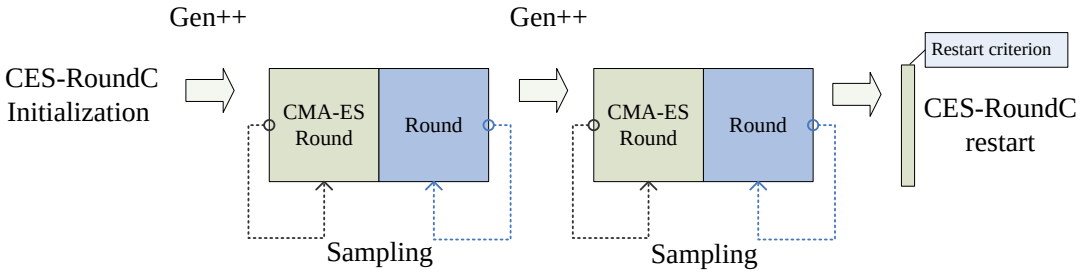


Figure 5.6: A schema of the CES-RoundC algorithm flow. The blue squares denote using Round to tackle categorical variables; The gray squares denote using CMA-ES and Round to tackle continuous and ordinal variables. The restart criterion rectangle denotes the triggering of the restart in CES-RoundC. The number of population generations is labeled as Gen. The initial value of Gen is equal to one. After each generation, $Gen++$.

Restarts are triggered using the three parameters $stopTolFunHist(=10^{-f})$, $stopTolFun(=10^{-e})$ and $stopTolX(=10^{-g})$; they refer to the improvement of the best objective function value in the last $10+\lceil 30D/\lambda \rceil$ generations, the function values of the recent generation, and the standard deviation of the normal distribution in all coordinates, respectively. The parameter settings of CES-RoundC are the same as those for CMA-ES. Note that a, b, c, d, e, f, g are the exposed parameters that are used for automatic tuning later. A schema of the CES-RoundC algorithm flow is shown in Fig 5.6.

5.3.2 CES_{MV}

CES_{MV} and CES-RoundC differ from the approach to handle categorical variables. In CES_{MV}, continuous variables are handled with a continuous optimization approach (CMA-ES), ordinal variables are handled with Round, and categorical variables are handled with a categorical optimization approach (CES_{MV}-c) in each generation. A schema of the CES_{MV} algorithm flow is shown in Fig 5.7. We focus on the description of the CES_{MV}-c approach in the following.

CES_{MV}-c is analogous to ACO_{MV}-c. CES_{MV}-c is a population based categorical optimization approach. It is independent from the approaches for continuous and ordinal variables. It uses informations from the population of categorical variables and from the solution fitness values. In CES_{MV}, the population size for CES_{MV}-c is the same as with CMA-ES. The initial population size is $4+\lfloor a \ln(D) \rfloor$. At each restart, the population size for CES_{MV}-c is increased by a factor d as in

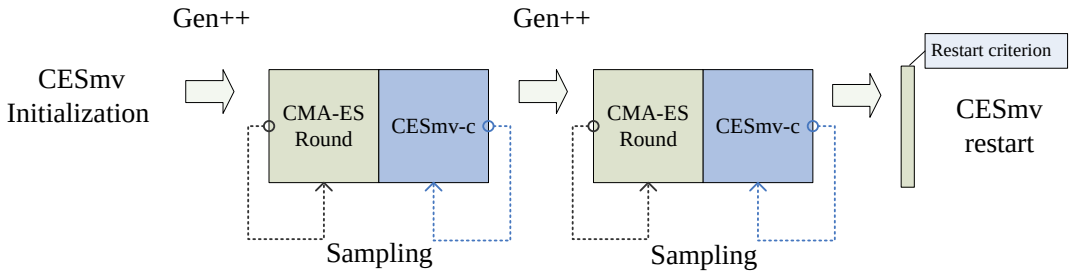


Figure 5.7: A schema of the CES$_{\mathbf{MV}}$ algorithm flow. The blue squares denote using CES$_{\mathbf{MV}}$-c to tackle categorical variables; the gray squares denote using CMA-ES and Round to tackle continuous and ordinal variables. The restart criterion rectangle denotes the triggering of the restart in CES$_{\mathbf{MV}}$. The number of generations is labeled as *Gen*. The initial value of *Gen* is equal to one. After each generation, $Gen++$.

CMA-ES. In CES$_{\mathbf{MV}}$-c, for each categorical variable i, $1 \leq i \leq c$, each solution entry in the population chooses probabilistically one of the t_i available values $v_l^i \in \{v_1^i, \ldots, v_{t_i}^i\}$. The probability of choosing the l-th value is given by

$$p_l^i = \frac{w_l}{\sum_{j=1}^{t_i} w_j}, \tag{5.11}$$

where w_l is the weight associated to the l-th available value. The weight w_l is calculated as

$$w_l = \begin{cases} \dfrac{\omega_{j_l}}{u_l^i} + \dfrac{1}{Gen \times \eta}, & if(\eta > 0, u_l^i > 0), \\ \dfrac{\omega_{j_l}}{u_l^i}, & if(\eta = 0, u_l^i > 0), \\ \dfrac{1}{Gen \times \eta}, & if(\eta > 0, u_l^i = 0), \end{cases} \tag{5.12}$$

A weight ω_j is associated with solution S_j. This weight is calculated using a Gaussian function defined by:

$$\omega_j = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(rank(j)-1)^2}{2\sigma^2}}, \tag{5.13}$$

where $\sigma = max(1/Gen, 0.5)$, and $rank(j)$ is a function that returns the rank of solution S_j. ω_{j_l} is calculated according to Equation (5.13) with j_l being the index of the highest quality solution that uses value v_l^i for the categorical variable i.

CES$_{\mathbf{MV}}$-c and ACO$_{\mathbf{MV}}$-c have three main differences. These differences are that (i) in each generation of ACO$_{\mathbf{MV}}$, the solution entries are chosen by a certain

number of ants to sample values, while in each generation of $\text{CES}_{\mathbf{MV}}$, each solution entry in the population is used. (ii) $\text{ACO}_{\mathbf{MV}}$-c uses parameter q in Equation 5.8, while $\text{CES}_{\mathbf{MV}}$-c use an adaptive variable $1/Gen$ to replace with q, resulting the new Equation 5.12; (iii) The computation of σ in Equation 5.13 does not follow the way of Equation 5.3. σ in Equation 5.13 is simply defined as $\sigma = max(1/Gen, 0.5)$. In $\text{CES}_{\mathbf{MV}}$, σ value is therefore kept equal to 0.5 from the second generation on. This lower bound 0.5 is used to keep $\text{CES}_{\mathbf{MV}}$ a sufficient deviation to search large mixed-variable domains.

5.3.3 CES-RelayC

The high performance of CMA-ES in continuous optimization has attracted researchers to extend it to tackle mixed discrete-continuous problems by using a two-partition strategy [Sambo et al., 2012, Wagner et al., 2011]. The two-partition strategy for mixed discrete-continuous problems consists in partitioning the variables into continuous and discrete variables, the variables of one partition are optimized separately for fixed values of the variables of the other partition. For instance, in [Wagner et al., 2011], a basic two-partition strategy was successfully applied for track cycling problem with mixed discrete-continuous variables. CMA-ES was first used to optimize continuous variables for fixed values of the discrete variables; then, a simple discrete optimization method (e.g. random search) was used to optimize discrete variables for setting the continuous variables to the best found setting in the previous phase.

In this section, we propose CES-RelayC, a relay version following the two-partition strategy where variables of one partition (containing the categorical ones) are optimized for fixed values of the variables of the other partition (containing the continuous and ordinal ones). In CES-RelayC, $\text{CES}_{\mathbf{MV}}$-c handles categorical variables in one partition and IPOP-CMA-ES and Round handle continuous and ordinal variables in another partition.

We describe CES-RelayC by focusing on how it handles continuous and categorical variables. The population is initialized with $Cpop$ randomly generated solutions. $\text{CES}_{\mathbf{MV}}$-c is then applied to update the values of the categorical variables of the $Cpop$ solutions. Next, the best ranked solution of this generation, S_{C-Gen}, is recorded, and the values of the continuous variables of S_{C-Gen} are used as the initial solution for IPOP-CMA-ES approach. The values of the categorical variables in S_{C-Gen} remain fixed, while IPOP-CMA-ES is run for a maximum of R_{eval} function evaluations. Next, the best-so-far solution after running IPOP-

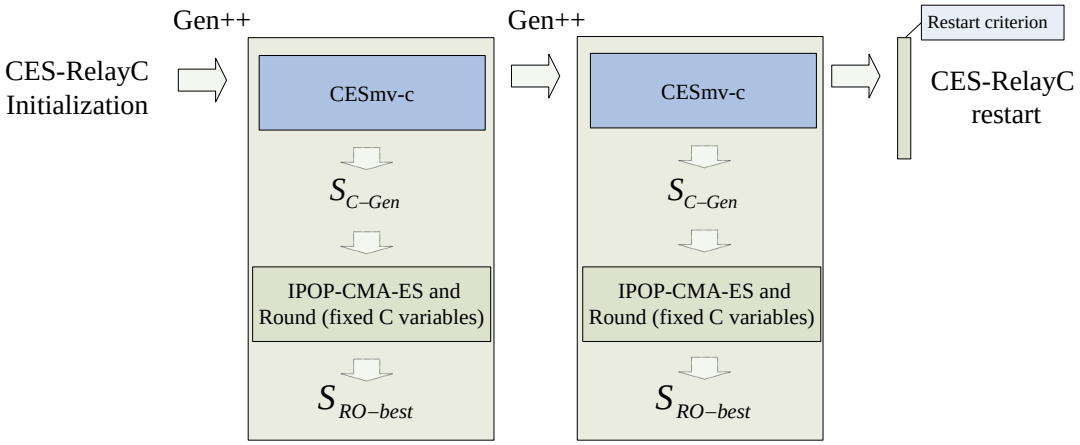


Figure 5.8: A schema of the CES-RelayC algorithm flow. After initialization, CES$_{\mathbf{MV}}$-c is first applied to update the values of the categorical variables. Next, the best ranked solution of this phase, S_{C-Gen}, is recorded, and the values of the continuous (and ordinal) variables of S_{C-Gen} are used as initial solution for the IPOP-CMA-ES (and Round) phase. Using the values of the categorical variables fixed to those in S_{C-Gen}, IPOP-CMA-ES (and Round) is then applied, resulting $S_{RO-best}$. In the next generation, CES$_{\mathbf{MV}}$-c updates the values of the categorical variables with those of continuous (and ordinal) variables fixed to those of in $S_{RO-best}$. The process continues until the stopping condition CES-RelayC is satisfied. CES-RelayC uses the same restart mechanism as ACO$_{\mathbf{MV}}$ for fighting stagnation.

CMA-ES is recorded in $S_{RO-best}$. With the values of the continuous variables fixed to those of $S_{RO-best}$, CES$_{\mathbf{MV}}$-c generates new values of the categorical variables for the next generation. The process continues until the stopping condition CES-RelayC is satisfied. CES-RelayC uses the same restart mechanism as ACO$_{\mathbf{MV}}$ for fighting stagnation. A schema of the CES-RelayC algorithm flow is shown in Fig 5.8.

5.4 Automatic tuning and performance evaluation

5.4.1 Automatic tuning

Besides serving for experimental studies, the new benchmark functions proposed in Section 5.1 can be used to generate a training set of problems for the automatic parameter tuning of mixed-variable optimization algorithms. The tuning of an

Table 5.3: Parameter settings for $\text{ACO}_{\mathbf{MV}}$ tuned by Iterated F-race

Parameter	Symbol	Value
Number of ants	m	5
Influence of best quality solutions	q	0.05099
Width of the search	ξ	0.6795
Archive size	k	90
Stagnating iterations before restart	*MaxStagIter*	650
Relative improvement threshold	ε	10^{-5}

Table 5.4: Parameter settings for CES-RoundC, $\text{CES}_{\mathbf{MV}}$ and CES-RelayC

Parameter	Internal parameter		CES-RoundC	$\text{CES}_{\mathbf{MV}}$	CES-RelayC
a	Init pop size: λ_0	$= 4 + \lfloor a \ln(D) \rfloor$	5.679	3.158	5.979
b	Parent size: μ	$= \lfloor \lambda / b \rfloor$	2.216	1.804	1.615
c	Init step size: σ_0	$= c \cdot (B - A)$	0.23690	0.1597	0.28280
d	IPOP factor: *ipop*	$= d$	1.728	1.913	2.722
e	*stopTolFun*	$= 10^e$	-14.62	-10.21	-13.28
f	*stopTolFunHist*	$= 10^f$	-10.14	-12.51	-14.58
g	*stopTolX*	$= 10^g$	-10.21	-11.9	-11.66
Cpop	Pop size for $\text{CES}_{\mathbf{MV}}$-c	$=$ *Cpop*	-	-	806
R_{eval}	Evaluations for IPOP-CMA-ES	$= R_{eval}$	-	-	6078
MaxStagIter	Stagnating iterations before restart	$=$ *MaxStagIter*	-	-	20
ε	Relative improvement threshold	$= \varepsilon$	-	-	10^{-5}

algorithm on a training set that is different from the test set is important to allow for an unbiased assessment of the algorithm's performance on (by the algorithm unseen) test problems [Birattari, 2009]. We therefore generate a training set of benchmark functions across all six mixed-variable benchmark functions, across various dimensions [Liao et al., 2011c] (taken from the set $n \in \{2, 4, 6, 8, 10, 12, 14\}$), and across various ratios of ordinal and categorical variables. As in the other tuning tasks in this thesis, we use Iterated F-Race [Balaprakash et al., 2007, Birattari et al., 2010] that is included in the `irace` [López-Ibáñez et al., 2011]. The performance measure for tuning is the objective function value of each problem instance after 10 000 function evaluations. The tuning budget for Iterated F-Race is set to 5 000 runs of each algorithm. We use the default settings of Iterated F-Race [Birattari et al., 2010, López-Ibáñez et al., 2011]. The obtained parameter settings after tuning $\text{ACO}_{\mathbf{MV}}$, CES-RoundC, $\text{CES}_{\mathbf{MV}}$, and CES-RelayC are given in Tables 5.3 and 5.4. Next, we use these parameter settings for a validation of performance.

5.4.2 Performance evaluation on benchmark functions

First we examine the performance of the three CMA-ES extensions, CES-RoundC, $\text{CES}_{\mathbf{MV}}$ and CES-RelayC on artificial mixed-variable benchmark functions ($f_{Sphere_{MV}}$, $f_{Ellipsoid_{MV}}$, $f_{Rosenbrock_{MV}}$, $f_{Ackley_{MV}}$, $f_{Rastrigin_{MV}}$ and $f_{Griewank_{MV}}$)

with continuous and categorical variables and with dimensions four, ten and twenty. Half of the dimensions are categorical variables and the other half are continuous variables. The experimental results are measured across 50 independent runs of 1 000 000 objective function evaluations for instances with $t = 10$ discrete points. In Fig. 5.9 we show their run-time behavior by using run-length distributions (RLDs, for short) [Hoos and Stützle, 2005]. An (empirical) RLD provides a graphical view of the development of the empirical frequency of finding a solution of a certain quality as a function of the number of objective function evaluations. Except the 10 and 20 dimensional versions of $f_{Ellipsoid_{MV}}$ and $f_{Rastrigin_{MV}}$, the solution quality required is set to 1.00E−10, which is used as zero threshold. The solution quality required for the 10 and 20 dimensional $f_{Ellipsoid_{MV}}$ and $f_{Rastrigin_{MV}}$ functions is enlarged due to the inherent difficulty of these problems, the required values are shown in the title of the corresponding RLDs plot. Success rate is the proportion of successful runs in the total number of runs. A successful run is a run during which the algorithm finds the required solution quality. Convergence speed refers to the number of function evaluations used to reach 100% success rate or a specific success rate required in some cases. Faster (slower) convergence speed corresponds to fewer (more) function evaluations needed.

$\mathbf{CES_{MV}}$ and CES-RoundC

$\mathrm{CES}_{\mathbf{MV}}$ performs clearly better than CES-RoundC on both success rates and convergence speed. As seen from the simplest $f_{Sphere_{MV}}$, in four, ten and twenty dimensions $\mathrm{CES}_{\mathbf{MV}}$ reaches a solution whose objective function value is equal to or less than 1.00E−10 with probability 1; CES-RoundC reaches a solution whose objective function value is equal to or less than 1.00E−10 with probability 1, 0.64 and about 0.44 respectively. Comparing to $\mathrm{CES}_{\mathbf{MV}}$, CES-RoundC stagnates at much lower success rates in higher dimensions, and converges much more slowly.

$\mathbf{CES_{MV}}$ and CES-RelayC

The success rates reached by CES-RelayC are never lower than 60%. CES-RelayC reaches a 100% success rate in $f_{Sphere_{MV}}$, $f_{Ackley_{MV}}$ in all dimensions; in dimension four, CES-RelayC reaches a 100% success rate for all functions except for $f_{Ellipsoid_{MV}}$ (96%), $f_{Rastrigin_{MV}}$ (98%) and $f_{Griewank_{MV}}$ (98%). Only in some cases such as the ten and twenty dimensional $f_{Rosenbrock_{MV}}$ function, CES-RelayC stagnates at a success rates lower than 100%. CES-RelayC clearly improves over CES-RoundC. However, CES-RelayC performs worse than $\mathrm{CES}_{\mathbf{MV}}$, especially concern-

Table 5.5: The number of function evaluations used by CES-RelayC and $\text{CES}_{\mathbf{MV}}$ for obtaining a 100% success rate in $f_{Sphere_{MV}}$, $f_{Ackley_{MV}}$ of each dimension.

Function	Dimension	CES-RelayC	$\text{CES}_{\mathbf{MV}}$
$f_{Sphere_{MV}}$	4	14 628	2 657
	10	100 356	7 451
	20	167 752	15 386
$f_{Ackley_{MV}}$	4	17 864	5 536
	10	112 574	15 257
	20	308 256	37 120

ing convergence speed. CES-RelayC uses much larger number of function evaluations to find the solution quality required than $\text{CES}_{\mathbf{MV}}$ does. This conclusion is clearly illustrated by their performance on $f_{Sphere_{MV}}$ and $f_{Ackley_{MV}}$. Table 5.5 shows the number of function evaluations used by CES-RelayC and $\text{CES}_{\mathbf{MV}}$ for obtaining a 100% success rate in $f_{Sphere_{MV}}$, $f_{Ackley_{MV}}$ of each dimension. Despite the relay and restart mechanisms in CES-RelayC facilitates the high success rates, this two partition mechanism converges slower than the $\text{CES}_{\mathbf{MV}}$ algorithm can not be avoided. In summary, we identify $\text{CES}_{\mathbf{MV}}$ as the best performing variant among the three CMA-ES extensions.

$\text{CES}_{\mathbf{MV}}$ and $\text{ACO}_{\mathbf{MV}}$

Next, we compare the performance of $\text{CES}_{\mathbf{MV}}$ and $\text{ACO}_{\mathbf{MV}}$. As seen from Fig. 5.9, either $\text{ACO}_{\mathbf{MV}}$ or $\text{CES}_{\mathbf{MV}}$ obtains the best performance on most of the functions. With respect to success rates, $\text{CES}_{\mathbf{MV}}$ performs better than $\text{ACO}_{\mathbf{MV}}$. Over dimension four, ten and twenty, $\text{CES}_{\mathbf{MV}}$ obtains 100% success rate when applied to solve almost all the functions, while $\text{ACO}_{\mathbf{MV}}$ stagnates at lower success rates in some cases such as $f_{Rosenbrock_{MV}}$ and $f_{Griewank_{MV}}$ functions. For $f_{Rosenbrock_{MV}}$ and $f_{Griewank_{MV}}$, $\text{CES}_{\mathbf{MV}}$ clearly obtains larger success rates than $\text{ACO}_{\mathbf{MV}}$. With respect to convergence speed, in some functions such as $f_{Sphere_{MV}}$, $f_{Ackley_{MV}}$, $\text{ACO}_{\mathbf{MV}}$ reach 100% success rate faster than $\text{CES}_{\mathbf{MV}}$.

We also evaluate the performance of $\text{CES}_{\mathbf{MV}}$ and $\text{ACO}_{\mathbf{MV}}$ on the artificial mixed-variable benchmark functions with continuous and ordinal variables. The experimental setups are the same as those with categorical variables. As seen from Fig. 5.10, $\text{CES}_{\mathbf{MV}}$ and $\text{ACO}_{\mathbf{MV}}$ reach very similar performance in the benchmark function with four dimension. In fact, $\text{ACO}_{\mathbf{MV}}$ even more often reaches 100% success rate faster than $\text{CES}_{\mathbf{MV}}$. However, in dimensions ten and twenty, $\text{CES}_{\mathbf{MV}}$ clearly performs better than $\text{ACO}_{\mathbf{MV}}$ in most cases. Most striking are the example of the 20 dimensional $f_{Rastrigin_{MV}}$ and $f_{Griewank_{MV}}$ functions, for which $\text{ACO}_{\mathbf{MV}}$ can not find an optimum solution while $\text{CES}_{\mathbf{MV}}$ reaches very high success rates.

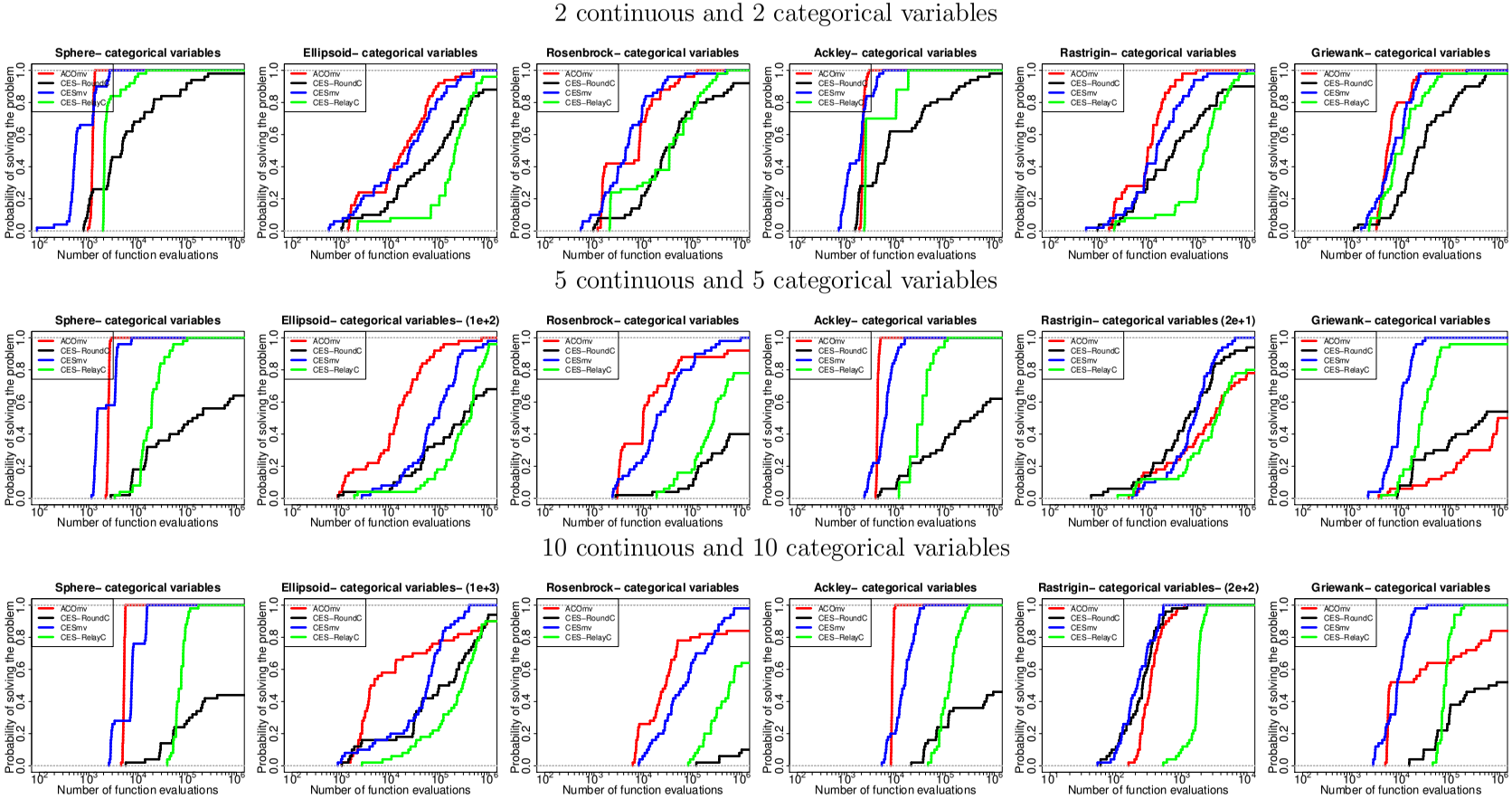


Figure 5.9: The qualified run-length distributions (RLDs, for short) over 50 independent runs obtained by $\text{ACO}_{\mathbf{MV}}$, CES-RoundC, $\text{CES}_{\mathbf{MV}}$ and CES-RelayC. The solution quality required is 1.00E−10 (zero threshold) except for the 10 and 20 dimensional $f_{Ellipsoid_{MV}}$ and $f_{Rastrigin_{MV}}$, for which the solution quality required is shown in the title of the corresponding plot.

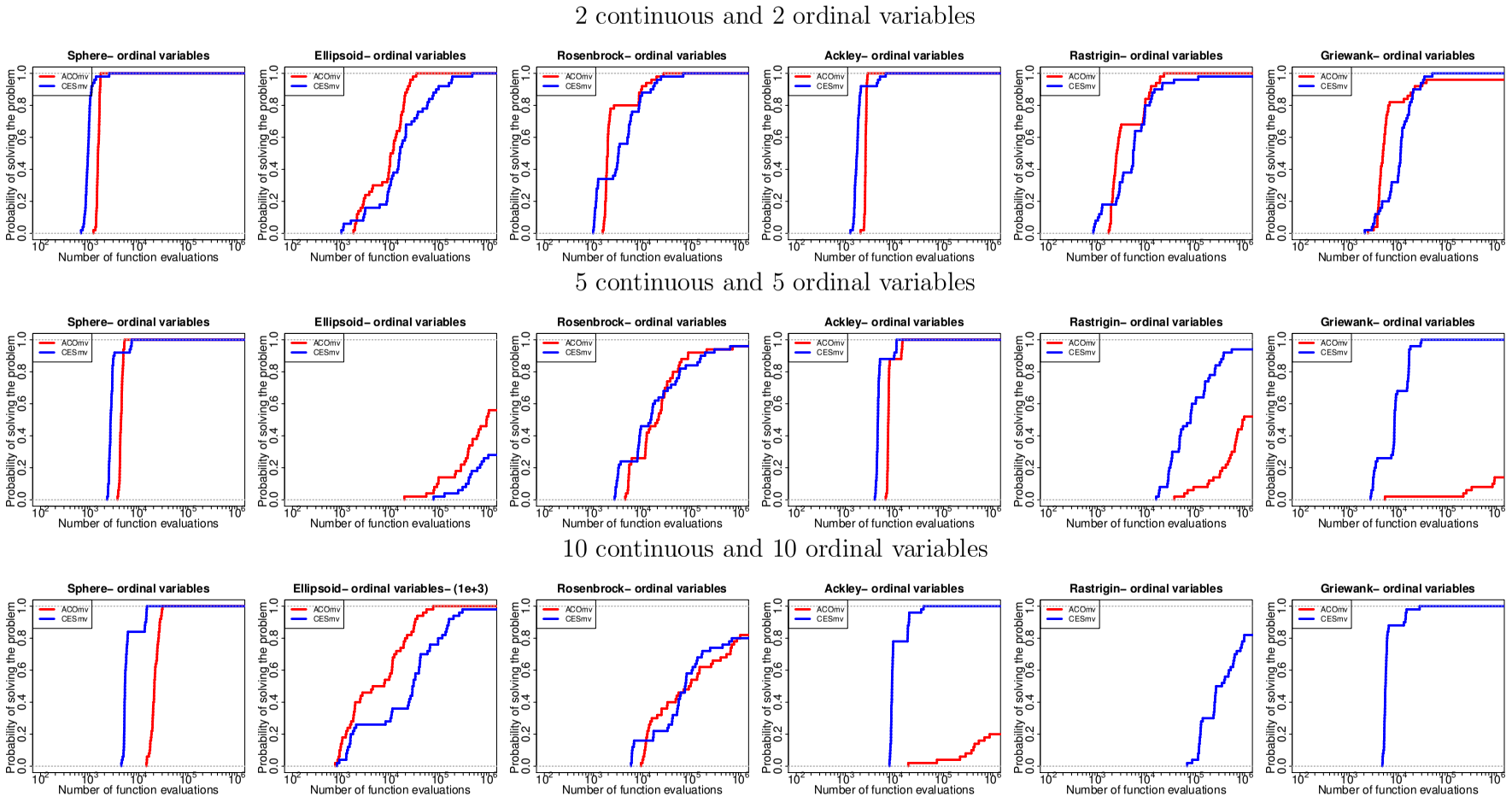


Figure 5.10: The qualified run-length distributions (RLDs, for short) over 50 independent runs obtained by $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$. The solution quality required is 1.00E−10 (zero threshold) except for the 20 dimensional $f_{Ellipsoid_{MV}}$ for which the solution quality required is shown in the title of the corresponding plot.

5.5 Application to engineering optimization problems

In this section, we apply $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ into mixed discrete-continuous engineering benchmark problems and compare the results of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ with those found in the literature. Note that our experiments comprise a larger set of benchmark problems than in any of the papers found in the literature, since these papers are often limited to a specific type of discrete variables (either ordinal or categorical). First, we classify the available engineering optimization problems in the literature into four groups according to the types of the decision variables used (see Table 5.6).

Table 5.6: The classification of engineering optimization problems.

Groups	The type of decision variables
Group I	Continuous variables[†]
Group II	Continuous and ordinal variables
Group III	Continuous and categorical variables
Group IV	Continuous, ordinal and categorical variables

[†] Problems with only continuous variables are considered as a particular class of mixed variables with an empty set of discrete variables, since $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ are also capable to solve pure continuous optimization problems.

Group I includes the welded beam design problem case A [Coello Coello, 2000]; Group II the pressure vessel design problem [Sandgren, 1990] and the coil spring design problem [Sandgren, 1990]; Group III the thermal insulation systems design problem [Kokkolaras et al., 2001]; and Group IV the welded beam design problem case B [Deb and Goyal, 1996]. The mathematical formulations of the problems are given in Appendix B. In this section, we compare the results obtained by $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ to those reported in the literature for these problems. We also show the run-time behavior of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ by using RLDs. It is important to note that NM-PSO [Zahara and Kao, 2009] and PSOLVER [Kayhan et al., 2010] report infeasible solutions that violate the problems' constraints; Črepinšek et al. [Črepinšek et al., 2012] pointed out that the authors of TLBO [Rao et al., 2011] used an incorrect formula for computing the number of objective function evaluations. Therefore, we did not include these three algorithms in our comparison. For our experiments, the tuned parameter configurations from Tables 5.3 and 5.4 were used. For simplifying the algorithm and giving prominence to the role of the $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ heuristic itself, the most fundamental constraint

handling technique was used, which consists in rejecting all infeasible solutions in the optimization process (also called "death penalty"). 100 independent runs were performed for each engineering problem. In the comparisons, f_{Best}, f_{Mean} and f_{Worst} are the abbreviations used to indicate the best, average and worst objective function values obtained, respectively. SR_B denotes the success rate of reaching the best known solution value. Sd gives the standard deviation of the mean objective function value; a value of Sd lower than 1.00E−10 is reported as 0. FEs gives the maximum number of objective function evaluations in each algorithm run. Note that the value of FEs may vary from algorithm to algorithm. To define the value of FEs for $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$, we first checked which is the smallest value of FEs used across all competing algorithms; let this value be denoted by $\mathrm{FEs}_{\min}$. Then the value of FEs for $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$ is set to $\mathrm{FEs}_{\min}$. Often, however, $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$ reached the best known solution values for the particular problem under concern in all runs (that is, with a 100% success rate) much faster than its competitors. In such cases, for $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$ we give, instead of the value $\mathrm{FEs}_{\min}$, in parenthesis the maximum number of objective function evaluations we observed across the 100 independent runs.

Group I : Welded beam design problem case A

Recently, many methods have been applied to the welded beam design problem case A. Table 5.7 shows basic summary statistics of the results obtained by nine other algorithms, $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$. Most other algorithms do not reach a success rate of 100% within a maximum number of objective function evaluations ranging from 30 000 (for $(\mu + \lambda)$ES [Mezura Montes and Coello Coello, 2005]) to 200 000 (for CPSO [He and Wang, 2007a]), while $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$ find the best-known solution value in every run using at most 2 303 and 2 070 objective function evaluations (measured across 100 independent trials). The only other algorithm that reaches the best-known solution value in every run is DELC [Wang and Li, 2010]; it does so using in every run at most 20 000 objective function evaluations (measured across 30 independent trials). Hence, $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$ are very efficient and robust algorithms for this problem. The run-time behavior of $\mathrm{ACO}_{\mathbf{MV}}$ and $\mathrm{CES}_{\mathbf{MV}}$ on this problems is illustrated also in Fig. 5.11, where the RLDs for this problem are given. The average and minimum number of objective function evaluations for $\mathrm{ACO}_{\mathbf{MV}}$ are 2 122 and 1 888, respectively. The average and minimum number of objective function evaluations for $\mathrm{CES}_{\mathbf{MV}}$ are 1 550 and 1 173, respectively.

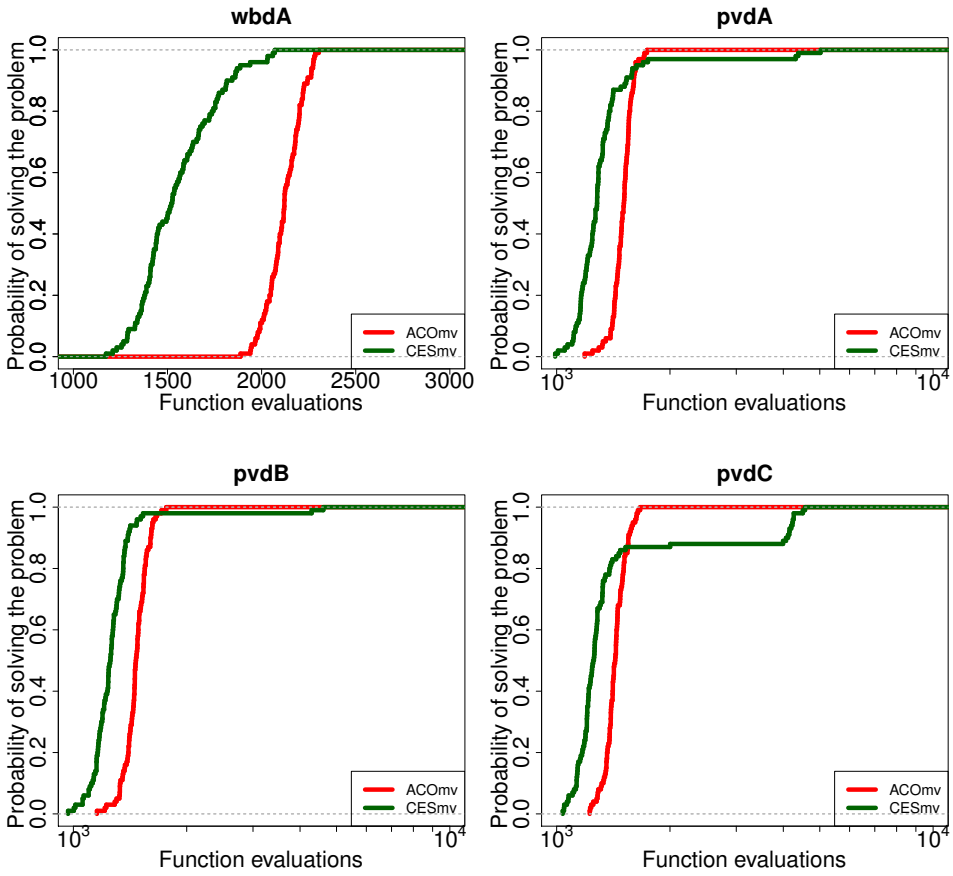


Figure 5.11: The RLDs of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ for the welded beam design problem case A and the pressure vessel design problem case A, B and C (wbdA, pvdA, pvdB and pvdC are the abbreviations of those problems, respectively).

Table 5.7: Basic summary statistics for the welded beam design problem case A. The best-known solution value is 1.724852. f_{Best}, f_{Mean} and f_{Worst} denote the best, mean and worst objective function values, respectively. Sd denotes the standard deviation of the mean objective function value. FEs denotes the maximum number of objective function evaluations in each algorithm run. For $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ we report in parenthesis the largest number of objective function evaluations it required in any of the 100 independent runs ($\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ reached in each run of at most 20 000 evaluations the best known solution value). "-" means that the information is not available.

Methods	f_{Best}	f_{Mean}	f_{Worst}	Sd	FEs
GA1 [Coello Coello, 2000]	1.748309	1.771973	1.785835	1.12E−02	-
GA2 [Coello Coello and Mezura Montes, 2002]	1.728226	1.792654	1.993408	7.47E−02	80 000
EP [Coello Coello and Becerra, 2004]	1.724852	1.971809	3.179709	4.43E−01	-
$(\mu + \lambda)$ES [Mezura Montes and Coello Coello, 2005]	1.724852	1.777692	-	8.80E−02	30 000
CPSO [He and Wang, 2007a]	1.728024	1.748831	1.782143	1.29E−02	200 000
HPSO [He and Wang, 2007b]	1.724852	1.749040	1.814295	4.01E−02	81 000
CLPSO [Gao and Hailu, 2010]	1.724852	1.728180	-	5.32E−03	60 000
DELC [Wang and Li, 2010]	1.724852	1.724852	1.724852	0	20 000
ABC [Akay and Karaboga]	1.724852	1.741913	-	3.10E−02	30 000
$\text{ACO}_{\mathbf{MV}}$	1.724852	1.724852	1.724852	0	(2 303)
$\text{CES}_{\mathbf{MV}}$	1.724852	1.724852	1.724852	0	(2 070)

Group II: Pressure vessel design problem case A, B, C and D

There are four distinct cases (A, B, C and D) of the pressure vessel design problem 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. In case A, B and C (see Table 5.8), $\text{ACO}_{\mathbf{MV}}$ reaches the best-known solution value with a 100% success rate in a maximum of 1 737, 1 764 and 1 666 objective function evaluations, respectively; the average number of objective function evaluations of the successful runs are 1 500, 1 470 and 1 433 respectively. $\text{CES}_{\mathbf{MV}}$ reaches the best-known solution value with a 100% success rate in a maximum of 5 021, 4 612 and 4 568 objective function evaluations, respectively; the average number of objective function evaluations of the successful runs are 1 381, 1 313 and 1 604 respectively. Other algorithms do not reach a success rate of 100% with respect to the best-known solution value even after many more objective function evaluations. The run-time behavior of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ is illustrated in Fig. 5.11, where the RLDs for these problems are given.

Case D is more difficult to solve due to the larger range of side constraints for decision variables. Therefore, Case D was analyzed in more detail in the recent literature. We limit $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ to use a maximum number of 30 000 objective function evaluations, the same as done for several other approaches from

Table 5.8: Results for case A, B and C of the pressure vessel design problem. f_{Best} denotes the best objective function value. SR_B denotes the success rate of reaching the best known solution value. FEs denotes the maximum number of objective function evaluations in each algorithm run. For ACO_{MV} and CES_{MV} we report in parenthesis the largest number of objective function evaluations it required in any of the 100 independent runs (ACO_{MV} and CES_{MV} reached in each run the best known solution value). Given is also the average number of objective function evaluations of the successful runs. "-" means that the information is not available.

Case A	NLIDP [Sandgren, 1990]	MIDCP [Fu et al., 1991]	DE [Lampinen and Zelinka, 1999c]			ACO_{MV}	CES_{MV}
f_{Best}	7 867.0	7 790.588	7 019.031			7 019.031	7 019.031
SR_B	-	-	89.2%			100%	100%
FEs	-	-	10 000			(1 737)	(5 021)
						(1 500.0)	(1 380.86)
Case B	NLIDP [Sandgren, 1990]	SLA [Loh and Papalambros, 1991]	GA [Wu and Chow, 1995]	DE [Lampinen and Zelinka, 1999c]	HSIA [Guo et al., 2004]	ACO_{MV}	CES_{MV}
f_{Best}	7 982.5	7 197.734	7 207.497	7 197.729	7 197.9	7 197.729	7 197.729
SR_B	-	-	-	90.2%	-	100%	100%
FEs	-	-	-	10 000	-	(1 764)	(4 612)
						(1 470.48)	(1 313.34)
Case C	NLMDP [Li and Chou, 1994]	EP [Cao and Wu, 1997]	ES [Thierauf and Cai, 2000]	DE [Lampinen and Zelinka, 1999c]	CHOPA [Schmidt and Thierauf, 2005]	ACO_{MV}	CES_{MV}
f_{Best}	7 127.3	7 108.616	7 006.9	7 006.358	7 006.51	7 006.358	7 006.358
SR_B	-	-	-	98.3%	-	100%	100%
FEs	-	-	4 800	10 000	10 000	(1 666)	(4 568)
						(1 433.42)	(1 603.73)

Table 5.9: Basic summary statistics for the pressure vessel design problem case D. The best-known objective function value is 6059.7143. f_{Best}, f_{Mean} and f_{Worst} denotes the best, mean and worst objective function values, respectively. Sd denotes the standard deviation of the mean objective function value. FEs denotes the maximum number of objective function evaluations in each algorithm run."-" means that the information is not available.

Methods	f_{Best}	f_{Mean}	f_{Worst}	Sd	FEs
GA1 [Coello Coello, 2000]	6 288.7445	6 293.8432	6 308.1497	7.413E+00	-
GA2 [Coello Coello and Mezura Montes, 2002]	6 059.9463	6 177.2533	6 469.3220	1.309E+02	80 000
$(\mu + \lambda)$ES [Mezura Montes and Coello Coello, 2005]	6 059.7143	6 379.9380	-	2.10E+02	30 000
CPSO [He and Wang, 2007a]	6 061.0777	6 147.1332	6 363.8041	8.645E+01	200 000
HPSO [He and Wang, 2007b]	6 059.7143	6 099.9323	6 288.6770	8.620E+01	81 000
RSPSO [Wang and Yin, 2008]	6 059.7143	6 066.2032	6 100.3196	1.33E+01	30 000
CLPSO [Gao and Hailu, 2010]	6 059.7143	6 066.0311	-	1.23E+01	60 000
DELC [Wang and Li, 2010]	6 059.7143	6 059.7143	6 059.7143	0	30 000
ABC [Akay and Karaboga]	6 059.7143	6 245.3081	-	2.05E+02	30 000
$\text{ACO}_{\mathbf{MV}}$	6 059.7143	6 059.7164	6 059.9143	1.94E−02	30 000
$\text{CES}_{\mathbf{MV}}$	6 059.7143	6 059.7480	6 089.9893	4.34E+00	30 000

the literature. Table 5.9 shows clearly the second and the third best performing algorithms for what concerns the average and the worst objective function values. In fact, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ reaches a 100% success rate (measured over 100 independent runs) at 30 717 and 43 739 objective function evaluations, while at 30 000 evaluations they reached a success rate of 98% and 76%, which are slightly lower than the success rate of 100% reported by DELC [Wang and Li, 2010]. The run-time behavior of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ are illustrated in Fig. 5.12, where the RLD for this problem is given. The average and minimum number of objective function evaluations of $\text{ACO}_{\mathbf{MV}}$ is 9 448 and 1 726, respectively. The average and minimum number of objective function evaluations of $\text{CES}_{\mathbf{MV}}$ is 19 675 and 1 417, respectively.

It is noteworthy that DELC [Wang and Li, 2010] reaches the aforementioned performance using parameter settings that are specific for each test problem, while we use a same parameter setting for all test problems. Using instance specific parameter settings potentially biases the results in favor of the DELC algorithm. In a practical setting, one would not know a priori which parameter setting to apply before actually solving the problem. Thus, there are methodological problems in the results presented for DELC [Wang and Li, 2010].

Group II: Coil spring design problem

Most of the research reported in the literature considering the coil spring design problem focused on reaching the best-known solution or improving the best-known one. Only recent work [Datta and Figueira, 2010, Lampinen and Zelinka, 1999c] gave some attention to the number of objective functions evaluations necessary to reach the best-known solution. A comparison of the obtained results is presented in Table 5.10. Only a differential evolution algorithm [Lampinen and Zelinka, 1999c], $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ obtained the best-known objective function value, 2.65856. At 8 000 evaluations $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ reached success rates of 74% and 44%, which are lower than the success rate of 95% reported by the DE algorithm of [Lampinen and Zelinka, 1999c]; however, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ reach 100% and 93% success rates with 19 588 and 47 792 objective function evaluations, respectively. The run-time behavior of $\text{ACO}_{\mathbf{MV}}$ is illustrated in Fig. 5.12, where the RLD for this problem is given. The average and minimum number of objective function evaluations of $\text{ACO}_{\mathbf{MV}}$ are 9 948 and 1 726, respectively. The average and minimum number of objective function evaluations of $\text{CES}_{\mathbf{MV}}$ are 12 107 and 527, respectively.

It is important to note that the DE algorithm of [Lampinen and Zelinka, 1999c] was not designed to handle categorical variables. Another DE algorithm proposed in [Datta and Figueira, 2010] did not report a success rate, but the corresponding objective function values were reported to be in the range of $[2.658565, 2.658790]$ and the number of objective function evaluations varied in the range $[539\,960, 3\,711\,560]$, thus, showing a clearly worse performance than $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$.

Group III: Thermal insulation systems design problem

The thermal insulation systems design problem is one of the engineering problems used in the literature that deals with categorical variables. In previous studies, the categorical variables describing the type of insulators used in different layers were not considered as optimization variables, but rather as parameters. Only the more recent work of Kokkolaras et al. [Kokkolaras et al., 2001] and Abramson et al. [Abramson, 2004], which are able to handle such categorical variables properly, consider these variables for optimization. Research focuses on improving the best-known solution value for this difficult engineering problem. $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ reaches a better solution than MVP [Kokkolaras et al., 2001] and FMGPS [Abramson, 2004]; Table 5.11 presents the best found solutions by the four algorithms we

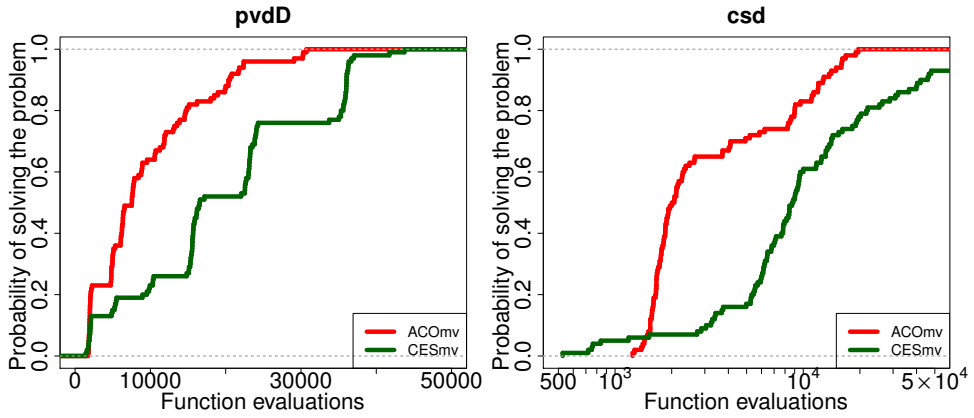


Figure 5.12: The RLDs of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ for the pressure vessel design problem case D and the coil spring design problem (pvdD and csd are the abbreviations of those problems, respectively).

Table 5.10: Results for the coil spring design problem. f_{Best} denotes the best objective function value. SR_B denotes the success rate of reaching the best known solution value. FEs denotes the maximum number of objective function evaluations in each algorithm run. For $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ we report in parenthesis the largest number of objective function evaluations it required in any of the 100 independent runs ($\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ reached in each run the best known solution value). "-" means that the information is not available.

Algs	NLIDP [Sandgren, 1990]	GA [Chen and Tsao, 1993]	GA [Wu and Chow, 1995]	DE [Lampinen and Zelinka, 1999c]
N	10	9	9	9
D [inch]	1.180701	1.2287	1.227411	1.223041
d [inch]	0.283	0.283	0.283	0.283
f_{Best}	2.7995	2.6709	2.6681	2.65856
SR_B	-	-	-	95.0%
FEs	-	-	-	8 000
Algs	**HSIA [Guo et al., 2004]**	**DE [Datta and Figueira, 2010]**	**$\text{ACO}_{\mathbf{MV}}$**	**$\text{CES}_{\mathbf{MV}}$**
N	9	9	9	9
D [inch]	1.223	1.223044	1.223041	1.223041
d [inch]	0.283	0.283	0.283	0.283
f_{Best}	2.659	2.658565	2.65856	2.65856
SR_B	-	-	74% (100%)	44% (93%)
FEs	-	-	8 000 (19 588)	8 000 (47 792)

Table 5.11: Comparison of the best fitness value for the thermal insulation systems design problem

Objective function	MVP [Kokkolaras et al., 2001]	FMGPS [Abramson, 2004]	ACO_{MV}	CES_{MV}
Power($\frac{PL}{A}(\frac{W}{cm})$)	25.294	25.58	24.148	23.914

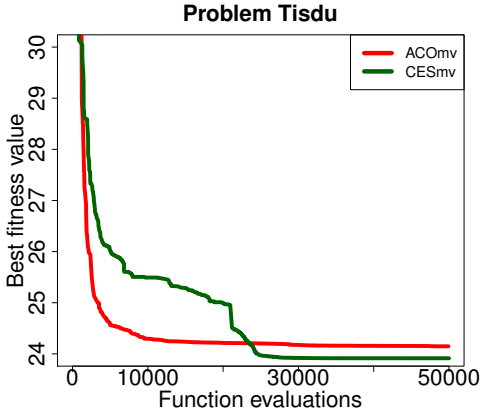


Figure 5.13: The best solution trail of ACO_{MV} and CES_{MV} for the thermal insulation systems design problem

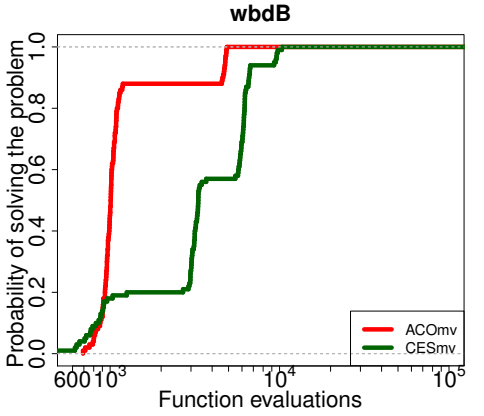


Figure 5.14: The RLDs of ACO_{MV} and CES_{MV} for the welded beam design problem case B (wbdB is its abbreviation).

compare here. Both, ACO_{MV} and CES_{MV} improve over the best solution found by MVP and FMGPS.

The evolution of the best solution as a function of number of objective function evaluations of ACO_{MV} and CES_{MV} are shown in Fig.5.13. In fact, as the number of objective function evaluations increases, the solution quality continues to improve, though ACO_{MV} appears to show issues of early convergence.

Table 5.12: Basic summary statistics for welded beam design design problem case B. f_{Best} and f_{Mean} denotes the best and mean objective function values, respectively. Sd denotes the standard deviation of the mean objective function value. Mean-FEs-Success denotes the average number of evaluations of the successful runs. "-" means that the information is not available.

Methods	f_{Best}	f_{Mean}	Sd	Mean-FEs-Success
GeneAS [Deb and Goyal, 1996]	1.9422	-	-	-
RSPSO [Wang and Yin, 2008]	1.9421	-	-	-
PSOA[Dimopoulos, 2007]	1.7631	1.7631	0	6 570
CLPSO [Gao and Hailu, 2010]	1.5809	1.7405	2.11E−01	-
$\text{ACO}_{\mathbf{MV}}$	1.5029	1.5029	0	1 436
$\text{CES}_{\mathbf{MV}}$	1.5029	1.5029	0	4 188

Group IV: Welded beam design problem case B

The welded beam design problem case B is taken from Deb and Goyal [Deb and Goyal, 1996] and Dimopoulos [Dimopoulos, 2007]. It is a variation of case A and it includes ordinal and categorical variables. Table 5.12 shows that $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ reaches a new best-known solution value with a 100% success rate. Additionally, the average number of objective function evaluations required by $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ are also fewer than that of PSOA [Dimopoulos, 2007]. The run-time behavior of $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ is illustrated in Fig. 5.14.

5.6 Summary

In this chapter, we have presented two algorithms for mixed discrete continuous optimization problems. One of these algorithms is an ant colony optimization algorithm, $\text{ACO}_{\mathbf{MV}}$, that extends the $\text{ACO}_{\mathbb{R}}$ method to mixed discrete continuous optimization. In particular, $\text{ACO}_{\mathbf{MV}}$ integrates a continuous optimization solver ($\text{ACO}_{\mathbb{R}}$), a continuous relaxation approach ($\text{ACO}_{\mathbf{MV}}$-o) and a categorical optimization approach ($\text{ACO}_{\mathbf{MV}}$-c) to solve continuous and mixed discrete-continuous optimization problems. As a second algorithm, we have introduced $\text{CES}_{\mathbf{MV}}$, an covariance matrix adaptation evolution strategy algorithm for tackling mixed discrete-continuous optimization problems. $\text{CES}_{\mathbf{MV}}$ integrates a continuous optimization solver (CMA-ES), a continuous relaxation approach (Round, same as $\text{ACO}_{\mathbf{MV}}$-o) and a categorical optimization approach ($\text{CES}_{\mathbf{MV}}$-c) to solve continuous and mixed discrete-continuous optimization problems. The categorical optimization part of $\text{CES}_{\mathbf{MV}}$ is adapted in a rather straightforward way from the way $\text{ACO}_{\mathbf{MV}}$ handles categorical variables. We have also proposed and evaluated other CMA-ES extensions we proposed for tackling mixed discrete-continuous

optimization problems, and identify $\text{CES}_{\mathbf{MV}}$ as the most performing variant.

We have also proposed a new set of artificial mixed discrete-continuous benchmark functions.

We evaluate $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ on the artificial mixed discrete-continuous benchmark functions and apply them into real-world engineering problems. The experimental results for real-world engineering problems illustrate that $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ not only can tackle various classes of decision variables robustly, but also that they are efficient in finding high-quality solutions when compared to many other algorithms from the literature. In the welded beam design case A, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ are the ones of the three available algorithms that reach the best-known solution with a 100% success rate; in the pressure vessel design problem case A, B and C, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ are the only two available algorithms that reach the best-known solution with a 100% success rate. In these four problems, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ do so using fewer objective function evaluations than those used by the competing algorithms. In the pressure vessel design problem case D, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ are two of the three available algorithms that reach the best-known solution with a 100% success rate, and they do so using only slightly more objective function evaluations than the other algorithm, which uses problem specific parameter tuning to boost algorithm performance. In the coil spring design problem, $\text{ACO}_{\mathbf{MV}}$ is the only available algorithm that reaches the best-known solution with a 100% success rate. In the thermal insulation systems design problem, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ obtains new best solutions, and in the welded beam design problem case B, $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ obtain new best solutions with a 100% success rate in fewer evaluations than those used by the other algorithms.

Chapter 6

Summary and future work

6.1 Summary

This thesis focuses on the design, engineering and configuration of high-performing heuristic optimization algorithms for tackling continuous and mixed discrete-continuous optimization problems. Continuous optimization problems have variable domains that typically are a subset of the real numbers; mixed discrete-continuous optimization problems have additionally discrete variables, so that some variables are continuous and others are on an ordinal or categorical scale. Continuous and mixed discrete-continuous problems have a wide range of applications in many disciplines and these problems are also often hard to solve due to their inherent difficulties such as a large number of variables, many local optima or other factors making problems hard.

We tackle continuous and mixed discrete-continuous optimization problems with two types of population-based heuristic algorithms, ant colony optimization (ACO) algorithms and evolution strategies. The main contributions of this thesis are that (i) we advance the design and engineering of ACO algorithms; the resulting algorithms are competitive or superior to recent state-of-the-art algorithms for continuous and mixed discrete-continuous optimization problems, (ii) we propose a new hybrid algorithm that improves upon a state-of-the-art evolution strategy, the covariance matrix adaptation evolution strategy (CMA-ES), and (iii) we develop effective algorithms based on the ACO framework and CMA-ES, respectively, to tackle mixed discrete-continuous optimization problems.

More in detail, we propose a unified ant colony optimization (ACO) framework for continuous optimization (UACOR). This framework synthesizes algorithmic components of two ACO algorithms ($\text{ACO}_{\mathbb{R}}$ and $\text{DACO}_{\mathbb{R}}$) that have been proposed in the literature and an incremental ACO algorithm with local search for continuous optimization ($\text{IACO}_{\mathbb{R}}$-LS). $\text{IACO}_{\mathbb{R}}$-LS is a variant of $\text{ACO}_{\mathbb{R}}$ that uses

local search and that features a growing solution archive. We proposed IACO$_{\mathbb{R}}$-LS during my doctoral research and the paper describing IACO$_{\mathbb{R}}$-LS received the best paper award of the ACO-SI track at the GECCO 2011 conference. The UACOR framework is flexible and its design allows the usage of automatic algorithm configuration techniques to automatically derive new ACO algorithms for continuous optimization. The computational results showed that the automatically configured ACO algorithms obtain better performance on the SOCO and CEC'05 benchmark sets than the tuned variants of the three ACO algorithms that underly the UACOR framework, namely ACO$_{\mathbb{R}}$, DACO$_{\mathbb{R}}$ and IACO$_{\mathbb{R}}$-LS. When UACOR is automatically configured for the SOCO benchmark set, it performs better or competitive to all the recent 16 algorithms benchmarked on this benchmark set; when configured for the CEC'05 benchmark set, it performs competitive to IPOP-CMA-ES, a state-of-the-art algorithm on this benchmark set and also competitive or superior to other five recent high-performance continuous optimizers that were evaluated on this benchmark set. Finally, we proposed UACOR^{+}, a re-designed and improved UACOR that includes the option of using CMA-ES as a local search. This helps the UACOR framework, as also shown experimentally, to improve results on rotated functions, thus, mainly on the CEC benchmark set. For example, when configured for the CEC'05 benchmark set, it performs superior to IPOP-CMA-ES and statistically significantly better than other five recent high-performance continuous optimizers that were evaluated on this benchmark set. In summary, we have proven the high potential ACO algorithms have for continuous optimization and that automatic algorithm configuration has a high potential also for the development of continuous optimizers out of algorithm components.

In later work, summarized in Chapter 4, we proposed iCMAES-ILS, a hybrid algorithm that loosely couples IPOP-CMA-ES and a new iterated local search (ILS) algorithm for continuous optimization. The hybrid algorithm starts with a competition phase, where IPOP-CMA-ES and the ILS algorithm compete for deployment in a second phase. Some information exchange from IPOP-CMA-ES to the ILS algorithm during the competition phase implements some cooperative aspect. iCMAES-ILS reaches excellent performance on the CEC'05 benchmark function set. The computational results with a default parameter settings and further fine-tuned parameter settings establish iCMAES-ILS as a state-of-the-art algorithm for continuous optimization. In fact, iCMAES-ILS improves statistically significantly over IPOP-CMA-ES, the ILS algorithm, various other state-of-the-art algorithms and it gives better results than other possible hybrid designs such as iCMAES-ILS-portfolio, iCMAES-ILS-relay and iCMAES-LTH-ILS. As a final

step, we compare iCMAES-ILS to UACOR$^+$, the UACOR version that included CMA-ES as a local search, on the CEC'05 and SOCO benchmark functions set, respectively. Each of iCMAES-ILS and UACOR$^+$ shows own advantages with respect to the two different benchmark function sets. iCMAES-ILS performs statistically significantly better than UACOR$^+$ on the CEC'05 benchmark functions set; UACOR$^+$ performs statistically significantly better than iCMAES-ILS on the SOCO benchmark functions set.

To tackle mixed discrete-continuous optimization problems, we extend ACO$_{\mathbf{MV}}$ and propose CES$_{\mathbf{MV}}$, an ant colony optimization algorithm and a covariance matrix adaptation evolution strategy, respectively. In ACO$_{\mathbf{MV}}$ and CES$_{\mathbf{MV}}$, the decision variables of an optimization problem can be declared as continuous, ordinal, or categorical, which allows the algorithm to treat them adequately. ACO$_{\mathbf{MV}}$ and CES$_{\mathbf{MV}}$ include three solution generation mechanisms: a continuous optimization mechanism, a continuous relaxation mechanism for ordinal variables, and a categorical optimization mechanism for categorical variables. Together, these mechanisms allow ACO$_{\mathbf{MV}}$ and CES$_{\mathbf{MV}}$ to tackle mixed variable optimization problems. We also have proposed and evaluated other CMA-ES extensions for mixed-variable optimization problems, but we identify CES$_{\mathbf{MV}}$ as the most performing variant. We also propose a set of artificial, mixed-variable benchmark functions, which can simulate discrete variables as ordered or categorical. These benchmark functions provide a sufficiently controlled environment for the investigation of the performance of mixed-variable optimization algorithms, and a training environment for automatic parameter tuning. We use them to automatically tune ACO$_{\mathbf{MV}}$ and CES$_{\mathbf{MV}}$'s parameters and benchmark their performance. Finally, we test ACO$_{\mathbf{MV}}$ and CES$_{\mathbf{MV}}$ on various real-world continuous and mixed-variable engineering optimization problems. The experimental results for real-world engineering problems illustrate that ACO$_{\mathbf{MV}}$ and CES$_{\mathbf{MV}}$ not only can tackle various classes of decision variables robustly, but also that it is efficient in finding high-quality solutions.

Apart from these main contributions, during my doctoral research I have accomplished a number of additional contributions, which concern (i) a note on the bound constraints handling for the CEC'05 benchmark set, (ii) computational results for an automatically tuned IPOP-CMA-ES on the CEC'05 benchmark set and (iii) a study of artificial bee colonies for continuous optimization. These additional contributions are found in the appendix to this thesis.

6.2 Future work

There are a number of relevant research directions to extend the work presented in this thesis. One promising direction is to extend UACOR to synthesize other probability density functions for the generation of candidate solutions, alternative ways of handling the archive and other local search algorithms. Another possibility would be to design a more general algorithm framework from which different types of continuous optimizers other than ACO algorithms can be automatically configured.

Several directions appear promising in the future work of iCMAES-ILS. A first direction is to further refine the design of the hybrid algorithm by integrating other algorithms that are complementary in performance with respect to the already included IPOP-CMA-ES and ILS. One possibility, for example would be to use an algorithm based on UACOR. An interesting alternative is also to automatize the design of the hybrid algorithm exploiting recent ideas underlying the automatic design of algorithm portfolios and algorithm selection [Xu et al., 2010]. Recent work on the definition of function features for continuous optimization problems may prove useful in this respect [Bischl et al., 2012, Mersmann et al., 2011]. Finally, the ideas pursued in the design of our algorithms may be extended to address large-scale, noisy or dynamic problems [Hansen et al., 2009b, Lozano et al., 2011, Morrison and Jong, 1999] and to tackle constrained continuous optimization problems.

A promising application for $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ are algorithm configuration problems [Birattari et al., 2010], in which typically not only the setting of numerical parameters but also that of categorical parameters needs to be determined. To do so, we will integrate $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ into the `irace` framework [López-Ibáñez et al., 2011]. Another promising direction is use $\text{ACO}_{\mathbf{MV}}$ and $\text{CES}_{\mathbf{MV}}$ to tackle constrained mixed discrete-continuous optimization problems in the real world with an effective constraint-handling technique.

Appendices

Appendix A

The results obtained by UACOR$^+$

In Appendix A, we give some more detailed results for the analysis of UACOR$^+$ and a comparison of its performance to other algorithms from the literature.

- Table A.1 gives the parameter settings we obtained when tuning UACOR$^+$ for the SOCO and the CEC'05 benchmark sets, respectively.
- Figures A.1 and A.2 give the algorithm flowchart for UACOR$^+$ tuned on the SOCO and CEC'05 benchmark sets.
- Figure A.3 compares UACOR$^+$-s and the three ACO algorithms, ACO$_{\mathbb{R}}$-s, DACO$_{\mathbb{R}}$-s and IACO$_{\mathbb{R}}$-Mtsls1-s; shows the benefit of the incremental archive mechanism used by UACOR$^+$-s when compared to a fixed archive size.
- Figure A.4 compares UACOR$^+$-s with all 13 candidate algorithms published in the SOCO special issue and to the three algorithms that were chosen as reference algorithms.
- Table A.2 shows the average error values of UACOR$^+$-s and the three ACO algorithms, ACO$_{\mathbb{R}}$-s, DACO$_{\mathbb{R}}$-s and IACO$_{\mathbb{R}}$-Mtsls1-s, and MOS-DE on the SOCO benchmark functions.
- Table A.3 shows the average error values of UACOR$^+$-c and the three ACO algorithms, ACO$_{\mathbb{R}}$-c, DACO$_{\mathbb{R}}$-c and IACO$_{\mathbb{R}}$-Mtsls1-c, and IPOP-CMA-ES on the CEC'05 benchmark functions.
- Tables A.4 and A.5 compare UACOR$^+$-c with five recent state-of-the-art continuous optimization algorithms.

.

Table A.1: The left part of the table gives the list of parameter settings and their domains. Some settings are only significant for certain values of other settings. The parameter settings of the automatically configured algorithms are given in the central part and the right part, depending on which training set of benchmark functions was used for tuning.

Module	Para Name	Type	Domain	Tuning on SOCO	Tuning on CEC'05
				UACOR$^+$-s	UACOR$^+$-c
Mode	*DefaultMode*	c	{T, F}	**T**	**T**
	EliteQ$_{best}$	r	[0, 1]	*	*
DefNants	*InitAS*	i	[20, 100]	**54**	**85**
	NaIsAS	c	{T, F}	**F**	**T**
	Na	i	[2, 20]	**14**	*
SolConstr	Q_{best}	r	[0, 1]	**0.2365**	**0.4582**
	WeightGsol	c	{T, F}	**T**	**F**
	q	r	(0, 1)	**0.3091**	*
	ξ	r	(0, 1)	**0.6934**	**0.6753**
SAUpdate	*RmLocalWorse*	c	{T, F}	**F**	**T**
	SnewvsGsol	c	{T, F}	*	**T**
LS	*LsType*	c	{F, Powell, Mtsls1, CMA-ES}	**Mtsls1**	**CMA-ES**
	LsIter	i	[1, 100]	**86**	*****
	LsFailures	i	[1, 20]	**6**	**3**
IncArch	*IsIncrement*	c	{T, F}	**T**	**T**
	GrowthIter	i	[1, 30]	**5**	**11**
RestartMech	*RestartType*	c	{F, 1st, 2nd}	**F**	**2nd**
	StagIter	r	[1, 1000]	*	**8**
	StagThresh	r	[-15, 0]	*	**-3.189**
	Shakefactor	r	[-15, 0]	*	**-0.03392**
	RestartAS	i	[2, 100]	*	**12**

* denotes the value of the parameter is not relevant for the corresponding algorithm.

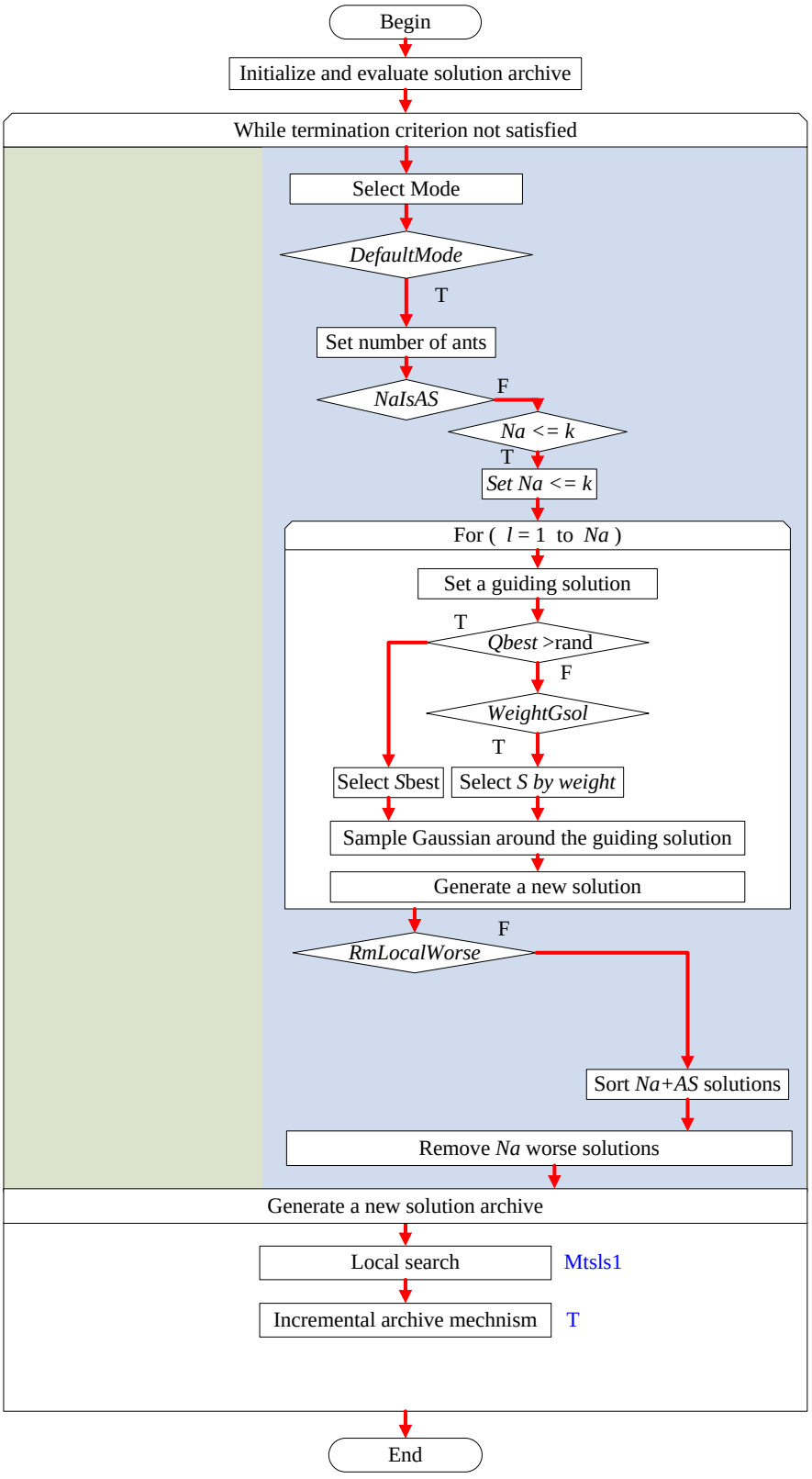


Figure A.1: UACOR$^+$-s is highlighted in the flowchart of UACOR$^+$.

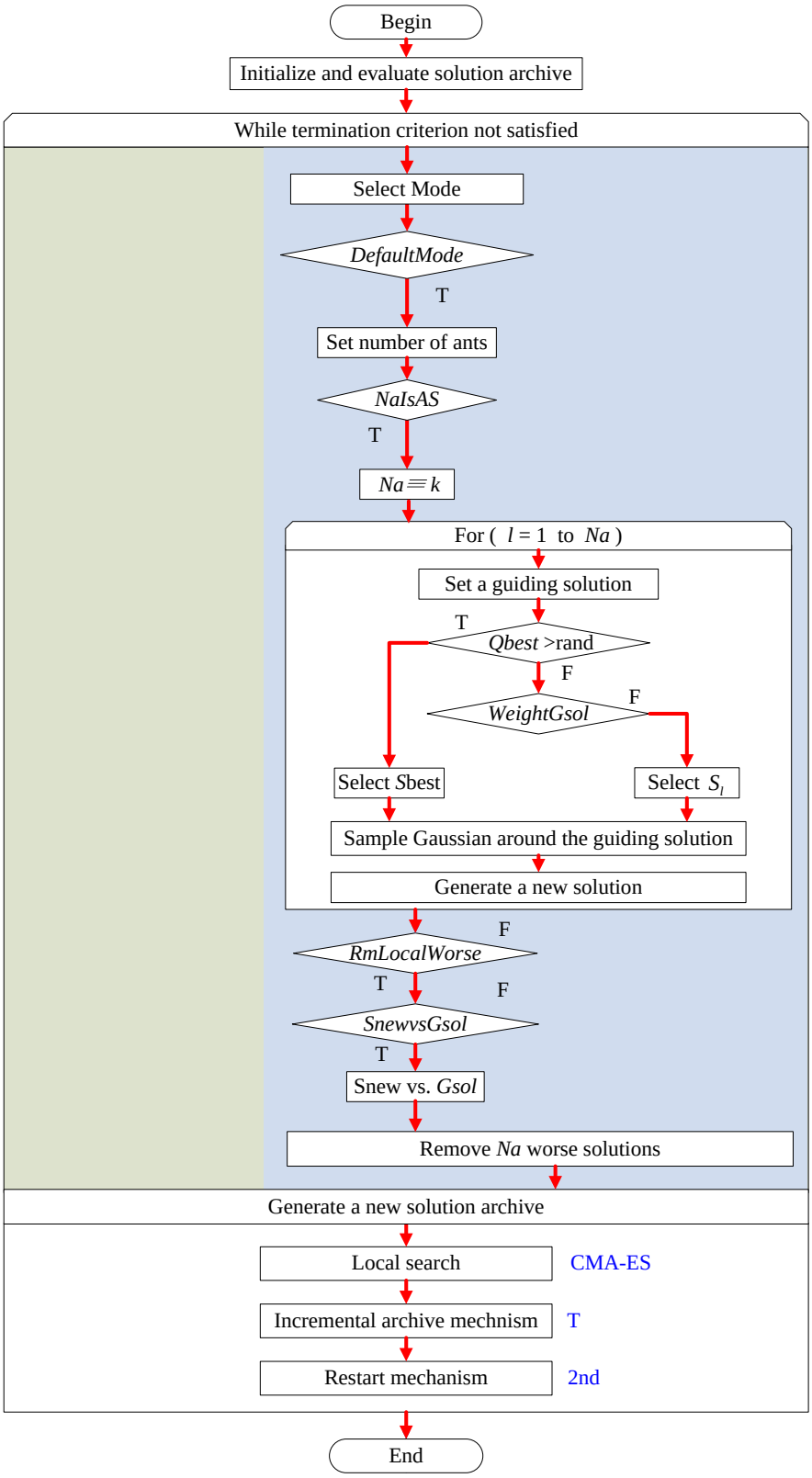


Figure A.2: UACOR+-c is highlighted in the flowchart of UACOR+.

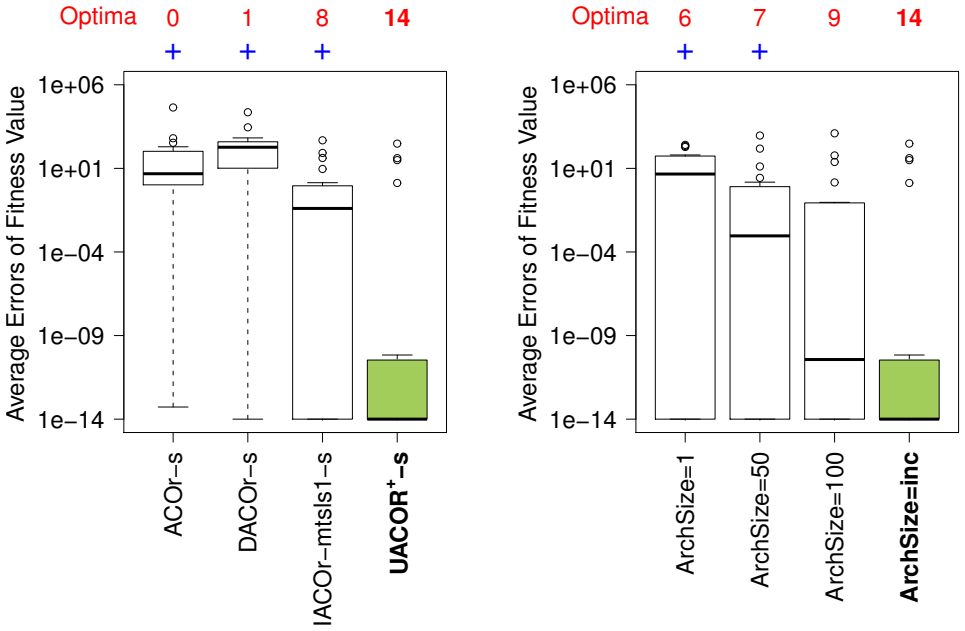
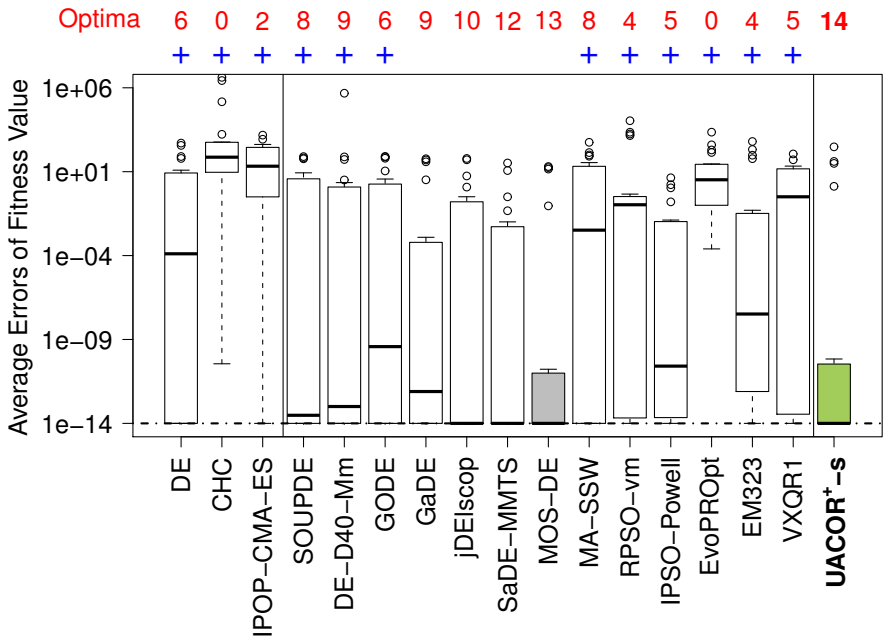


Figure A.3: The box-plots show the distribution of the average errors obtained on the 19 SOCO benchmark functions of dimension 100. The left plot compares the performance of UACOR$^+$-s with ACO$_{\mathbb{R}}$-s, DACO$_{\mathbb{R}}$-s and IACO$_{\mathbb{R}}$-Mtsls1-s. The right plot shows the benefit of the incremental archive size used in UACOR$^+$-s. A + symbol on top of each box-plot denotes a statistically significant difference at the 0.05 α-level between the results obtained by the indicated algorithm and those obtained with UACOR$^+$-s. The absence of a symbol means that the difference is not statistically significant. The numbers on top of a box-plot denote the number of the averages below the optimum threshold 10^{-14} found by the indicated algorithms.



16 algorithms in SOCO

Figure A.4: The box-plot show the distribution of the average errors obtained on the 19 SOCO benchmark functions of dimension 100. The results obtained by the three reference algorithms (left), 13 algorithms (middle) published in SOCO and UACOR$^+$-s (right) are shown on the plot. The line at the bottom of the boxplot represents the optimum threshold (10^{-14}). A + symbol on top of the two box-plot denotes a statistically significant difference at the 0.05 α-level between the results obtained with the indicated algorithm and those obtained with UACOR$^+$-s detected with a Friedman test and its associated post test on the 17 algorithms. The absence of a symbol means that the difference is not significant. The numbers on top of a box-plot denote the number of averages below the optimum threshold 10^{-14} found by the indicated algorithms.

Table A.2: The average errors obtained by $ACO_{\mathbb{R}}$-s, $DACO_{\mathbb{R}}$-s, $IACO_{\mathbb{R}}$-Mtsls1-s, MOS-DE and UACOR^{+}-s for each SOCO function. The numbers in parenthesis at the bottom of the table represent the number of times an algorithm is better, equal or worse, respectively, than UACOR^{+}-s. Error values lower than 10^{-14} are approximated to 10^{-14}. The average errors that correspond to a better result between MOS-DE and UACOR^{+}-c are highlighted.

Dim	f_{soco}	$ACO_{\mathbb{R}}$-s	$DACO_{\mathbb{R}}$-s	$IACO_{\mathbb{R}}$-Mtsls1-s	MOS-DE	UACOR^{+}-s
100	f_{soco1}	5.32E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
	f_{soco2}	2.77E+01	3.82E+01	5.27E−12	2.94E−12	6.86E−11
	f_{soco3}	1.96E+02	2.86E+03	4.77E+02	2.03E+01	3.02E+02
	f_{soco4}	6.34E+02	3.89E+02	1.00E−14	1.00E−14	1.00E−14
	f_{soco5}	2.96E−04	4.96E−01	1.00E−14	1.00E−14	1.00E−14
	f_{soco6}	2.04E−08	3.49E+00	1.00E−14	1.00E−14	1.00E−14
	f_{soco7}	9.94E−10	5.01E−01	1.00E−14	1.00E−14	1.00E−14
	f_{soco8}	4.39E+04	2.28E+04	1.39E+00	9.17E−02	1.34E+00
	f_{soco9}	4.78E+00	1.84E+02	1.62E−01	1.00E−14	1.00E−14
	f_{soco10}	2.75E+00	2.09E+01	1.00E−14	1.00E−14	1.00E−14
	f_{soco11}	3.96E+00	1.90E+02	1.67E−01	1.00E−14	1.00E−14
	f_{soco12}	1.13E+01	2.39E+02	4.01E−02	1.00E−14	1.00E−14
	f_{soco13}	1.47E+02	3.92E+02	4.19E+01	1.75E+01	3.12E+01
	f_{soco14}	3.40E+02	2.40E+02	9.23E+00	1.68E−11	1.00E−14
	f_{soco15}	5.89E−01	4.41E+00	1.00E−14	1.00E−14	1.00E−14
	f_{soco16}	1.61E+00	4.18E+02	4.24E−01	1.00E−14	1.00E−14
	f_{soco17}	6.27E+01	6.65E+02	8.40E+01	1.43E+01	4.08E+01
	f_{soco18}	1.57E+01	1.17E+02	1.07E−01	1.00E−14	1.00E−14
	f_{soco19}	1.48E+00	1.61E+01	1.00E−14	1.00E−14	1.00E−14
By f_{soco}		(1, 0, 18)†	(0, 1, 18)†	(1, 8, 10)†	(5, 13, 1)	

† denotes a significant difference between the corresponding algorithm and UACOR^{+}-s by a Friedman test at the 0,05 α-level over the distribution of average errors of $ACO_{\mathbb{R}}$-s, $DACO_{\mathbb{R}}$-s, $IACO_{\mathbb{R}}$-Mtsls1-s and UACOR^{+}-s.

Table A.3: The average errors obtained by ACO$_\mathbb{R}$-c, DACO$_\mathbb{R}$-c, IACO$_\mathbb{R}$-Mtsls1-c, IPOP-CMA-ES and UACOR$^+$-c for each CEC'05 function. The numbers in parenthesis at the bottom of the table represent the number of times an algorithm is better, equal or worse, respectively, compared to UACOR$^+$-c. Error values lower than 10^{-8} are approximated to 10^{-8}. The average errors that correspond to a better result between IPOP-CMA-ES and UACOR$^+$-c are highlighted.

Dim	f_{cec}	ACO$_\mathbb{R}$-c	DACO$_\mathbb{R}$-c	IACO$_\mathbb{R}$-Mtsls1-c	IPOP-CMA-ES	UACOR$^+$-c
30	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.00E−08	4.74E+00	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	3.88E+05	4.21E+06	2.19E+05	1.00E−08	1.00E−08
	f_{cec4}	2.75E−04	2.62E+02	9.45E+03	1.11E+04	1.14E+03
	f_{cec5}	1.00E−08	1.00E−08	6.79E−08	1.00E−08	1.00E−08
	f_{cec6}	8.99E+00	2.50E+01	1.64E+02	1.00E−08	1.00E−08
	f_{cec7}	1.80E−02	1.54E−02	1.00E−02	1.00E−08	1.00E−08
	f_{cec8}	2.00E+01	2.02E+01	2.00E+01	2.01E+01	2.00E+01
	f_{cec9}	2.50E+01	6.35E+01	1.00E−08	9.38E−01	2.95E+01
	f_{cec10}	4.51E+01	6.58E+01	1.19E+02	1.65E+00	2.91E+01
	f_{cec11}	3.75E+01	3.19E+01	2.36E+01	5.48E+00	3.82E+00
	f_{cec12}	3.59E+03	1.92E+04	7.89E+03	4.43E+04	2.21E+03
	f_{cec13}	3.67E+00	4.57E+00	1.28E+00	2.49E+00	2.26E+00
	f_{cec14}	1.25E+01	1.34E+01	1.32E+01	1.29E+01	1.32E+01
	f_{cec15}	3.40E+02	3.28E+02	2.48E+02	2.08E+02	1.92E+02
	f_{cec16}	1.33E+02	1.93E+02	2.49E+02	3.50E+01	6.14E+01
	f_{cec17}	1.49E+02	1.81E+02	3.35E+02	2.91E+02	2.87E+02
	f_{cec18}	9.12E+02	9.07E+02	9.02E+02	9.04E+02	8.73E+02
	f_{cec19}	9.11E+02	9.07E+02	8.92E+02	9.04E+02	8.83E+02
	f_{cec20}	9.12E+02	9.07E+02	8.97E+02	9.04E+02	8.78E+02
	f_{cec21}	5.38E+02	5.00E+02	5.12E+02	5.00E+02	5.00E+02
	f_{cec22}	9.08E+02	8.70E+02	9.90E+02	8.03E+02	8.57E+02
	f_{cec23}	5.75E+02	5.35E+02	5.66E+02	5.34E+02	5.34E+02
	f_{cec24}	2.27E+02	7.85E+02	1.26E+03	9.10E+02	3.41E+02
	f_{cec25}	2.19E+02	2.31E+02	5.60E+02	2.11E+02	2.63E+02
50	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	1.80E−04	2.61E+03	4.08E−07	1.00E−08	1.00E−08
	f_{cec3}	6.68E+05	6.72E+06	5.60E+05	1.00E−08	1.00E−08
	f_{cec4}	8.36E+03	4.69E+04	5.33E+04	4.68E+05	1.69E+04
	f_{cec5}	2.23E−05	2.02E−01	7.95E−07	2.85E+00	1.00E−08
	f_{cec6}	2.92E+01	5.18E+01	1.73E+02	1.00E−08	1.00E−08
	f_{cec7}	9.93E−03	1.08E−02	4.53E−03	1.00E−08	1.00E−08
	f_{cec8}	2.00E+01	2.02E+01	2.00E+01	2.01E+01	2.01E+01
	f_{cec9}	4.82E+01	1.15E+02	1.00E−08	1.39E+00	8.18E+01
	f_{cec10}	9.77E+01	1.42E+02	2.83E+02	1.72E+00	7.45E+01
	f_{cec11}	7.30E+01	5.81E+01	4.63E+01	1.17E+01	1.07E+01
	f_{cec12}	2.74E+04	1.07E+05	1.47E+04	2.27E+05	1.27E+04
	f_{cec13}	7.24E+00	1.06E+01	2.13E+00	4.59E+00	4.76E+00
	f_{cec14}	2.24E+01	2.29E+01	2.26E+01	2.29E+01	2.31E+01
	f_{cec15}	3.26E+02	3.61E+02	2.64E+02	2.04E+02	1.74E+02
	f_{cec16}	1.10E+02	1.64E+02	2.89E+02	3.09E+01	7.01E+01
	f_{cec17}	1.62E+02	2.45E+02	5.65E+02	2.34E+02	3.17E+02
	f_{cec18}	9.33E+02	9.26E+02	9.31E+02	9.13E+02	9.08E+02
	f_{cec19}	9.34E+02	9.27E+02	9.18E+02	9.12E+02	8.76E+02
	f_{cec20}	9.36E+02	9.27E+02	9.19E+02	9.12E+02	8.64E+02
	f_{cec21}	5.39E+02	9.82E+02	5.12E+02	1.00E+03	5.00E+02
	f_{cec22}	9.48E+02	9.07E+02	1.06E+03	8.05E+02	8.92E+02
	f_{cec23}	5.56E+02	1.02E+03	5.53E+02	1.01E+03	5.41E+02
	f_{cec24}	2.94E+02	9.06E+02	1.40E+03	9.55E+02	8.26E+02
	f_{cec25}	2.63E+02	3.39E+02	9.52E+02	2.15E+02	3.53E+02
By f_{cec}		(13, 5, 32)†	(6, 4, 40)†	(6, 5, 39)†	**(14, 14, 22)**	

† denotes a significant difference between the corresponding algorithm and UACOR$^+$-s by a Friedman test at the 0,05 α-level over the distribution of average errors of ACO$_\mathbb{R}$-c, DACO$_\mathbb{R}$-c, IACO$_\mathbb{R}$-Mtsls1-c and UACOR$^+$-c.

Table A.4: The average errors obtained by HDDE, Pro-JADE, Pro-SaDE, Pro-DEGL, ABC-MR and UACOR$^+$-c for for each CEC'05 function. The numbers in parenthesis at the bottom of the table represent the number of times an algorithm is better, equal or worse, respectively, compared to UACOR$^+$-c. Error values lower than 10^{-8} are approximated to 10^{-8}. The lowest average errors values are highlighted.

Dim	f_{cec}	HDDE	Pro-JADE	Pro-SaDE	Pro-DEGL	ABC-MR	UACOR$^+$-c
30	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	8.13E+00	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	2.31E+06	1.85E+04	2.28E+06	4.20E+04	2.20E+05	1.00E−08
	f_{cec4}	1.33E+02	1.00E−08	2.00E−05	1.00E−08	1.00E−08	1.14E+03
	f_{cec5}	7.66E+02	5.90E+01	5.51E+01	1.91E+01	6.02E+03	1.00E−08
	f_{cec6}	3.19E+01	1.89E+01	1.36E+00	1.20E+00	1.38E+02	1.00E−08
	f_{cec7}	4.70E+03	4.70E+03	4.70E+03	4.69E+03	1.49E−02	1.00E−08
	f_{cec8}	2.09E+01	2.09E+01	2.10E+01	2.09E+01	2.09E+01	2.00E+01
	f_{cec9}	3.91E+00	1.00E−08	1.00E−08	3.58E+01	6.60E+01	2.95E+01
	f_{cec10}	6.01E+01	8.18E+01	1.01E+02	5.18E+01	2.01E+02	2.91E+01
	f_{cec11}	2.57E+01	3.01E+01	3.37E+01	2.01E+01	3.56E+01	3.82E+00
	f_{cec12}	7.86E+03	2.63E+04	1.48E+03	2.35E+04	9.55E+04	2.21E+03
	f_{cec13}	2.04E+00	3.32E+00	2.95E+00	3.40E+00	1.07E+01	2.26E+00
	f_{cec14}	1.27E+01	1.29E+01	1.31E+01	1.24E+01	1.88E−01	1.32E+01
	f_{cec15}	3.17E+02	3.71E+02	3.86E+02	3.53E+02	2.88E+02	1.92E+02
	f_{cec16}	8.58E+01	1.14E+02	6.97E+01	1.76E+02	3.06E+02	6.14E+01
	f_{cec17}	1.01E+02	1.45E+02	7.20E+01	1.60E+02	3.01E+02	2.87E+02
	f_{cec18}	9.03E+02	8.60E+02	8.56E+02	9.09E+02	8.12E+02	8.73E+02
	f_{cec19}	9.04E+02	8.90E+02	8.67E+02	9.10E+02	8.17E+02	8.83E+02
	f_{cec20}	9.04E+02	8.96E+02	8.52E+02	9.10E+02	8.23E+02	8.78E+02
	f_{cec21}	5.00E+02	5.06E+02	5.00E+02	6.79E+02	6.42E+02	5.00E+02
	f_{cec22}	8.76E+02	8.95E+02	9.09E+02	8.94E+02	9.04E+02	8.57E+02
	f_{cec23}	5.34E+02	5.00E+02	5.00E+02	6.77E+02	8.20E+02	5.34E+02
	f_{cec24}	2.00E+02	2.00E+02	2.00E+02	7.77E+02	2.01E+02	3.41E+02
	f_{cec25}	1.28E+03	1.67E+03	1.63E+03	1.64E+03	2.00E+02	2.63E+02
50	f_{cec1}	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
	f_{cec2}	3.30E+02	1.00E−08	7.40E−04	1.00E−08	1.00E−08	1.00E−08
	f_{cec3}	4.44E+06	4.30E+04	7.82E+05	1.93E+05	1.13E+06	1.00E−08
	f_{cec4}	2.94E+03	3.19E−01	6.64E+01	1.08E−01	3.82E+02	1.69E+04
	f_{cec5}	3.22E+03	1.83E+03	1.95E+03	2.23E+03	1.03E+04	1.00E−08
	f_{cec6}	5.74E+01	1.04E+01	1.15E+01	8.77E−01	2.47E+03	1.00E−08
	f_{cec7}	6.20E+03	6.20E+03	6.20E+03	6.20E+03	8.10E−01	1.00E−08
	f_{cec8}	2.11E+01	2.10E+01	2.11E+01	2.11E+01	2.11E+01	2.01E+01
	f_{cec9}	1.87E+01	2.77E+01	6.61E−01	7.94E+01	2.59E+02	8.18E+01
	f_{cec10}	1.13E+02	1.99E+02	6.23E+01	9.24E+01	4.58E+02	7.45E+01
	f_{cec11}	5.19E+01	6.03E+01	6.61E+01	6.14E+01	7.03E+01	1.07E+01
	f_{cec12}	3.84E+04	9.45E+04	7.34E+03	6.32E+04	8.88E+05	1.27E+04
	f_{cec13}	4.36E+00	9.14E+00	6.90E+00	5.41E+00	3.50E+01	4.76E+00
	f_{cec14}	2.23E+01	2.26E+01	2.28E+01	2.26E+01	2.32E+01	2.31E+01
	f_{cec15}	2.62E+02	3.80E+02	3.96E+02	3.44E+02	2.52E+02	1.74E+02
	f_{cec16}	1.05E+02	1.44E+02	4.85E+01	1.63E+02	3.39E+02	7.01E+01
	f_{cec17}	1.15E+02	1.92E+02	9.36E+01	1.94E+02	3.08E+02	3.17E+02
	f_{cec18}	9.17E+02	9.26E+02	9.05E+02	9.28E+02	9.85E+02	9.08E+02
	f_{cec19}	9.16E+02	9.32E+02	8.97E+02	9.28E+02	9.70E+02	8.76E+02
	f_{cec20}	9.16E+02	9.33E+02	9.11E+02	9.29E+02	9.70E+02	8.64E+02
	f_{cec21}	7.37E+02	5.00E+02	5.00E+02	9.50E+02	8.34E+02	5.00E+02
	f_{cec22}	9.01E+02	9.49E+02	9.60E+02	9.28E+02	8.75E+02	8.92E+02
	f_{cec23}	7.85E+02	5.00E+02	5.06E+02	9.35E+02	5.71E+02	5.41E+02
	f_{cec24}	2.00E+02	2.00E+02	2.00E+02	6.81E+02	2.01E+02	8.26E+02
	f_{cec25}	1.37E+03	1.71E+03	1.69E+03	1.67E+03	2.01E+02	3.53E+02
By f_{cec}		(12, 4, 34)†	(13, 5, 32)†	(20, 5, 25)†	(8, 4, 38)†	(12, 4, 34)†	

† denotes a significant difference between the corresponding algorithm and UACOR$^+$-c by a Friedman test at the 0,05 α-level over the distribution of average errors of HDDE, Pro-JADE, Pro-SaDE, Pro-DEGL, ABC-MR and UACOR$^+$-c.

Table A.5: Given are the average rank, the number of optimum thresholds reached, and the number of times the lowest average errors reached by each algorithm presented in Table A.4. In addition, we give the publication source for each reference algorithm.

Algorithms	Average Ranking	Num of Optima	Num of lowest average error values	Publication Sources
UACOR$^+$-c	**2.52**	**12**	**25**	
Pro-SaDE†	3.22	4	17	IEEE TEC, 2011
Pro-JADE†	3.54	6	11	IEEE TEC, 2011
HDDE†	3.54	2	8	IEEE TEC, 2011
Pro-DEGL†	3.89	5	6	IEEE TEC, 2011
ABC-MR†	4.29	5	12	Information Sciences, 2012

† denotes a significant difference between the corresponding algorithm and UACOR$^+$-c by a Friedman test at the 0,05 α-level over the distribution of average errors of HDDE, Pro-JADE, Pro-SaDE, Pro-DEGL, ABC-MR and UACOR$^+$-c.

Appendix B

Mathematical formulation of engineering problems

In Appendix B, we give an explicit definition of the engineering problems that we used for benchmark the $\text{ACO}_{\mathbf{MV}}$ and the $\text{CES}_{\mathbf{MV}}$ algorithms in Chapter 5.

Welded beam design problem case A

The mathematical formulation of the welded beam design problem is given in Table B.1. The schematic view of this problem is shown in Fig. B.1

Welded beam design problem case B

The welded beam design problem case B is a variation of case A. It is extended to include two types of welded joint configuration and four possible beam materials. The changed places are shown in Equation B.1 and Table B.2.

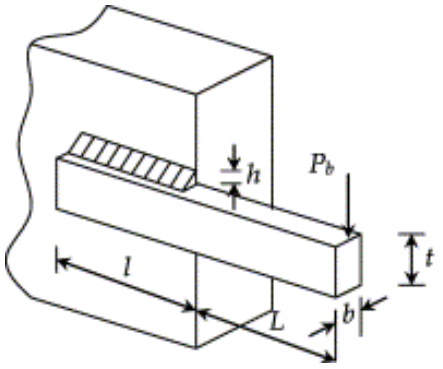


Figure B.1: Schematic view of welded beam design problem case A [Kayhan et al., 2010].

Table B.1: The mathematical formulation of welded beam design problem case A.

$\min f(\vec{x}) = 1.10471\, x_1^2 x_2 + 0.04811\, x_3 x_4\, (14 + x_2)$	
g_1	$\tau(\vec{x}) - \tau_{max} \leq 0$
g_2	$\sigma(\vec{x}) - \sigma_{max} \leq 0$
g_3	$x_1 - x_4 \leq 0$
g_4	$0.10471\, x_1^2 + 0.04811\, x_3 x_4\, (14 + x_2) - 5 \leq 0$
g_5	$0.125 - x_1 \leq 0$
g_6	$\delta(\vec{x}) - \delta_{max} \leq 0$
g_7	$P - P_c(\vec{x}) \leq 0$
g_8	$0.1 \leq x_1, x_4 \leq 2.0$
g_9	$0.1 \leq x_2, x_3 \leq 10.0$
where	$\tau(\vec{x}) = \sqrt{(\tau')^2 + 2\tau'\tau'' \frac{x_2}{2R} + (\tau'')^2}$
	$\tau' = \frac{P}{\sqrt{2} x_1 x_2}, \tau'' = \frac{MR}{J}, M = P(L + \frac{x_2}{2})$
	$R = \sqrt{\frac{x_2^2}{4} + (\frac{x_1 + x_3}{2})^2}$
	$J = 2\left\{\sqrt{2} x_1 x_2 \left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\}$
	$\sigma(\vec{x}) = \frac{6PL}{x_4 x_3^2}, \delta(\vec{x}) = \frac{4PL^3}{E x_3^3 x_4}$
	$P_c(\vec{x}) = \frac{4.013E\sqrt{\frac{x_3^2 x_4^6}{36}}}{L^2}\left(1 - \frac{x_3}{2L}\sqrt{\frac{E}{4G}}\right)$
	$P = 6\,000lb, L = 14in., E = 30 \times 10^6 psi, G = 12 \times 10^6 psi$
	$\tau_{max} = 1\,360psi, \sigma_{max} = 30\,000psi, \delta_{max} = 0.25in.$

$$
\begin{aligned}
&\min f(\vec{x}) = (1 + c_1)\, x_1^2 x_2 + c_2\, x_3 x_4\, (14 + x_2) \\
&\sigma(\vec{x}) - S \leq 0 \\
&J = 2\left\{\sqrt{2} x_1 x_2 \left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\},\ \textit{if } x_6 : \textit{twoside} \\
&J = 2\left\{\sqrt{2} x_1 \left[\frac{(x_1 + x_2 + x_3)^3}{12}\right]\right\},\ \textit{if } x_6 : \textit{fourside} \\
&\tau_{max} = 0.577 \cdot S
\end{aligned}
\tag{B.1}
$$

Table B.2: Material properties for the welded beam design problem case B

Methods x_5	$S(10^3 psi)$	$E(10^6 psi)$	$G(10^6 psi)$	c_1	c_2
Steel	30	30	12	0.1047	0.0481
Cast iron	8	14	6	0.0489	0.0224
Aluminum	5	10	4	0.5235	0.2405
Brass	8	16	6	0.5584	0.2566

Pressure vessel design problem

The pressure vessel design problem requires designing a pressure vessel consisting of a cylindrical body and two hemispherical heads such that the manufacturing cost is minimized subject to certain constraints. The schematic picture of the vessel is presented in Fig. B.2. 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

Table B.3: The mathematical formulation the cases (A, B, C and D) of the pressure vessel design problem.

No	Case A	Case B	Case C	Case D
$\min f = 0.6224\,T_s RL + 1.7781\,T_h R^2 + 3.1611\,T_s^2 L + 19.84\,T_s^2 R$				
g_1	$-T_s + 0.0193\,R \leq 0$			
g_2	$-T_h + 0.00954\,R \leq 0$			
g_3	$-\pi\,R^2 L - \frac{4}{3}\pi\,R^3 + 750 \cdot 1728 \leq 0$			
g_4	$L - 240 \leq 0$			
g_5	$1.1 \leq T_s \leq 12.5$	$1.125 \leq T_s \leq 12.5$	$1 \leq T_s \leq 12.5$	$0 \leq T_s \leq 100$
g_6	$0.6 \leq T_h \leq 12.5$	$0.625 \leq T_h \leq 12.5$		$0 \leq T_h \leq 100$
g_7	$0.0 \leq R \leq 240$			$10 \leq R \leq 200$
g_8	$0.0 \leq L \leq 240$			$10 \leq L \leq 200$

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 multiples of 0.0625 inch. The mathematical formulation of the four cases A, B, C and D is given in Table B.3.

Coil spring design problem

The problem consists in designing a helical compression spring that holds 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 Fig. B.3. 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 d is a discrete variable, whose possible values are given in Table B.4. The mathematical formulation is in Table B.5.

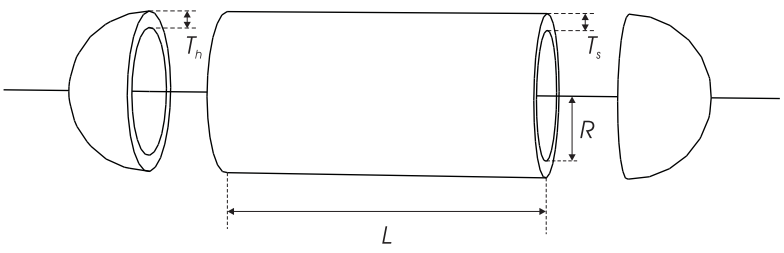


Figure B.2: Schematic view of the pressure vessel to be designed.

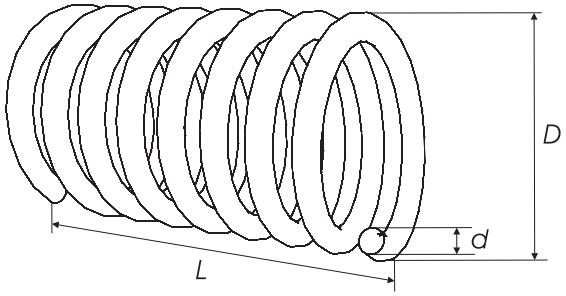


Figure B.3: Schematic view of the coil spring to be designed.

Table B.4: 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

Table B.5: The mathematical formulation for the coil spring design problem.

$\min f_c(N,D,d) = \frac{\pi^2 Dd^2(N+2)}{4}$	
	Constraint
g_1	$\frac{8C_f F_{\max} D}{\pi d^3} - S \leq 0$
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	$\sigma_w - \frac{F_{\max} - F_p}{K} \leq 0$
where	$C_f = \frac{4\frac{D}{d}-1}{4\frac{D}{d}-4} + \frac{0.615\,d}{D}$
	$K = \frac{Gd^4}{8ND^3}$
	$\sigma_p = \frac{F_p}{K}$
	$l_f = \frac{F_{\max}}{K} + 1.05(N+2)d$

Thermal insulation systems design problem

The schema of a thermal insulation system is shown in Fig. B.4. Such a thermal insulation system is characterized by the number of intercepts, the locations and temperatures of the intercepts, and the types of insulators allocated between each pair of neighboring intercepts. In the thermal insulation system, heat intercepts are used to minimize the heat flow from a hot to a cold surface. The heat is intercepted by imposing a cooling temperature T_i at locations x_i, $i = 1, 2, ..., n$.

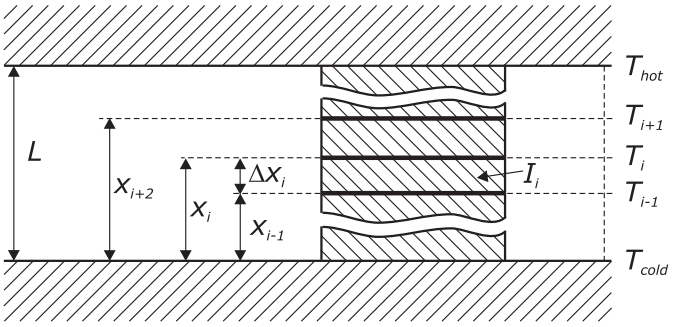


Figure B.4: Schematic view of the thermal insulation system.

The basic mathematical formulation of the classic model of thermal insulation systems is defined in Table B.6. The effective thermal conductivity k of all these insulators varies with the temperature and does so differently for different materials. Considering that the number of intercepts n is defined in advance, and based on the model presented (n=10), 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, there are $n+1$ categorical variables chosen from a set of $\mathbf{M}$ of available materials. The remaining $2n+2$ variables are continuous.

Table B.6: The mathematical formulation for the coil spring design problem.

$f(\mathbf{x}, \mathbf{T}) = \sum_{i=1}^{n} P_i$ $= \sum_{i=1}^{n} AC_i \left(\frac{T_{\text{hot}}}{T_i} - 1\right) \left(\frac{\int_{T_i}^{T_{i+1}} k dT}{\Delta x_i} - \frac{\int_{T_{i-1}}^{T^i} k dT}{\Delta x_{i-1}}\right)$	
	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	$\sum_{i=1}^{n+1} \Delta x_i = L$
where	$C = 2.5$ if $T \geq 71\,\text{K}$ $C = 4$ if $71\,\text{K} > T > 4.2\,\text{K}$ $C = 5$ if $T \leq 4.2\,\text{K}$

Appendix C

A note on the bound constraints handling for the CEC'05 benchmark set

The benchmark functions and some of the algorithms proposed for the special session on real parameter optimization of CEC'05 play an important role in the assessment of the state of the art in the continuous optimization field. In this note we first show that, if boundary constraints are not enforced, state-of-the-art algorithms produce on a majority of the CEC'05 benchmark functions infeasible best candidate solutions, even though the optima of 23 out of the 25 CEC'05 functions are within the specified bounds. This observation has important implications on algorithm comparisons. In fact, this note also draws the attention to the fact that authors may have drawn wrong conclusions from experiments using the CEC'05 problems.

C.1 Introduction

The special session on real parameter optimization of CEC'05 has played an important role in evolutionary computation and other affine fields for two reasons. First, it provided a set of 25 scalable benchmark functions that anyone can use to evaluate the performance of new algorithms. Those 25 functions have become a standard benchmark set that researchers use to compare algorithms. The central role that this benchmark function set currently plays is also illustrated by the more than 600 citations in google scholar (as of April 2013) to the original technical report that introduced the benchmark function set [Suganthan et al., 2005]. Second, it served to assess the state of the art in continuous optimization. In particular, the best performing algorithm of the special session, IPOP-CMA-ES [Auger and Hansen, 2005], is since then considered to be a representative of the state of the art in continuous optimization. Consequently, it is nowadays standard practice to

compare the results of a new algorithm to the published results of IPOP-CMA-ES.

In the definition of the CEC'05 benchmark functions it is stated that each component of the solution S must be a value in the closed interval $[A, B]$, with $A < B$. There are two exceptions, which are functions f_7 and f_{25}, where the given interval specifies only an initialization range, and not a boundary constraint. For the other 23 functions, their global optima are within the feasible search space area as defined by the bound constraints; on functions f_8 and f_{20}, the global optima are known to be on the bounds. We used the `C` implementation of IPOP-CMA-ES available from Hansen's website, `http://www.lri.fr/~hansen/cmaes_inmatlab.html`. This `C` version of IPOP-CMA-ES does not use an explicit bound constraint handling mechanism. When running this code (without bound constraint handling) on the CEC'05 benchmark functions, we noticed that on a majority of the benchmark functions the best solutions found do violate the bound constraints even though their global optima are known to be inside the bounds for 23 of the 25 functions. While it is known that this can happen on other functions, for example, Schwefel's sine root function [Schwefel, 1981] whose global optimum is outside the usual feasible search space defined by bound constraints, we were surprised by the high frequency with which this phenomenon occurs on the CEC'05 benchmark function set.

This observation raises the more general and critical issue of validity of many of the published results that rely on the CEC'05 benchmark set. In fact, most articles do not explicitly report whether a bound constraint handling mechanism was used, and if they do, many do not describe it. Perhaps more importantly, claims about algorithms with a statistically significantly better performance than IPOP-CMA-ES (e.g., [Molina et al., 2010a, Müller et al., 2009] may not be valid because the comparison that supports those claims may include algorithms that enforce bound constraints and algorithms that do not.

To show how misleading such a comparison can be, we report our experimental evidence on the impact of the handling of bound constraints. We evaluate two variants of the `C` version of IPOP-CMA-ES. The first is the version in which the bound constraints are never enforced (*ncb* for "never clamp bounds"; we refer to this version as IPOP-CMA-ES-*ncb* in what follows). The second is a version in which we introduce a mechanism to enforce bound constraints in the `C` code from Hansen's webpage. In particular, we clamp dimension by dimension a variable's value that is outside the variable's feasible domain to the closest boundary value; that is, if $x_i < A$ we set $x_i = A$ and if $x_i > B$ we set $x_i = B$ before evaluating these solutions and continuing with the algorithm execution (*acb* for "always

clamp bounds"; this version is referred to as IPOP-CMA-ES-*acb*). The same two variants where bound constraints are never enforced or always enforced by clamping infeasible solutions to the nearest solutions on the bounds are tested using a memetic algorithm, MA-LSCh-CMA [Molina et al., 2010a], which is a recent memetic algorithm that uses CMA-ES as a local search.

C.2 Experiments on enforcing bound constraints

In this first set of experiments, we examine the impact of bound handling on the performance of the `C` version of IPOP-CMA-ES. For the experiments on the CEC'05 benchmark functions, we followed the protocol described in [Suganthan et al., 2005], that is, we ran IPOP-CMA-ES using its default parameter settings 25 times on each function and recorded the evolution of the objective function value with respect to the number of function evaluations used. The maximum number of function evaluations was $10000 \cdot D$, where $D \in \{10, 30, 50\}$ is the dimensionality of a function. The algorithm stops when the maximum number of evaluations is reached or the error obtained is lower than 10^{-8}. Error values lower than this optimum threshold are considered equal to 10^{-8}, and these values therefore are clamped to 10^{-8} for consistency.

We compare IPOP-CMA-ES-*ncb* and IPOP-CMA-ES-*acb* in Table C.1.[1] The two-sided Wilcoxon matched-pairs signed-rank test at the 0.05 level of the error of first type was used to check for statistical differences on each function. On the majority of the functions, that is in 14 to 17 functions depending on the dimensionality, IPOP-CMA-ES-*ncb* obtains final solutions outside the bounds. In most of the cases where infeasible solutions are found, all 25 runs return final solutions outside the bounds. We have observed statistically significant differences between IPOP-CMA-ES-*ncb* and IPOP-CMA-ES-*acb* when the final solutions of IPOP-CMA-ES-*ncb* are outside the bounds. Surprisingly, while *a priori* we would expect that IPOP-CMA-ES-*ncb* gives worse results than IPOP-CMA-ES-*acb* on functions where the optima are known to be inside the bounds, IPOP-CMA-ES-*ncb* in fact outperforms IPOP-CMA-ES-*acb* in six functions (f_9, f_{12}, f_{18}, f_{19}, f_{20} and f_{22} for dimensions 30 and 50). In all these functions, except f_9, all solutions obtained by IPOP-CMA-ES-*ncb* are outside the bounds.

Analogously, Table C.2 shows the performance of *ncb* and *acb* for MA-LSCh-

[1] The results in Table C.1 and in Tables C.2 and C.3 are based on average errors. In the supplementary pages to this chapter at `http://iridia.ulb.ac.be/supp/IridiaSupp2011-013` additional tables are given that show the median results and more detailed information such as the best, 0.25 quartile, median, 0.75 quartile and worst error values for each function.

CMA (MA-LSCh-CMA is run using default parameter settings). Again, version *ncb* obtains many final solutions outside the bounds, for dimensions 30 and 50 this is the case on 18 and 19 functions, respectively.[2] Taking the 50 dimensional benchmark functions as an example, all functions for which MA-LSCh-CMA-*ncb* outperforms MA-LSCh-CMA-*acb* are cases in which all solutions obtained by MA-LSCh-CMA-*ncb* are outside the bounds (f_5, f_{11}, f_{12}, f_{15}, f_{18}, f_{19}, f_{20} and f_{22}).

C.3 The impact of bound handling on algorithm comparisons

We now focus on the comparison of the average errors between PS-CMA-ES [Müller et al., 2009], MA-LSCh-CMA [Molina et al., 2010a], IPOP-CMA-ES-*ncb* and IPOP-CMA-ES-05 in Table C.3. IPOP-CMA-ES-05 uses the Matlab version of CMA-ES and was used to generate the results for the CEC'05 benchmark functions presented in [Auger and Hansen, 2005]; it handles bound constraints by an approach based on penalty functions, which is described in [Hansen et al., 2009c]. PS-CMA-ES and MA-LSCh-CMA are examples of algorithms that have been reported to outperform IPOP-CMA-ES-05; they use CMA-ES as a local search operator inside a particle swarm optimization algorithm and a real-coded steady state genetic algorithm, respectively.

In fact, Table C.3 shows that PS-CMA-ES, MA-LSCh-CMA, but also IPOP-CMA-ES-*ncb*, are superior to IPOP-CMA-ES-05 on 30 and 50 dimensions in the sense that they find more often better average errors than IPOP-CMA-ES-05. However, there is an interesting pattern related to the fact whether IPOP-CMA-ES-*ncb* has the final solutions outside the bounds or not. Let us focus on the cases where IPOP-CMA-ES-*ncb* obtains all solutions outside the bounds and statistically significantly improves over IPOP-CMA-ES-05 (as indicated by the "<" symbol in Table C.3). In many such cases, PS-CMA-ES does obtain the same average errors (see, for example, functions f_{18}–f_{20} and f_{24} for both, 30 and 50 dimensions and function f_{22} for 50 dimensions), or very similar values (see, for example, functions f_{22} and f_{23} for 50 dimensions); such cases are underlined in Table C.3. A similar pattern also arises for the published results of the MA-LSCh-CMA algorithm. Interestingly, MA-LSCh-CMA checks bound constraints only for

[2]Note that for both, IPOP-CMA-ES-*ncb* and MA-LSCh-CMA-*ncb*, the fraction of functions for which at least once in 25 runs an infeasible solution is found, increases with the dimensionality of the functions. The probable reason for this effect is that for a solution to be infeasible it suffices that one single variable takes a value outside the boundary constraints—such an effect occurs more likely for higher dimensional functions.

the solutions generated by the steady-state GA part but not for solutions returned by the CMA-ES local search. After re-running the publicly available version of MA-LSCh-CMA, we found that it returns on several functions infeasible final solutions (as indicated by the symbols $^{\oslash}$ and $^{\odot}$ in Table C.3). This knowledge together with the similar pattern of the average errors puts at least serious doubts on the fact whether the average errors reported in [Müller et al., 2009] correspond all to solutions that are inside the bounds. This analysis shows that claims of superiority of one algorithm over another may in fact not be valid if the algorithms involved do not **all** enforce bound constraints.

C.4 Conclusions

In this note, we show that lack of enforcement of bound constraints greatly influences the feasibility of the solutions on the CEC'05 benchmark function suite, as it is observed in IPOP-CMA-ES and MA-LSCh-CMA. Without explicit enforcement of bound constrains, surprisingly often infeasible solutions are obtained on the CEC'05 benchmark functions; in many cases, these infeasible solutions are better than the best feasible solutions found if bound constraints are enforced even though it is known that on most functions the optimal solutions are within the bounds. This issue points toward a significant impact on CEC'05 benchmark functions for what concerns algorithm comparison, but it certainly may apply also to other benchmark sets [Hansen et al., 2009a, Herrera et al., 2010, Tang et al., 2007]. In particular, claims about superior performance of one algorithm over another may be erroneous due to the generation of infeasible solutions with respect to bound constraints. To avoid possible doubts about the feasibility of the solutions, we strongly recommend that in the future every paper that reports results on the CEC'05 benchmark function suite but also on other benchmark suites should (i) explicitly describe the used bound handling mechanism, (ii) explicitly check the feasibility of the final solutions[3], and (iii) present the final solutions at least in supplementary pages to the paper to avoid misinterpretations. All the solutions generated by the algorithms discussed in this note are available at `http://iridia.ulb.ac.be/supp/IridiaSupp2011-013`.

[3]We notice that many bound handling mechanisms use penalty approaches such as the one also used by [Hansen et al., 2009c]. However, it still is possible to obtain infeasible solutions if bound constraints are not explicitly enforced, no matter what penalty approach is used as the bound handling mechanism.

Table C.1: The comparison between IPOP-CMA-ES-*ncb* and IPOP-CMA-ES-*acb* over 25 independent runs for each of the 25 CEC'05 functions. 23 of these 25 functions actually have bound constraints; for functions f_7 and f_{25} only a bounded initialization range is specified. "$\oslash$" denotes that all 25 final solutions are outside the bounds. "$\odot$" denotes that some but not all of the 25 solutions are outside the bounds. Symbols $<$, $\approx$, and $>$ denote whether the performance of IPOP-CMA-ES-*ncb* is statistically better, indifferent, or worse than that of IPOP-CMA-ES-*acb* according to a two-sided Wilcoxon matched-pairs signed-rank test at the 0.05 α-level. The average errors that correspond to a statistically better result are highlighted. The numbers in parenthesis at the bottom of the table represent the frequency of $<$, $\approx$, and $>$, respectively.

	10 dimensions		30 dimensions		50 dimensions	
f_{cec}	*ncb*	*acb*	*ncb*	*acb*	*ncb*	*acb*
f_1	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08
f_2	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08
f_3	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08
f_4	1.00E−08 $\approx$	1.00E−08	2.44E+03 $^{\odot}$ $\approx$	6.58E+02	1.32E+05 $^{\odot}$ $>$	1.43E+04
f_5	1.00E−08 $^{\oslash}$ $\approx$	1.00E−08	2.30E+01 $^{\oslash}$ $>$	1.00E−08	7.91E+02 $^{\oslash}$ $>$	7.41E−02
f_6	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08
f_7†	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08	1.00E−08 $\approx$	1.00E−08
f_8	2.01E+01 $^{\oslash}$ $\approx$	2.00E+01	2.07E+01 $^{\oslash}$ $>$	2.04E+01	2.11E+01 $^{\oslash}$ $>$	2.09E+01
f_9	1.59E−01 $\approx$	1.59E−01	1.01E+00 $<$	1.87E+00	1.12E+00 $^{\odot}$ $<$	4.36E+00
f_{10}	1.19E−01 $\approx$	3.18E−01	1.37E+00 $\approx$	1.44E+00	2.36E+00 $\approx$	2.89E+00
f_{11}	6.44E−01 $^{\oslash}$ $>$	1.00E−08	6.36E+00 $^{\oslash}$ $>$	7.17E−02	1.49E+01 $^{\oslash}$ $>$	9.94E−02
f_{12}	6.77E+01 $^{\oslash}$ $<$	4.07E+03	1.38E+03 $^{\oslash}$ $<$	1.19E+04	7.38E+03 $^{\oslash}$ $<$	4.25E+04
f_{13}	6.78E−01 $\approx$	6.49E−01	2.47E+00 $\approx$	2.63E+00	4.31E+00 $\approx$	4.44E+00
f_{14}	2.61E+00 $^{\odot}$ $>$	1.96E+00	1.28E+01 $^{\oslash}$ $\approx$	1.26E+01	2.34E+01 $^{\oslash}$ $>$	2.28E+01
f_{15}	2.00E+02 $^{\oslash}$ $\approx$	2.15E+02	2.01E+02 $^{\oslash}$ $>$	2.00E+02	2.01E+02 $^{\oslash}$ $>$	2.00E+02
f_{16}	9.02E+01 $\approx$	9.04E+01	7.95E+01 $^{\odot}$ $>$	1.48E+01	1.36E+02 $^{\oslash}$ $>$	1.10E+01
f_{17}	1.33E+02 $^{\odot}$ $\approx$	1.17E+02	4.31E+02 $^{\oslash}$ $>$	2.52E+02	7.69E+02 $^{\oslash}$ $>$	1.91E+02
f_{18}	7.48E+02 $^{\odot}$ $>$	3.16E+02	8.16E+02 $^{\oslash}$ $<$	9.04E+02	8.36E+02 $^{\oslash}$ $<$	9.13E+02
f_{19}	7.75E+02 $^{\odot}$ $>$	3.20E+02	8.16E+02 $^{\oslash}$ $<$	9.04E+02	8.36E+02 $^{\oslash}$ $<$	9.13E+02
f_{20}	7.62E+02 $^{\odot}$ $>$	3.20E+02	8.16E+02 $^{\oslash}$ $<$	9.04E+02	8.36E+02 $^{\oslash}$ $<$	9.15E+02
f_{21}	1.06E+03 $^{\oslash}$ $>$	5.00E+02	8.57E+02 $^{\oslash}$ $>$	5.00E+02	7.15E+02 $^{\oslash}$ $\approx$	6.64E+02
f_{22}	6.38E+02 $^{\oslash}$ $<$	7.28E+02	5.98E+02 $^{\oslash}$ $<$	8.10E+02	5.00E+02 $^{\oslash}$ $<$	8.19E+02
f_{23}	1.09E+03 $^{\oslash}$ $>$	5.86E+02	8.69E+02 $^{\oslash}$ $>$	5.34E+02	7.27E+02 $^{\oslash}$ $\approx$	6.97E+02
f_{24}	4.05E+02 $^{\oslash}$ $>$	2.33E+02	2.10E+02 $^{\oslash}$ $>$	2.00E+02	2.14E+02 $^{\oslash}$ $>$	2.00E+02
f_{25}†	4.34E+02 $\approx$	4.34E+02	2.10E+02 $\approx$	2.10E+02	2.14E+02 $\approx$	2.14E+02
	f_1-f_{25} ($<$, $\approx$, $>$): (2, 15, 8)		f_1-f_{25}($<$, $\approx$, $>$): (6, 10, 9)		f_1-f_{25}($<$, $\approx$, $>$): (6, 10, 9)	
	f_7 *or* f_{25} ($<$, $\approx$, $>$): (2, 13, 8)		f_7 *or* f_{25} ($<$, $\approx$, $>$): (6, 8, 9)		f_7 *or* f_{25} ($<$, $\approx$, $>$): (6, 8, 9)	
	$<$ *or* $>$: 10/23 (43%)		$<$ *or* $>$: 15/23 (65%)		$<$ *or* $>$: 15/23 (65%)	
	functions $\oslash$ or $\odot$: 14/23 (61%)		functions $\oslash$ or $\odot$: 16/23 (70%)		functions $\oslash$ or $\odot$: 17/23 (74%)	

† denotes that the specialized initialization ranges are applied instead of bound constraints according to CEC'05's protocol.

Table C.2: The comparison between MA-LSCh-CMA-*ncb* and MA-LSCh-CMA-*acb* over 25 independent runs for each of the 25 CEC'05 functions. For an explanation of the symbols and their interpretation we refer to the caption of Table C.1 (replacing IPOP-CMA-ES by MA-LSCh-CMA where relevant).

	10 dimensions			30 dimensions			50 dimensions		
f_{cec}	*ncb*		*acb*	*ncb*		*acb*	*ncb*		*acb*
f_1	1.00E−08	$\approx$	1.00E−08	1.00E−08	$\approx$	1.00E−08	1.00E−08	$\approx$	1.00E−08
f_2	1.00E−08	$\approx$	1.00E−08	2.51E−08	$\approx$	1.00E−08	8.99E−01	$\approx$	3.06E−02
f_3	3.68E+02	$>$	1.00E−08	4.41E+03 $\odot$	$\approx$	2.75E+04	8.11E+04 $\odot$	$>$	3.21E+04
f_4	1.00E−08	$\approx$	5.54E−03	1.28E+02	$<$	3.02E+02	5.38E+03 $\odot$	$>$	3.23E+03
f_5	7.78E+01 $\oslash$	$>$	6.75E−07	6.12E+02 $\oslash$	$<$	1.26E+03	2.08E+03 $\oslash$	$<$	2.69E+03
f_6	1.00E−08	$\approx$	3.19E−01	2.31E+02 $\odot$	$>$	1.12E+00	5.58E+02 $\odot$	$>$	4.10E+00
f_7†	1.65E−01	$\approx$	1.43E−01	1.57E−02	$\approx$	1.75E−02	4.23E−03	$\approx$	5.40E−03
f_8	2.00E+01 $\oslash$	$\approx$	2.00E+01	2.00E+01 $\oslash$	$\approx$	2.00E+01	2.00E+01 $\oslash$	$\approx$	2.00E+01
f_9	1.00E−08	$\approx$	1.00E−08	1.00E−08	$\approx$	1.00E−08	1.00E−08	$\approx$	1.00E−08
f_{10}	3.14E+00	$\approx$	2.67E+00	2.00E+01 $\odot$	$\approx$	2.25E+01	4.80E+01 $\odot$	$\approx$	5.01E+01
f_{11}	4.53E+00 $\oslash$	$>$	2.43E+00	2.20E+01 $\oslash$	$\approx$	2.15E+01	3.95E+01 $\oslash$	$<$	4.13E+01
f_{12}	2.95E+02 $\odot$	$\approx$	1.14E+02	7.52E+02 $\oslash$	$<$	1.67E+03	4.56E+03 $\oslash$	$<$	1.39E+04
f_{13}	5.03E−01	$\approx$	5.45E−01	2.04E+00	$\approx$	2.03E+00	3.67E+00	$>$	3.15E+00
f_{14}	2.87E+00 $\odot$	$>$	2.25E+00	1.32E+01 $\oslash$	$>$	1.25E+01	2.30E+01 $\oslash$	$>$	2.22E+01
f_{15}	2.27E+02 $\odot$	$\approx$	2.24E+02	2.59E+02 $\oslash$	$<$	3.00E+02	2.29E+02 $\oslash$	$<$	3.72E+02
f_{16}	9.45E+01 $\odot$	$\approx$	9.18E+01	1.06E+02 $\odot$	$\approx$	1.26E+02	5.91E+01 $\odot$	$>$	6.90E+01
f_{17}	1.04E+02	$\approx$	1.01E+02	1.66E+02 $\odot$	$\approx$	1.83E+02	1.41E+02 $\oslash$	$\approx$	1.47E+02
f_{18}	8.20E+02 $\odot$	$<$	8.84E+02	8.22E+02 $\oslash$	$<$	8.98E+02	8.47E+02 $\oslash$	$<$	9.41E+02
f_{19}	8.17E+02 $\odot$	$\approx$	8.78E+02	8.22E+02 $\oslash$	$<$	9.01E+02	8.48E+02 $\oslash$	$<$	9.38E+02
f_{20}	7.69E+02 $\odot$	$\approx$	8.63E+02	8.23E+02 $\oslash$	$<$	8.96E+02	8.48E+02 $\oslash$	$<$	9.28E+02
f_{21}	8.57E+02 $\odot$	$\approx$	7.94E+02	8.47E+02 $\oslash$	$>$	5.12E+02	7.23E+02 $\oslash$	$>$	5.00E+02
f_{22}	7.63E+02 $\odot$	$>$	7.53E+02	5.34E+02 $\oslash$	$<$	8.80E+02	5.00E+02 $\oslash$	$<$	9.14E+02
f_{23}	8.74E+02	$\approx$	8.88E+02	8.40E+02 $\oslash$	$>$	5.34E+02	7.26E+02 $\oslash$	$>$	5.39E+02
f_{24}	3.94E+02 $\oslash$	$>$	2.28E+02	2.14E+02 $\oslash$	$>$	2.00E+02	2.21E+02 $\oslash$	$>$	2.00E+02
f_{25}†	4.88E+02	$\approx$	4.55E+02	2.13E+02	$\approx$	2.14E+02	2.21E+02	$\approx$	2.21E+02
	f_1-f_{25} ($<$, $\approx$, $>$): (1, 18, 6)			f_1-f_{25}($<$, $\approx$, $>$): (8, 12, 5)			f_1-f_{25}($<$, $\approx$, $>$): (8, 8, 9)		
	$\overline{f_7 \text{ or } f_{25}}$ ($<$, $\approx$, $>$): (1, 16, 6)			$\overline{f_7 \text{ or } f_{25}}$ ($<$, $\approx$, $>$): (8, 10, 5)			$\overline{f_7 \text{ or } f_{25}}$ ($<$, $\approx$, $>$): (8, 6, 9)		
	$<$ *or* $>$: 7/23 (30%)			$<$ *or* $>$: 13/23 (57%)			$<$ *or* $>$: 17/23 (74%)		
	functions $\oslash$ or $\odot$: 13/23 (57%)			functions $\oslash$ or $\odot$: 18/23 (79%)			functions $\oslash$ or $\odot$: 19/23 (83%)		

† denotes that the specialized initialization ranges are applied instead of bound constraints according to CEC'05's protocal.

Table C.3: The average errors obtained by PS-CMA-ES, MA-LSCh-CMA (MA), IPOP-CMA-ES-*ncb* (IPOP-*ncb*) and IPOP-CMA-ES-05 (IPOP-05) over 25 independent runs for each CEC'05 function. The numbers in parenthesis represent the number of times an algorithm is better, equal or worse, respectively, compared to IPOP-CMA-ES-05. Error values lower than 10^{-8} are approximated to 10^{-8}. The underlined values indicate that the corresponding average error values of PS-CMA-ES (or MA-LSCh-CMA) are the same or very close to the infeasible average error values obtained by IPOP-CMA-ES-*ncb*.

	30 dimensions				50 dimensions			
f_{cec}	PS-CMA-ES	MA	IPOP-*ncb*	IPOP-05	PS-CMA-ES	MA	IPOP-*ncb*	IPOP-05
f_1	1.00E−08	———	1.00E−08	1.00E−08	1.00E−08	———	1.00E−08	1.00E−08
f_2	1.00E−08	———	1.00E−08	1.00E−08	9.79E−04	———	1.00E−08	1.00E−08
f_3	8.00E+04	———	1.00E−08	1.00E−08	3.28E+05	———	1.00E−08	1.00E−08
f_4	8.47E−04	———	2.44E+03 $^{\odot}$ <	1.11E+04	1.58E+03	———	1.32E+05 $^{\odot}$ <	4.68E+05
f_5	3.98E+02	———	2.30E+01 $^{\oslash}$ >	1.00E−08	1.18E+03	———	7.91E+02 $^{\oslash}$ >	2.85E+00
f_6	1.35E+01	1.19E+01	1.00E−08	1.00E−08	2.98E+01	6.58E+01	1.00E−08	1.00E−08
f_7	1.00E−08	8.87E−04	1.00E−08	1.00E−08	1.00E−08	2.37E−03	1.00E−08	1.00E−08
f_8	2.10E+01	2.03E+01	2.07E+01 $^{\oslash}$ >	2.01E+01	2.11E+01	2.05E+01	2.11E+01 $^{\oslash}$ >	2.01E+01
f_9	1.00E−08	1.00E−08	1.01E+00	9.38E−01	1.00E−08	1.00E−08	1.12E+00 $^{\odot}$ <	1.39E+00
f_{10}	1.00E−08	1.84E+01	1.37E+00	1.65E+00	1.00E−08	3.75E+01	2.36E+00	1.72E+00
f_{11}	3.91E+00	4.35E+00	6.36E+00 $^{\oslash}$ >	5.48E+00	1.22E+01	1.08E+01	1.49E+01 $^{\oslash}$ >	1.17E+01
f_{12}	7.89E+01	7.69E+02 $^{\odot}$	1.38E+03 $^{\oslash}$ <	4.43E+04	2.36E+03	2.76E+03 $^{\odot}$	7.38E+03 $^{\oslash}$ <	2.27E+05
f_{13}	2.11E+00	2.34E+00	2.47E+00	2.49E+00	4.00E+00	3.51E+00	4.31E+00	4.59E+00
f_{14}	1.29E+01	1.27E+01	1.28E+01 $^{\oslash}$ <	1.29E+01	2.25E+01	2.23E+01	2.34E+01 $^{\oslash}$ >	2.29E+01
f_{15}	2.10E+02	3.08E+02 $^{\oslash}$	2.01E+02 $^{\oslash}$ <	2.08E+02	2.64E+02	2.88E+02 $^{\oslash}$	2.01E+02 $^{\oslash}$ <	2.04E+02
f_{16}	2.61E+01	1.36E+02 $^{\oslash}$	7.95E+01 $^{\odot}$ >	3.50E+01	2.27E+01	6.40E+01 $^{\oslash}$	1.36E+02 $^{\oslash}$ >	3.09E+01
f_{17}	5.17E+01	1.35E+02 $^{\oslash}$	4.31E+02 $^{\oslash}$ >	2.91E+02	6.16E+01	8.32E+01 $^{\oslash}$	7.69E+02 $^{\oslash}$ >	2.34E+02
f_{18}	8.16E+02	8.16E+02 $^{\oslash}$	8.16E+02 $^{\oslash}$ <	9.04E+02	8.36E+02	8.45E+02 $^{\oslash}$	8.36E+02 $^{\oslash}$ <	9.13E+02
f_{19}	8.16E+02	8.16E+02 $^{\oslash}$	8.16E+02 $^{\oslash}$ <	9.04E+02	8.36E+02	8.45E+02 $^{\oslash}$	8.36E+02 $^{\oslash}$ <	9.12E+02
f_{20}	8.16E+02	8.16E+02 $^{\oslash}$	8.16E+02 $^{\oslash}$ <	9.04E+02	8.36E+02	8.41E+02 $^{\oslash}$	8.36E+02 $^{\oslash}$ <	9.12E+02
f_{21}	7.11E+02	5.12E+02 $^{\oslash}$	8.57E+02 $^{\oslash}$ >	5.00E+02	7.18E+02	5.45E+02 $^{\oslash}$	7.15E+02 $^{\oslash}$ <	1.00E+03
f_{22}	5.00E+02	5.26E+02 $^{\oslash}$	5.98E+02 $^{\oslash}$ <	8.03E+02	5.00E+02	5.00E+02 $^{\oslash}$	5.00E+02 $^{\oslash}$ <	8.05E+02
f_{23}	7.99E+02	5.34E+02 $^{\oslash}$	8.69E+02 $^{\oslash}$ >	5.34E+02	7.24E+02	5.81E+02 $^{\oslash}$	7.27E+02 $^{\oslash}$ <	1.01E+03
f_{24}	2.10E+02	2.00E+02 $^{\oslash}$	2.10E+02 $^{\oslash}$ <	9.10E+02	2.14E+02	2.00E+02 $^{\oslash}$	2.14E+02 $^{\oslash}$ <	9.55E+02
f_{25}	2.10E+02	2.11E+02	2.10E+02	2.11E+02	2.14E+02	5.81E+02	2.14E+02	2.15E+02
	(14, 4, 7)	(11, 2, 7)	(12, 5, 8)		(16, 2, 7)†	(13, 0, 7)‡	(13, 5, 7)	

$^{\oslash}$ denotes that all 25 solutions of IPOP-CMA-ES-*ncb* are outside the bounds. $^{\odot}$ denotes some of the 25 solutions of IPOP-CMA-ES-*ncb* are outside the bounds.

† (‡) denotes there is a significant difference over the distribution of average errors between PS-CMA-ES (MA-LSCh-CMA) and IPOP-CMA-ES-05 according to a two-sided Wilcoxon matched-pairs signed-rank test at the 0,05 (0.1) α-level.

Appendix D

Computational results for an automatically tuned IPOP-CMA-ES on the CEC'05 benchmark set

In this chapter, we apply an automatic algorithm configuration tool to improve the performance of the CMA-ES algorithm with increasing population size (iCMA-ES), the best performing algorithm on the CEC'05 benchmark set for continuous function optimization. In particular, we consider a separation between tuning and test sets and, thus, tune iCMA-ES on a different set of functions than the ones of the CEC'05 benchmark set. Our experimental results show that the tuned iCMA-ES improves significantly over the default version of iCMA-ES. Furthermore, we provide some further analyses on the impact of the modified parameter settings on iCMA-ES performance and a comparison to recent results of algorithms that use CMA-ES as a subordinate local search.

D.1 Introduction

The special session on real parameter optimization of CEC'05 initiated a series of research efforts on benchmarking continuous optimizers and the development of new, improved continuous optimization algorithms. Two noteworthy results of this session are the establishment of a benchmark set of 25 hard benchmark functions and the establishment of CMA-ES with increasing population size (iCMA-ES) [Auger and Hansen, 2005] as the state-of-the-art continuous optimizer at least for what concerns the field of nature-inspired computation in the widest sense.

Here, we explore whether we can improve iCMA-ES's performance on the CEC'05 benchmark set by further fine-tuning iCMA-ES using automatic algorithm configuration tools. In fact, iCMA-ES has a number of parameters and hidden constants in its code that make it a parameterized algorithm. Although its designers have spent a considerable effort in the design choices and certainly also in the

definition of its parameters, over the last few years evidence has arisen that many algorithms' performance can be improved by considering automatic algorithm configuration and tuning tools [Adenso-Diaz and Laguna, 2006, Balaprakash et al., 2007, Bartz-Beielstein, 2006, Birattari et al., 2002, Hutter et al., 2007, 2009a,b, Nannen and Eiben, 2007]. It is therefore a natural question to ask whether and by how much the performance of iCMA-ES could be further improved by such tools. Note that the answer to this question has also implications on methodological aspects in algorithm development. When trying to improve over an algorithm such as iCMA-ES, the design and tuning process of a new algorithm often starts by some new idea that is then iteratively refined manually until better performance on the considered benchmark set is obtained. One such idea is to embed CMA-ES as a local search into other algorithms. In fact, various authors have followed this path and have reported positive results, claiming better performance than iCMA-ES on the CEC'05 benchmark set [Molina et al., 2010a, Müller et al., 2009]. As an alternative approach, it is reasonable to simply try to improve directly iCMA-ES by fine-tuning it further. Hence, the question arises as to how iCMA-ES would perform against these "improved" algorithms if additional effort is put directly in iCMA-ES instead of the design of "new" algorithms. In this chapter, we also try to shed some light on this issue.

The present chapter is not the first to try to further tune CMA-ES using automatic algorithm configuration tools. CMA-ES was used by Hutter et al. [2009a] as a benchmark algorithm to be tuned for evaluating SPO$^+$, their improved variant of SPO [Bartz-Beielstein, 2006]. Following earlier work on SPO, they tuned CMA-ES only on individual functions, thus, in this sense "overtuning" CMA-ES on individual functions. (One has to remark, however, that the interest of Hutter et al. [2009a] was to evaluate SPO and the improved variant SPO$^+$ rather than proposing a new, generally improved parameter setting for CMA-ES.) Another attempt of tuning CMA-ES was made by Smit and Eiben in the paper "Beating the World Champion Evolutionary Algorithm via REVAC Tuning" [Smit and Eiben, 2010]. However, rather than the full version of CMA-ES, which is this "world champion evolutionary algorithm," in their study they use a reduced version that has limitations on rotated functions (for details, see Section 2). They reported significant improvements of their tuned algorithm over the default settings across the full range of functions of the CEC'05 benchmark set. From a tuning perspective, it should be mentioned that they tuned their algorithm on the whole set of the CEC'05 benchmark functions. For the tuning, they allowed the CMA-ES variant they used on each 10 dimensional function a maximum of 100 000 function eval-

uations; then they were running the tests with the tuned algorithm on the same functions for 1 000 000 function evaluations.

In this chapter, we tune iCMA-ES on a set of functions that has no overlap with the functions of the CEC'05 benchmark set. In this sense, we try to avoid a bias of the results obtained due to potentially overtuning [Birattari, 2009] the algorithm on the same benchmark functions as those on which the algorithm is tested. As such, this gives a better assessment of the potential for what concerns the tuning of continuous optimizers as we have a separation between tuning and test set. Note that such a separation is standard when studying tuning algorithms for combinatorial problems [Birattari, 2009, Birattari et al., 2002, Hutter et al., 2009b]. This separation of tuning and test sets for continuous functions is also different from our own previous applications, where we have tuned algorithms on small dimensional functions and later tested them on (much) larger dimensional variants of the same functions [Liao et al., 2011c]. In this latter case, the training and testing functions differ only in their dimensionality, which may potentially lead to some biases in the tuning.

As the tuning set, we consider small dimensional benchmark functions from the recent special issue of the Soft Computing journal [Herrera et al., 2010, Lozano et al., 2011] on large-scale function optimization. This SOCO benchmark set contains 19 functions whose dimension is freely choosable. Four of these functions are the same as in the CEC'05 benchmark set, so we removed them from the tuning set. As tuner, we apply the irace software [López-Ibáñez et al., 2011] to automatically tune seven parameters of iCMA-ES on the 10 dimensional SOCO benchmark functions (we refer to this tuned version of iCMA-ES as iCMA-ES-tsc). Then, we benchmark iCMA-ES-tsc on the whole CEC'05 benchmark function suite for 10, 30 and 50 dimensions. The experimental results show that iCMA-ES-tsc improves over the default parameter setting of iCMA-ES (called iCMA-ES-dp), and, maybe surprisingly, also is competitive or even improves over a version of iCMA-ES that we have tuned on the 10 dimensional CEC'05 benchmark set (we refer to this tuned version of iCMA-ES as iCMA-ES-tcec). We also compare iCMA-ES-tsc with MA-LSch-CMA [Molina et al., 2010a] and PS-CMA-ES [Müller et al., 2009], two state-of-the-art algorithms based on CMA-ES on CEC'05 benchmark function suite. Finally, we explore different possible choices of the tuning setup and, in particular, the choice of different sets of tuning functions.

Table D.1: Parameters that have been considered for tuning. Given are the default values of the parameters and the continuous range we considered for tuning. The last two columns give for each set of tuning instances the found algorithm configurations.

Parameters	Formulas	Factor	Default Values	Range	Tuned configurations	
					f_{cec}*	f_{soco}*
Pop size (λ)	$4 + \lfloor a \ln(D) \rfloor$	a	3	[1, 10]	7.315	9.600
Parent size (μ)	$\lfloor \lambda/b \rfloor$	b	2	[1, 5]	3.776	1.452
Init step size ($\sigma^{(0)}$)	$c(B - A)$	c	0.5	(0,1)	0.8297	0.6034
IPOP factor (d)	d	d	2	[1, 4]	2.030	3.292
stopTolFun	10^e	e	-12	[-20, -6]	-8.104	-8.854
stopTolFunHist	10^f	f	-20	[-20, -6]	-6.688	-9.683
stopTolX	10^g	g	-12	[-20, -6]	-13.85	-12.55

D.2 Parameterized iCMA-ES

iCMA-ES [Auger and Hansen, 2005] is a variant of the CMA-ES algorithm [Hansen and Ostermeier, 1996, 2001, Hansen et al., 2003] that uses a restart schema coupled with an increasing population size. The main details of the iCMA-ES algorithm have already been described in Section 2.3.2 and we refer the reader to the details described there. In particular, CMA-ES adapts the full covariance matrix of a normal search distribution. Sep-CMA-ES [Ros, 2009, Ros and Hansen, 2008] is a modification of CMA-ES with lower time complexity that instead of the full covariance matrix uses a diagonal matrix (that is, the covariances are assumed to be zero); in a sense, in Sep-CMA-ES the step size for each variable is adapted independently of the other variables. Sep-CMA-ES is also the variant that was used in the paper by Smit and Eiben [2010], which was mentioned in the introduction.

For tuning iCMA-ES, we considered seven parameters related to the above mentioned default settings. The parameters are given in Table D.1. The first four parameters are actually used in a formula to compute some internal parameters of iCMA-ES and the remaining three are used to define the termination of CMA-ES. Note that if a run of iCMA-ES is terminated, CMA-ES is restarted with an increased population size λ. For the increase of the population size, we here introduce a parameter d we call IPOP factor. The first five columns of Table D.1 give the parameters we use, the formula where they are used, their default values and the range that we considered for tuning. The remaining two columns are explained later.

D.3 Experimental setup and tuning

We used the C version of iCMA-ES (last modification date 10/16/10) from Hansen's webpage `https://www.lri.fr/~hansen/cmaes_inmatlab.html`. We modified the code to handle bound constraints by clamping the variable values outside the bounds on the nearest bound value. (The issues about the effects of enforcing and ignoring bound constraints have been addressed by Liao et al. [2011a]. Our test-suite consists of 25 CEC'05 benchmark functions (functions labeled as $f_{cec}*$) of dimensions $n \in \{10, 30, 50\}$. The training instances of iCMA-ES-tsc and iCMA-ES-tcec involve the 10-dimensional SOCO and CEC'05 benchmark functions, respectively. The SOCO and CEC'05 benchmark sets have four same functions (identical except for the shift vectors for moving the known optimum solution) and therefore we have removed these four functions from the SOCO benchmark set that we used as training set for tuning. In particular, we eliminated the four SOCO functions f_{soco1}, f_{soco3}, f_{soco4} and f_{soco8}, which are the same as the CEC'05 functions f_{cec1}, f_{cec6}, f_{cec9} and f_{cec2}, respectively.

The CEC'05 and SOCO benchmark functions are listed in Table 2.1 and 2.2. The two benchmark sets have common characteristics such as unimodality, multimodality, separability. Several of the functions in the two benchmark sets are also defined as compositions of other two functions; we refer to these also as hybrid functions in what follows. A major difference between the two benchmark sets is that in the CEC'05 benchmark 16 of the 25 functions are rotated functions, while all SOCO benchmark functions are unrotated.[1] For a more detailed explanation of the respective benchmark sets we refer to their original description [Herrera et al., 2010, Suganthan et al., 2005]; for a more recent intent to develop specific, more low-level function features for their classification, we refer to Mersmann et al. [2011]. We followed the protocol described in Suganthan et al. [2005] for the CEC'05 test-suite, that is, the maximum number of function evaluations was $10\,000 \times D$ where $D \in \{10, 30, 50\}$ is the dimensionality of a function when using them as test set (or as training set in the case of iCMA-ES-tcec). The investigated algorithms were run 25 times on each function. We report error values defined as $f(S) - f(S^*)$, where S is a candidate solution and S^* is the optimal solution. Error values lower

[1] Recall that "rotational invariance" is an important feature of iCMA-ES. This feature of iCMA-ES means that its performance is not negatively affected by a rotation of a function with respect to the coordinate system. From the perspective of parameter tuning, this is an important property since it implies that we should be able to tune iCMA-ES on unrotated functions of the SOCO benchmark set. In fact, in the experimental part we consider as a control experiment also tuning iCMA-ES directly on the CEC'05 benchmark set with the rotated functions.

than 10^{-8} are clamped to 10^{-8}, which is the zero threshold defined in the CEC'05 protocol [Suganthan et al., 2005]. Our analysis considers the median errors, mean errors and the solution quality distribution for each function.

For tuning the parameters of iCMA-ES, we employ Iterated F-Race [Birattari et al., 2010], a racing algorithm for algorithm configuration that is included in the irace package [López-Ibáñez et al., 2011]. The performance measure is the fitness error value of each instance. In the automatic parameter tuning process, the maximum budget is set to 5 000 runs of iCMA-ES. The setting of Iterated F-Race we used is the default [López-Ibáñez et al., 2011]. The input to Iterated F-Race are the ranges for each parameter, which are given in Table D.1, and a set of training instances. When using the SOCO benchmark set of tuning, the 10 dimensional versions of f_{soco1}-f_{soco19} (except f_{soco1}, f_{soco3}, f_{soco4} and f_{soco8}) were sampled as training instances in a random order and the number of function evaluations of each run is set equal to $5\,000 \times D$ ($D = 10$). When using the CEC'05 benchmark set of tuning in some control experiments, the 10 dimensional variants of f_{cec1}-f_{cec25} were sampled as training instances in a random order and the number of function evaluations of each run is equal to $10\,000 \times D$ ($D = 10$). The differences of the lengths of iCMA-ES runs on the SOCO and CEC'05 benchmark sets are due to the different termination criteria used in the definition of these benchmark sets. The default and tuned settings of iCMA-ES' parameters are presented in Table D.1.

Comparing the tuned parameter settings to the default settings, maybe the most noteworthy difference is that the tuned settings imply a more explorative search behavior. This is true least for the initial phases of the iCMA-ES search at the algorithm start and after each restart. This more explorative search behavior is due to the larger initial population size (through parameter a), a larger initial step size and a larger factor for the increase of the population size at restarts (at least for configuration iCMA-ES-tsc, which, as we will see later, is the best performing one). The performance of iCMA-ES-dp and iCMA-ES-tsc will be compared in Section D.4.1.

The significance of the differences of the algorithmic variants is assessed using statistical tests in two ways. First, on an instance level, we use a two-sided Wilcoxon signed-rank test at the 0.05 α-level to check whether the performance of two algorithms is statistically significantly different. Recall that each algorithm is run 25 independent times on each benchmark function. Second, across all benchmark functions, we apply a two-sided Wilcoxon matched-pairs signed-rank test at the 0.05 α-level to check whether the differences in the mean or median results

Table D.2: Overview of the abbreviations used in the chapter

Iterated F-Race	an algorithm for algorithm configuration that is included in the irace package [Birattari et al., 2010, López-Ibáñez et al., 2011]
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
iCMA-ES	CMA-ES with increasing population size
iCMA-ES-dp	iCMA-ES with default parameter setting
iCMA-ES-tcec	iCMA-ES with parameters tuned on CEC'05 functions f_{cec1}-f_{cec25}
iCMA-ES-tsc	iCMA-ES with parameters tuned on SOCO functions f_{soco1}-f_{soco19} (except f_{soco1}, f_{soco3}, f_{soco4} and f_{soco8})
iCMA-ES-$\oplus$	iCMA-ES with parameters tuned on all hybrid functions of SOCO
iCMA-ES-uni	iCMA-ES with parameters tuned on uni-modal functions of SOCO except f_{soco1}, f_{soco8}
iCMA-ES-multi	iCMA-ES with parameters tuned on all non-hybrid multi modal functions of SOCO except f_{soco3}, f_{soco4}
iCMA-ES-uni+	adding uni-modal functions f_{soco1} and f_{soco8} to respective training set
iCMA-ES-multi+	adding multi modal functions f_{soco3} and f_{soco4} to respective training set
References	
Sep-CMA-ES	a modification of CMA-ES that uses a diagonal matrix instead of the full covariance matrix [Ros and Hansen, 2008]
Sep-iCMA-ES	Sep-CMA-ES with increasing population size [Ros, 2009, Smit and Eiben, 2010]
Sep-iCMA-ES-tsc	results of a tuned Sep-iCMA-ES from [Liao et al., 2011c]
iCMA-ES-05	results of the Matlab version of iCMA-ES with a sophisticated bound handling mechanism from the CEC'05 special session [Auger and Hansen, 2005]
MA-LSch-CMA	a memetic algorithm integrating CMA-ES as a local search algorithm [Molina et al., 2010a]
PS-CMA-ES	a particle swarm optimization algorithm integrating CMA-ES as a local search algorithm [Müller et al., 2009]
Benchmarks	
CEC'05	special session on real parameter optimization of the 2005 IEEE Congress on Evolutionary Computation [Suganthan et al., 2005]
SOCO	benchmark set of a special issue of the Soft Computing journal on the scalability of evolutionary algorithms and other metaheuristics for large scale continuous optimization problems [Herrera et al., 2010]

obtained by two algorithms on each of the 25 CEC'05 benchmark functions is statistically significant.

In Table D.2, we give an overview of the abbreviations that are used the experimental analysis section.

D.4 Experimental study

In this section, we compare the performance of iCMA-ES-tsc to the default paramter settings, to iCMA-ES-dp, and to other algorithms that use CMA-ES as a local search operator and that have been proposed with the aim of improving over iCMA-ES.

D.4.1 iCMA-ES-tsc vs. iCMA-ES-dp

First, we focus on the improvement iCMA-ES-tsc obtains over iCMA-ES-dp. Table D.3 shows the performance of iCMA-ES-dp (default parameters) and iCMA-ES-tsc (tuned on SOCO benchmark functions) on the CEC'05 benchmark function suite. Considering the differences on individual functions, we first observe that iCMA-ES-dp and iCMA-ES-tsc reach on surprisingly many functions similar performance at least from the perspective of the applied Wilcoxon test: on 19, 17, and 14 functions for 10, 30, and 50 dimensions, respectively, no statistically significant differences could be observed. On the one hand, this may be due to the relatively small number of 25 independent runs of the algorithms; on the other hand, this is also caused by the fact that several benchmark functions are very easy to solve and therefore introduce floor effects. For example, functions f_{cec1}, f_{cec2}, f_{cec3}, f_{cec6}, and f_{cec7} are solved by iCMA-ES-dp and iCMA-ES-tsc in all runs and in all dimensions to the zero threshold. In the other functions, iCMA-ES-tsc performs better than iCMA-ES-dp except in 2, 1, and 1 cases for dimensions 10, 30, and 50, respectively, indicating superior performance of iCMA-ES-tsc over iCMA-ES-dp.

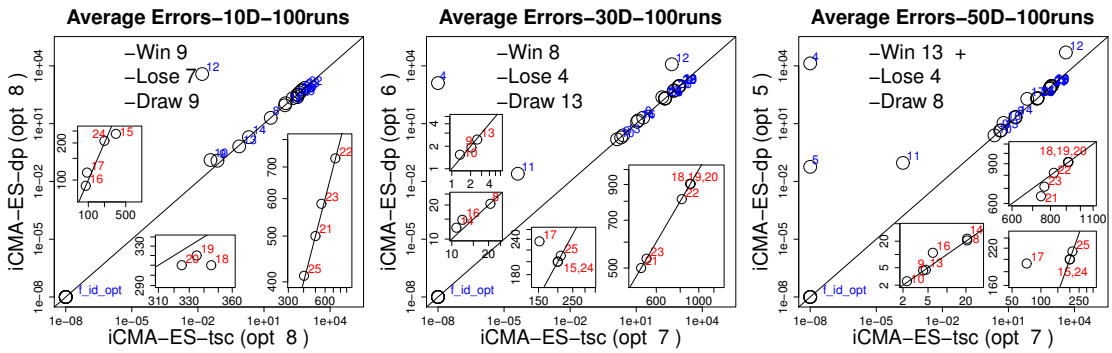


Figure D.1: Correlation plots of iCMA-ES-dp and iCMA-ES-tsc on dimensions 10, 30 and 50 respectively. Each point represents the mean error value obtained by either of the two algorithms. A point on the upper triangle delimited by the diagonal indicates better performance for the algorithm on the x-axis; a point on the lower right triangle indicates better performance for the algorithm on the y-axis. The number labeled beside some outstanding points represent the index of the corresponding function. The comparison is conducted based on mean error values and the comparison results of the algorithm on the x-axis are presented in form of -win, -draw, -lose, respectively. We marked with a + symbol those cases in which there is a statistically significant difference at the 0.05 α-level between the algorithms. The number of opt on the axes shows the number of means lower than the zero threshold by the corresponding algorithm.

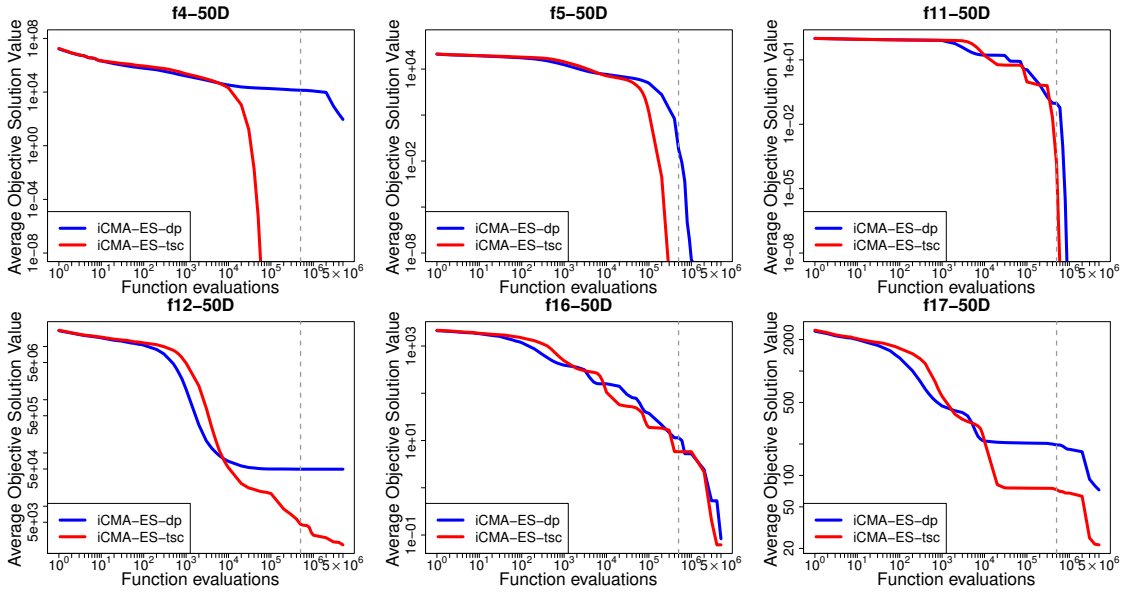


Figure D.2: The development of the mean error of the fitness values across 100 independent runs of iCMA-ES-dp and iCMA-ES-tsc over the number of function evaluations on functions f_{cec4}, f_{cec5}, f_{cec11}, f_{cec12}, f_{cec16} and f_{cec17} of 50 dimensions. The vertical, dotted line in each plot indicates 5.00E+05 function evaluations, which is the termination criterion for the number of function evaluations in the CEC'05 protocol.

When comparing the means and medians obtained across each of the benchmark functions, we can observe that iCMA-ES-tsc reaches statistically better results on dimension 50 according to the Wilcoxon test, while on dimensions 10 and 30 the observed differences are not statistically significant. However, on a function-by-function basis iCMA-ES-tsc is statistically better than iCMA-ES-dp on more functions than vice-versa.

On few functions, the differences in the solution qualities are very strong. As an example, consider function f_{cec4}, where iCMA-ES-dp stagnated at very high mean error values of 6.58E+02 and 1.43E+04 for dimensions 30 and 50, respectively, while iCMA-ES-tsc reached in each trial a solution better than the zero threshold. On other functions of dimension 50, such as functions f_{cec12}, f_{cec16}, and f_{cec17}, iCMA-ES-tsc more than halved the error values that were reached by iCMA-ES-dp.

Fig D.1 shows correlation plots where each point has as x and y coordinate the mean error obtained with iCMA-ES-tsc and iCMA-ES-dp on a same function. The plots of Fig D.1 show the mean errors for the 10, 30, and 50 dimensional problems, respectively. Clearly, on some functions iCMA-ES-tsc reaches results that are of

much better solution quality than those of iCMA-ES-dp (indicated by the circles that are above the diagonal). This is the case especially on functions f_{cec4}, f_{cec5}, f_{cec11}, f_{cec12}, f_{cec16} and f_{cec17}. f_{cec4} and f_{cec17} are the two noisy functions of the CEC'05 benchmark. The other four are multi-modal functions; among these, f_{cec5} has the optimum on the bounds. Next, we focus on these six functions.

Fig D.2 shows the development of the mean error for iCMA-ES-dp and iCMA-ES-tsc over the number of function evaluations on functions f_{cec4}, f_{cec5}, f_{cec11}, f_{cec12}, f_{cec16} and f_{cec17} of dimension 50. We observe that iCMA-ES-tsc and iCMA-ES-dp perform similar up to about 1.00E+04 function evaluations. As the number of function evaluation increases, the advantage of iCMA-ES-tsc over iCMA-ES-dp starts to become apparent. At the stopping criterion of 5.00E+05 function evaluations from the CEC'05 competition rules (indicated by the dotted, vertical lines in the plots), iCMA-ES-tsc shows generally much lower mean errors. Especially on f_{cec4} and f_{cec5}, iCMA-ES-tsc converges fast to the zero threshold after 1.00E+04 function evaluations. Looking at what happens beyond the termination criterion of 5.00E+06 function evaluations (right from the dotted vertical lines in the plots), we can see that iCMA-ES-dp catches up with the lower mean errors of iCMA-ES-tsc on functions f_{cec5}, f_{cec11}, and f_{cec16}. Hence, on these functions the tuned parameter settings appear to result in a faster convergence towards near-optimal solutions. On functions f_{cec4} and f_{cec12} the advantage of iCMA-ES-tsc with respect to the mean error remains substantial. These general conclusions are also backed up by a more detailed analysis of the algorithms' qualified run-length distributions (RLDs) [Hoos and Stützle, 2005]. Qualified RLDs give the distribution of the number of function evaluations to reach specific bounds on the errors. For details on the qualified RLDs, which are measured across 100 independent algorithm trials, we refer to supplementary information pages `http://iridia.ulb.ac.be/supp/IridiaSupp2011-023`.

Finally, we consider qualified RLDs for iCMA-ES-dp and iCMA-ES-tsc on functions f_{cec1}, f_{cec2}, f_{cec3}, f_{cec6} and f_{cec7} on dimension 50. On these functions each trial of iCMA-ES-dp and iCMA-ES-tsc reaches the zero threshold within the termination criterion of the CEC'05 protocol. Fig D.3 shows the qualified RLDs for reaching the zero threshold over 100 independent runs for iCMA-ES-dp and iCMA-ES-tsc on these five functions. We observe that on f_{cec1}, f_{cec2} and f_{cec3}, both iCMA-ES-dp and iCMA-ES-tsc converge very fast to the zero threshold in each trial without recurring to restarts. For these three, relatively easy functions, iCMA-ES-tsc converges slightly more slowly than iCMA-ES-dp mainly because of its larger initial population size. On f_{cec6} and f_{cec7}, in contrast, iCMA-ES-

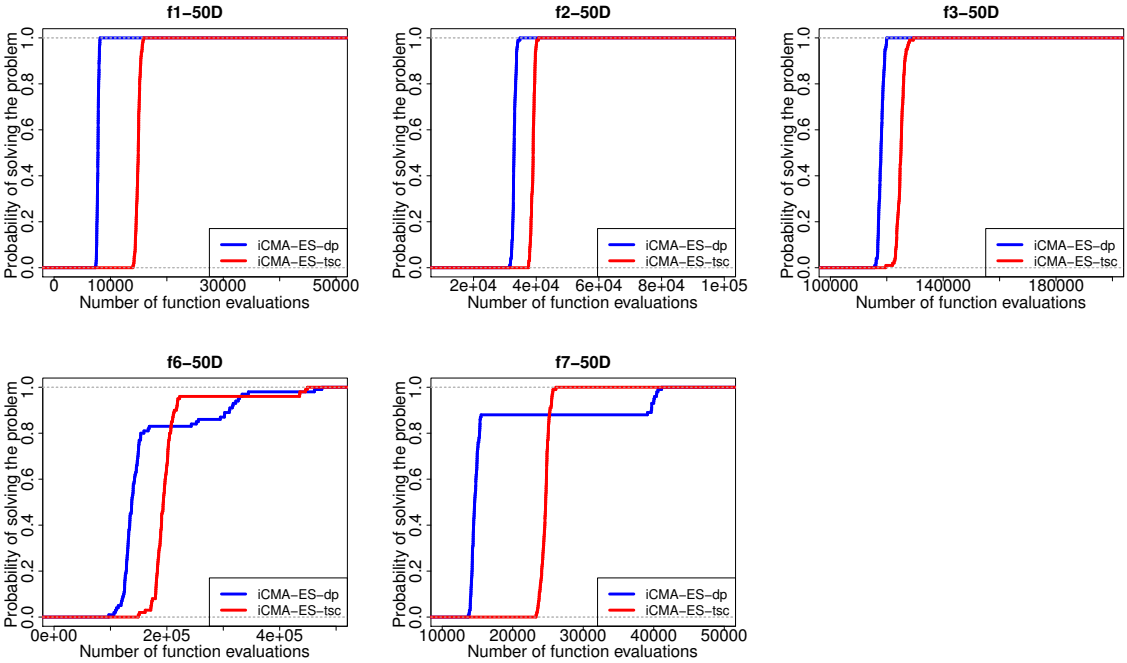


Figure D.3: The qualified run-length distributions (RLDs, for short) over 100 independent runs obtained by iCMA-ES-dp and iCMA-ES-tsc on the 50 dimensional versions of functions f_{cec1}, f_{cec2}, f_{cec3}, f_{cec6} and f_{cec7}. The solution quality demanded is 1.00E−08 for each function.

tsc reaches a 100% success rate faster than iCMA-ES-dp, although there is no dominance relationship among the RLDs.

As said at the end of the previous section, the parameter settings of iCMA-ES-tsc imply a larger exploration at the beginning of the search. For example, for dimension 50, the population size is 15 for the default settings but 41 for the tuned settings, that is, almost three times larger.[2] The experimental results are somehow in accordance with this interpretation of a higher exploration. In fact, on the hard noisy functions, most multi-modal and especially the hybrid functions the larger exploration apparently leads to better final performance. However, the larger initial exploration also leads to a slightly slower convergence to the optimum on several, relatively easily solved unimodal functions such as f_{cec1} , f_{cec2} and f_{cec3}, as we have shown through qualified RTDs in Figure D.3.

[2]Note that a different interpretation of the population size is that not the parameter setting for factor a should be changed but possibly the scaling function with the problem dimension. In fact, the scaling by a logarithmic function may be too weak and other scaling laws may be examined leading to overall better behavior of iCMA-ES.

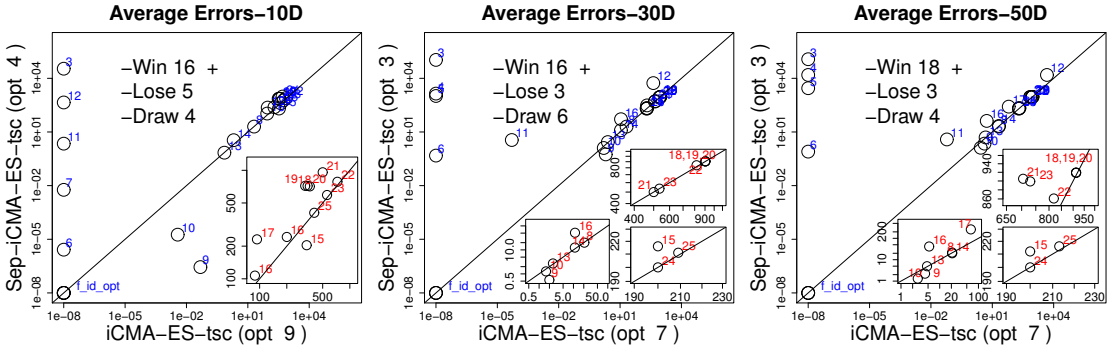


Figure D.4: Correlation plots of iCMA-ES-tsc and Sep-iCMA-ES-tsc on dimensions 10, 30 and 50 respectively. Each point represents the mean error value over 25 independent runs obtained by either of the two algorithms. A point on the upper triangle delimited by the diagonal indicates better performance for the algorithm on the x-axis; a point on the lower right triangle indicates better performance for the algorithm on the y-axis. The number labeled beside some outstanding points represent the index of the corresponding function. The comparison is conducted based on mean error values and the comparison results of the algorithm on the x-axis are presented in the form of -win, -draw, -lose, respectively, using iCMA-ES-tsc as the reference. We marked with a + symbol those cases in which there is a statistically significant difference at the 0.05 α-level between the algorithms. The number of opt on the axes shows the number of means that is lower than the zero threshold, obtained by the corresponding algorithm.

D.4.2 iCMA-ES-tsc vs. iCMA-ES-tcec

One may wonder whether tuning iCMA-ES on the CEC'05 benchmark suite directly incurs better final performance on this set of functions. To explore this question, we compare in Table D.4 the performance of iCMA-ES-tcec and iCMA-ES-tsc on the CEC'05 benchmark set. iCMA-ES-tcec and iCMA-ES-tsc are mutually statistically better than each other on four functions of dimension 10, respectively. On the 10 dimensional functions, iCMA-ES-tsc is slightly worse than iCMA-ES-tcec with respect to the distribution of mean or median errors. This may be due to the fact that iCMA-ES-tcec is tuned on the 10 dimensional CEC'05 benchmark set, the same functions on which it is tested. Interestingly, this slight superiority of iCMA-ES-tcec on the 10 dimensional functions does not generalize to higher dimensions. As an example, consider functions f_{cec18}, f_{cec19}, f_{cec20} of dimension 10. On these, iCMA-ES-tcec obtains an error value of 3.00E+02 in all independent 25 runs which is the lowest value reported in the literature for these functions as far as we aware. However, on the 50 dimensional version of these functions, iCMA-ES-tcec is sig-

nificantly worse than iCMA-ES-tsc. Moreover, considering the differences on all functions of dimension 50, iCMA-ES-tsc statistically significantly improves upon iCMA-ES-tcec on 12 functions while it performs statistically significantly worse than iCMA-ES-tcec on only 3 functions. Considering the distribution of the mean or median error values of the 50 dimensional functions, iCMA-ES-tsc statistically significantly improves upon iCMA-ES-tcec.

It should also be mentioned that tuning on the SOCO benchmark functions is much faster than on the CEC'05 benchmark set. In fact, the difference in computation time amounts to a factor of about 50. This difference is mainly due to the fact that 16 of the 25 CEC'05 functions of each dimension are rotated functions, which requires more costly computations in the evaluation such as multiplication operations on a rotated matrix.

D.4.3 Comparison to state-of-the-art methods that exploit CMA-ES

At least two recent, newly designed state-of-the-art algorithms exploit CMA-ES as an underlying local search method; these are a memetic algorithm with local search chains based on CMA-ES (MA-LSch-CMA) [Molina et al., 2010a] and a hybridization of a PSO algorithm with CMA-ES (PS-CMA-ES) [Müller et al., 2009]. We compare iCMA-ES-tsc to these following the experimental analysis used in Molina et al. [2010a] and Müller et al. [2009], that is by (i) statistically analyzing for the distribution of the mean errors as in Molina et al. [2010a] and, (ii) ranking the mean errors as in Müller et al. [2009]. As the results of MA-LSch-CMA and PS-CMA-ES we use those reported in Liao et al. [2011a]. Note that the original results reported in Molina et al. [2010a], Müller et al. [2009] did not necessarily satisfy the bound constraints of the CEC'05 benchmark functions, which is corrected in the results reported in [Liao et al., 2011a]. Table D.5 shows that iCMA-ES-tsc performs statistically significantly better than MA-LSch-CMA in all dimensions; iCMA-ES-tsc performs statistically significantly better than PS-CMA-ES on the 50 dimensional functions and it reaches better performance than PS-CMA-ES on more functions for dimensions 10 and 30. Table D.5 also shows that iCMA-ES-tsc obtains the best average ranking in all dimensions and most often the zero threshold in all dimensions. Clearly, it would be interesting to also automatically tune MA-LSch-CMA and PS-CMA-ES to exploit possibly more their potential. Nevertheless, this comparison indicates that a feasible way to go for improving the performance of iCMA-ES-dp is to further fine-tune iCMA-ES parameters (or

maybe other design choices of iCMA-ES) instead of embedding CMA-ES into other algorithms. Recall that this also justifies the effort in tuning iCMA-ES because when designing new hybrid algorithms, often also a substantially large effort flows into the further, often manual fine-tuning of algorithm parameters and algorithm designs.

D.5 Additional experiments

D.5.1 Comparison to other results by iCMA-ES

We also compared iCMA-ES-tsc to Sep-iCMA-ES-tsc [Liao et al., 2011c], the algorithm used by Smit and Eiben [2010], on the full CEC'05 benchmark set. Fig. D.4 shows correlation plots that illustrate the relative performance for Sep-iCMA-ES-tsc and iCMA-ES-tsc on dimensions 10, 30 and 50, respectively. Each point represents the mean error value obtained by either of the two algorithms. These plots indicate superior performance of iCMA-ES-tsc over Sep-iCMA-ES-tsc, which is confirmed by the table of full results available at `http://iridia.ulb.ac.be/supp/IridiaSupp2011-023`. We verified that iCMA-ES-tsc reaches statistically significantly better performance than Sep-iCMA-ES-tsc on the distribution of mean error values on all dimensions. This comparison confirms our expectation of iCMA-ES-tsc's superiority over Sep-iCMA-ES-tsc on the CEC'05 benchmark set, where 16 of 25 functions are rotated functions. The most significant example is f_{cec3}, a unimodal rotated high conditional function, where Sep-iCMA-ES-tsc stagnated at very high mean error values for all dimensions, while iCMA-ES-tsc reached in each trial the zero threshold. However, Sep-iCMA-ES-tsc obtains better performance than iCMA-ES-tsc on f_{cec10}, a rotated Rastrigin function, over all dimensions. This case gives an indication that we can only conclude that iCMA-ES's rotational invariance plays a pivotal role to handle most but not all rotated functions.

Next, we take the mean errors reported for iCMA-ES in the CEC'05 special session as a reference and refer to these results as iCMA-ES-05. Note that the results of iCMA-ES-05 were obtained with a different implementation (using the Matlab and not the C code we use) and with a much more sophisticated bound handling mechanism. We summarize the comparison with iCMA-ES-tsc in Table D.6. iCMA-ES-tsc performs statistically significantly better than iCMA-ES-05 on dimension 30 and it reaches on more functions statistically significantly better results (on a per function basis). This confirms the high performance of iCMA-

ES-tsc.

D.5.2 Tuning setup

In this section, we examine different choices for the composition of the training set to obtain an indication which types of functions are important for high performance of the tuned iCMA-ES. We also explore alternative settings of the irace tool.

In what follows, we define training sets that are composed of different subsets of the SOCO benchmark functions and evaluate the tuned performance of iCMA-ES using the mean errors on the 50-dimensional CEC'05 benchmark functions. We summarize here our main findings and for detailed numerical results we refer to the supplementary page `http://iridia.ulb.ac.be/supp/IridiaSupp2011-023`.

The convention we use for labeling the training function sets is introduced first:

$\oplus$	all hybrid functions of SOCO
uni	uni-modal functions of SOCO except f_{soco1}, f_{soco8}
multi	all non-hybrid multi modal functions of SOCO except f_{soco3}, f_{soco4}
+	adds the uni-modal (f_{soco1}, f_{soco8}) or multi modal (f_{soco3},f_{soco4}) functions to respective training sets

For example, iCMA-ES-$\oplus$ denotes iCMA-ES tuned using only the eight hybrid functions, iCMA-ES-uni,multi denotes iCMA-ES tuned with the seven uni-modal and (non-hybrid) multi-modal functions of SOCO, and iCMA-ES-uni+ denotes iCMA-ES tuned with all seven uni-modal functions.

In Table D.7 we summarize the average ranking and the statistical analysis of several parameter settings of iCMA-ES that were obtained with the various training set compositions that we considered. In Table D.8 are given the parameter setting obtained using these training set compositions; the parameter settings for iCMA-ES-tsc and iCMA-ES-dp are given in Table D.1. The main conclusions we can obtain are the following.

1. The usage of the hybrid functions in the training set is a key to high tuned performance. iCMA-ES-$\oplus$ obtains about the same performance as iCMA-ES-tsc and if the hybrid functions are not part of the training set, the performance of the tuned iCMA-ES degrades considerably. Interestingly, the configuration iCMA-ES-$\oplus$ obtains on all 50-dimensional hybrid functions of CEC'05 significantly better results than iCMA-ES-tsc-uni,multi, indicating

that there are maybe some common aspects between the hybrid functions of the SOCO and the CEC'05 benchmark set.

2. The usage of the multi-modal functions only, that is, configuration iCMA-ES-multi, leads to significantly worse performance than iCMA-ES-$\oplus$. One may object that the set multi contains only two training functions; however, adding the two multi-modal functions f_{soco3} and f_{soco4} to the training set does not lead to much improved performance (configuration iCMA-ES-multi+, see `http://iridia.ulb.ac.be/supp/IridiaSupp2011-023`).

3. Configuration iCMA-ES-uni leads to, at first sight, surprisingly high performance on the CEC'05 functions, and it has only a slightly worse mean rank than iCMA-ES-$\oplus$. At a second glance, it is noteworthy that uni-modal functions such as those in the set "uni" can actually be quite difficult to optimize; for example, the default parameter setting of iCMA-ES has poor performance on uni-modal functions f_{cec4} and f_{cec5} (see Table D.3). Configurations iCMA-ES-uni+, which uses also functions f_{soco1} and f_{soco8} in the training set, is, however, worse than iCMA-ES-uni (and significantly worse performing than iCMA-ES-$\oplus$). This is possibly caused by floor effects obtained due to adding functions that are easily solved by iCMA-ES.

A common pattern among the best performing parameter settings, which are iCMA-ES-tsc, iCMA-ES-$\oplus$, and iCMA-ES-uni, is that they tend to increase the exploration performed by iCMA-ES. In fact, the commonalities of these parameter settings are a higher population size, a (slightly) larger initial step size, and a faster increase of the population size upon a restart than the default parameter settings. Since these parameter settings improve performance, in particular, on the hardest benchmark problems, it may be that the default settings were possibly biased by experiments on too simple benchmark functions.

Considering the tuning setup, we also made tests (i) replacing the F-test with a t-test (that is, using the Student t-test for the race) and (ii) increasing the tuning budget to 25000 runs. Similar to the results reported previously by iCMA-ES-tsc, the resulting configurations improved upon iCMA-ES-dp, being statistically significantly better than iCMA-ES-dp on the distribution of the mean or the median error values. These experiments also indicate that the observation of the superior performance of iCMA-ES-tsc over iCMA-ES-dp is relatively stable with respect to some (minor) changes in the tuning setup. The detailed data of these trials are available at `http://iridia.ulb.ac.be/supp/IridiaSupp2011-023`.

D.6 Conclusions and future work

In this chapter, we tuned iCMA-ES to improve its performance on the CEC'05 benchmark set. We did so by using a separation between training and test set to avoid the bias of the results due to potentially overtuning the algorithm. Our experimental results showed that the tuned iCMA-ES improves significantly over the default parameter settings of iCMA-ES-dp. While on some individual functions the improvements observed from a solution quality perspective are rather large, on many other functions only minor though often statistically significant improvements are observed. iCMA-ES-tsc also performs competitive or superior to methods, such as MA-LSch-CMA [Molina et al., 2010a] and PS-CMA-ES [Müller et al., 2009], which were developed with the goal of improving over iCMA-ES performance. This indicates that, instead of embedding CMA-ES into other algorithms to improve over its performance, a viable, alternative approach is to further fine-tune the parameter settings or maybe some design choices of iCMA-ES. This direction would involve to further parameterize choices that are currently fixed in the algorithm. Examples of such parameterizations are to treat further constants as parameters that are to be tuned or to consider alternative choices for specific functions. A concrete example could be the formula that is used to determine the initial population size, which is $4 + \lfloor a \ln(D) \rfloor$ (see also Table D.1). Here, the constant 4 could be replaced by a real-valued parameter and different functions (instead of + and ln may be considered.

It is also interesting to consider the impact the tuned parameter settings have on the behavior of iCMA-ES. In fact, a common pattern among the best performing tuned parameter settings we observed is that they lead to an increased exploration of the search space at least in the initial search phases and upon a restart of iCMA-ES. This more explorative behavior is implied by larger population sizes, larger step sizes, and a higher factor for the increase of the population size upon a restart of iCMA-ES. Interestingly, increasing search space exploration is also often the goal of hybrid algorithms such as the above mentioned MA-LSch-CMA and PS-CMA-ES where CMA-ES is used as a local search. In fact, it seems that such increased exploration can directly provided inside the iCMA-ES framework by modified parameter settings.

Our experimental results also indicate that using off-line automatic algorithm configuration to further improve adaptive algorithms is a viable approach—recall that iCMA-ES is such an adaptive algorithm where step sizes and search directions are adapted to the particular continuous optimization function under concern.

We have also presented initial results examining the role of the specific composition of a training set on the performance of the tuned parameter settings. On the one hand, these results indicated that the hybrid functions in the SOCO benchmark set alone are enough to derive high-performing tuned parameter settings. Maybe surprisingly, using only the uni-modal functions of the SOCO benchmark set resulted in a same level of performance of the tuned iCMA-ES on the CEC'05 benchmark set. Although it is known that uni-modal functions can be difficult to optimize, in future research the importance of the training set should be examined in much more detail. For this task, it is important to consider interactions between algorithm properties and properties of the training set. For example, the fact that iCMA-ES is rotationally invariant made it possible to use the SOCO benchmark set of functions which contains only unrotated functions—the rotational invariance implies that iCMA-ES's performance should be unaffected by rotations of the functions. For algorithms that are not invariant with respect to rotations, the usage of the SOCO benchmark set as training set may actually lead to poor performance. An interesting direction here would be to consider other benchmark set such as the BBOB benchmark function suite (see `http://coco.gforge.inria.fr/doku.php?id=bbob-2012`) or newly designed benchmark suites containing functions with specific properties for the tuning of continuous optimizers.

Table D.3: Results of the comparison between iCMA-ES-dp and iCMA-ES-tsc over 25 independent runs for CEC'05 functions. Symbols $<$, $\approx$, and $>$ denote whether the performance of iCMA-ES-dp is statistically better, indifferent, or worse than that of iCMA-ES-tsc according to the two-sided Wilcoxon matched-pairs signed-rank test at the 0.05 α-level. The numbers in parenthesis represent the times of $<$, $\approx$, and $>$, respectively. The numbers in parenthesis for ($<$, $=$, $>$) represent the times we have $<$, $=$, and $>$, respectively, when iCMA-ES-dp is compared with iCMA-ES-tsc based on the mean or median errors.

f_{cec}	10 dimensions		30 dimensions		50 dimensions	
	iCMA-ES-dp Mean and Median	iCMA-ES-tsc Mean and Median	iCMA-ES-dp Mean and Median	iCMA-ES-tsc Mean and Median	iCMA-ES-dp Mean and Median	iCMA-ES-tsc Mean and Median
f_1	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_2	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_3	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_4	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	6.58E+02 1.75E+00 $>$	1.00E−08 1.00E−08	1.43E+04 1.27E+04 $>$	1.00E−08 1.00E−08
f_5	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	7.41E−02 8.61E−08 $>$	1.00E−08 1.00E−08
f_6	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_7	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_8	2.00E+01 2.00E+01 $\approx$	2.02E+01 2.00E+01	2.04E+01 2.00E+01 $<$	2.08E+01 2.10E+01	2.09E+01 2.11E+01 $\approx$	2.10E+01 2.11E+01
f_9	1.59E−01 1.00E−08 $\approx$	4.81E−02 1.00E−08	1.87E+00 1.99E+00 $\approx$	1.99E+00 1.99E+00	4.36E+00 3.98E+00 $\approx$	4.18E+00 3.98E+00
f_{10}	3.18E−01 1.00E−08 $\approx$	3.73E−03 1.00E−08	1.44E+00 9.95E−01 $\approx$	1.59E+00 9.95E−01	2.89E+00 1.99E+00 $\approx$	2.71E+00 2.98E+00
f_{11}	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	7.17E−02 1.00E−08 $>$	5.09E−05 1.00E−08	9.94E−02 1.00E−08 $\approx$	6.03E−02 1.00E−08
f_{12}	4.07E+03 1.00E−08 $>$	1.00E−08 1.00E−08	1.19E+04 4.84E+03 $>$	4.22E+02 5.96E+01	4.25E+04 2.78E+04 $>$	4.69E+03 3.39E+03
f_{13}	6.49E−01 6.37E−01 $\approx$	7.14E−01 6.94E−01	2.63E+00 2.71E+00 $\approx$	2.53E+00 2.57E+00	4.44E+00 4.37E+00 $<$	4.70E+00 4.62E+00
f_{14}	1.96E+00 1.98E+00 $\approx$	2.03E+00 2.09E+00	1.26E+01 1.26E+01 $>$	1.10E+01 1.13E+01	2.28E+01 2.30E+01 $>$	2.09E+01 2.12E+01
f_{15}	2.15E+02 2.00E+02 $<$	3.32E+02 4.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02
f_{16}	9.04E+01 9.11E+01 $>$	8.86E+01 9.03E+01	1.48E+01 1.51E+01 $>$	1.11E+01 1.07E+01	1.10E+01 1.14E+01 $>$	5.34E+00 5.47E+00
f_{17}	1.17E+02 1.09E+02 $>$	9.34E+01 9.43E+01	2.52E+02 1.80E+02 $>$	2.08E+02 5.60E+01	1.91E+02 1.62E+02 $>$	6.36E+01 4.99E+01
f_{18}	3.16E+02 3.00E+02 $<$	3.60E+02 3.00E+02	9.04E+02 9.04E+02 $\approx$	9.04E+02 9.04E+02	9.13E+02 9.16E+02 $>$	9.13E+02 9.13E+02
f_{19}	3.20E+02 3.00E+02 $\approx$	3.20E+02 3.00E+02	9.04E+02 9.04E+02 $\approx$	9.04E+02 9.04E+02	9.13E+02 9.15E+02 $>$	9.13E+02 9.13E+02
f_{20}	3.20E+02 3.00E+02 $\approx$	3.40E+02 3.00E+02	9.04E+02 9.04E+02 $\approx$	9.04E+02 9.04E+02	9.15E+02 9.15E+02 $>$	9.13E+02 9.13E+02
f_{21}	5.00E+02 5.00E+02 $\approx$	5.00E+02 5.00E+02	5.00E+02 5.00E+02 $\approx$	5.00E+02 5.00E+02	6.64E+02 5.00E+02 $\approx$	7.05E+02 5.00E+02
f_{22}	7.28E+02 7.28E+02 $\approx$	7.28E+02 7.28E+02	8.10E+02 8.11E+02 $\approx$	8.17E+02 8.24E+02	8.19E+02 8.18E+02 $\approx$	8.19E+02 8.21E+02
f_{23}	5.86E+02 5.59E+02 $\approx$	5.59E+02 5.59E+02	5.34E+02 5.34E+02 $\approx$	5.34E+02 5.34E+02	6.97E+02 5.40E+02 $\approx$	7.30E+02 5.40E+02
f_{24}	2.33E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02
f_{25}	4.34E+02 4.04E+02 $>$	4.03E+02 4.03E+02	2.10E+02 2.10E+02 $>$	2.09E+02 2.09E+02	2.14E+02 2.14E+02 $>$	2.13E+02 2.13E+02
Mean	f_1-f_{25} ($<$, $=$, $>$): (6, 11, 8)		f_1-f_{25}($<$, $=$, $>$): (4, 13, 8)		f_1-f_{25}($<$, $=$, $>$): (4, 10, 11)†	
Median	f_1-f_{25} ($<$, $=$, $>$): (3, 19, 3)		f_1-f_{25}($<$, $=$, $>$): (2, 16, 7)		f_1-f_{25}($<$, $=$, $>$): (3, 12, 10)†	
By Func	f_1-f_{25} ($<$, $\approx$, $>$): (2, 19, 4)		f_1-f_{25}($<$, $\approx$, $>$): (1, 17, 7)		f_1-f_{25}($<$, $\approx$, $>$): (1, 14, 10)	
	10, 30 and 50 dimensional f_1-f_{25} ($<$, $>$):(5%, 28%)					

† denotes there is a significant difference over the distribution of mean or median errors between iCMA-ES-dp with iCMA-ES-tsc by a two-sided Wilcoxon matched-pairs signed-ranks test at the 0,05 α-level.

Table D.4: Comparison between iCMA-ES-tcec and iCMA-ES-tsc over 25 independent runs for CEC'05 functions. Symbols $<$, $\approx$, and $>$ denote whether the performance of iCMA-ES-tcec is statistically better, indifferent, or worse than that of iCMA-ES-tsc according to the two-sided Wilcoxon matched-pairs signed-rank test at the 0.05 α-level. The numbers in parenthesis represent the times of $<$, $\approx$, and $>$, respectively. The numbers in parenthesis for ($<$, $=$, $>$) represent the times we have $<$, $=$, and $>$, respectively, when iCMA-ES-dp is compared with iCMA-ES-tsc based on the mean or median errors.

f_{cec}	10 dimensions		30 dimensions		50 dimensions	
	iCMA-ES-tcec Mean and Median	iCMA-ES-tsc Mean and Median	iCMA-ES-tcec Mean and Median	iCMA-ES-tsc Mean and Median	iCMA-ES-tcec Mean and Median	iCMA-ES-tsc Mean and Median
f_1	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_2	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_3	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_4	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	2.85E+02 1.00E−08 $>$	1.00E−08 1.00E−08
f_5	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	2.69E−08 2.72E−08 $>$	1.00E−08 1.00E−08	5.02E−08 4.85E−08 $>$	1.00E−08 1.00E−08
f_6	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	2.79E−02 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_7	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08
f_8	2.01E+01 2.00E+01 $\approx$	2.02E+01 2.00E+01	2.08E+01 2.09E+01 $\approx$	2.08E+01 2.10E+01	2.08E+01 2.11E+01 $<$	2.10E+01 2.11E+01
f_9	2.64E−01 1.00E−08 $>$	4.81E−02 1.00E−08	2.15E+00 1.99E+00 $\approx$	1.99E+00 1.99E+00	5.49E+00 5.97E+00 $>$	4.18E+00 3.98E+00
f_{10}	1.59E−01 1.00E−08 $>$	3.73E−03 1.00E−08	1.67E+00 1.99E+00 $>$	1.59E+00 9.95E−01	3.54E+00 3.98E+00 $>$	2.71E+00 2.98E+00
f_{11}	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	8.69E−02 1.00E−08 $>$	5.09E−05 1.00E−08	1.83E−01 1.00E−08 $\approx$	6.03E−02 1.00E−08
f_{12}	1.00E−08 1.00E−08 $\approx$	1.00E−08 1.00E−08	4.88E+02 1.92E+02 $\approx$	4.22E+02 5.96E+01	6.41E+03 4.59E+03 $\approx$	4.69E+03 3.39E+03
f_{13}	6.14E−01 6.46E−01 $<$	7.14E−01 6.94E−01	2.48E+00 2.50E+00 $\approx$	2.53E+00 2.57E+00	4.42E+00 4.47E+00 $<$	4.70E+00 4.62E+00
f_{14}	8.07E−01 6.66E−01 $<$	2.03E+00 2.09E+00	9.93E+00 9.94E+00 $<$	1.10E+01 1.13E+01	2.02E+01 1.98E+01 $<$	2.09E+01 2.12E+01
f_{15}	2.96E+02 3.00E+02 $<$	3.32E+02 4.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02
f_{16}	8.83E+01 8.94E+01 $\approx$	8.86E+01 9.03E+01	1.06E+01 1.04E+01 $\approx$	1.11E+01 1.07E+01	9.46E+00 9.09E+00 $>$	5.34E+00 5.47E+00
f_{17}	1.20E+02 1.17E+02 $>$	9.34E+01 9.43E+01	2.14E+02 1.52E+02 $\approx$	2.08E+02 5.60E+01	9.55E+01 8.06E+01 $>$	6.36E+01 4.99E+01
f_{18}	3.00E+02 3.00E+02 $<$	3.60E+02 3.00E+02	9.04E+02 9.04E+02 $\approx$	9.04E+02 9.04E+02	9.15E+02 9.16E+02 $>$	9.13E+02 9.13E+02
f_{19}	3.00E+02 3.00E+02 $\approx$	3.20E+02 3.00E+02	9.04E+02 9.04E+02 $\approx$	9.04E+02 9.04E+02	9.16E+02 9.16E+02 $>$	9.13E+02 9.13E+02
f_{20}	3.00E+02 3.00E+02 $\approx$	3.40E+02 3.00E+02	9.04E+02 9.04E+02 $\approx$	9.04E+02 9.04E+02	9.15E+02 9.14E+02 $>$	9.13E+02 9.13E+02
f_{21}	5.00E+02 5.00E+02 $\approx$	5.00E+02 5.00E+02	5.00E+02 5.00E+02 $\approx$	5.00E+02 5.00E+02	9.68E+02 1.01E+03 $>$	7.05E+02 5.00E+02
f_{22}	7.27E+02 7.26E+02 $\approx$	7.28E+02 7.28E+02	8.10E+02 8.07E+02 $\approx$	8.17E+02 8.24E+02	8.15E+02 8.14E+02 $\approx$	8.19E+02 8.21E+02
f_{23}	5.59E+02 5.59E+02 $\approx$	5.59E+02 5.59E+02	5.34E+02 5.34E+02 $\approx$	5.34E+02 5.34E+02	9.04E+02 1.02E+03 $>$	7.30E+02 5.40E+02
f_{24}	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02	2.00E+02 2.00E+02 $\approx$	2.00E+02 2.00E+02
f_{25}	4.04E+02 4.04E+02 $>$	4.03E+02 4.03E+02	2.09E+02 2.09E+02 $>$	2.09E+02 2.09E+02	2.13E+02 2.13E+02 $>$	2.13E+02 2.13E+02
Mean	f_1-f_{25} ($<$, $=$, $>$): (9, 12, 4)		f_1-f_{25}($<$, $=$, $>$): (4, 14, 7)		f_1-f_{25}($<$, $=$, $>$): (4, 8, 14)†	
Median	f_1-f_{25} ($<$, $=$, $>$): (5, 18, 2)		f_1-f_{25}($<$, $=$, $>$): (5, 16, 4)		f_1-f_{25}($<$, $=$, $>$): (3, 11, 11)†	
By Func	f_1-f_{25} ($<$, $\approx$, $>$): (4, 17, 4)		f_1-f_{25}($<$, $\approx$, $>$): (1, 20, 4)		f_1-f_{25}($<$, $\approx$, $>$): (3, 10, 12)	
	10, 30 and 50 dimensional f_1-f_{25} ($<$, $>$):(11%, 27%)					

† denotes there is a significant difference over the distribution of mean or median errors between iCMA-ES-tcec with iCMA-ES-tsc by a two-sided Wilcoxon matched-pairs signed-ranks test at the 0,05 α-level.

Table D.5: The mean errors obtained by MA-LSch-CMA, PS-CMA-ES and iCMA-ES-tsc (MA, PS, iCMAESt for their abbreviations, respectively, in this table) over 25 independent runs for CEC'05 functions. The numbers in parenthesis represent the times of $<$, $=$, and $>$, respectively, when the corresponding algorithms are compared with iCMA-ES-tsc based on the mean errors. The number of means below the zero-threshold found by each algorithm (indicated by "Optima" and the average ranking of each algorithm are also given.

f_{cec}	10 dimensions			30 dimensions			50 dimensions		
	Mean Errors			Mean Errors			Mean Errors		
	MA	PS	iCMAESt	MA	PS	iCMAESt	MA	PS	iCMAESt
f_1	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08
f_2	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	1.00E−08	3.06E−02	7.36E−06	1.00E−08
f_3	1.00E−08	1.45E−01	1.00E−08	2.75E+04	2.96E+04	1.00E−08	3.21E+04	9.10E+04	1.00E−08
f_4	5.54E−03	1.00E−08	1.00E−08	3.02E+02	4.56E+03	1.00E−08	3.23E+03	2.17E+04	1.00E−08
f_5	6.75E−07	1.00E−08	1.00E−08	1.26E+03	2.52E+01	1.00E−08	2.69E+03	1.79E+03	1.00E−08
f_6	3.19E−01	1.00E−08	1.00E−08	1.12E+00	1.15E+01	1.00E−08	4.10E+00	2.91E+01	1.00E−08
f_7	1.43E−01	1.00E−08	1.00E−08	1.75E−02	1.00E−08	1.00E−08	5.40E−03	1.00E−08	1.00E−08
f_8	2.00E+01	2.00E+01	2.02E+01	2.00E+01	2.00E+01	2.08E+01	2.00E+01	2.00E+01	2.10E+01
f_9	1.00E−08	3.98E−02	4.81E−02	1.00E−08	8.76E−01	1.99E+00	1.00E−08	5.45E+00	4.18E+00
f_{10}	2.67E+00	1.00E−08	3.73E−03	2.25E+01	5.57E−01	1.59E+00	5.01E+01	5.33E+00	2.71E+00
f_{11}	2.43E+00	8.51E−01	1.00E−08	2.15E+01	7.10E+00	5.09E−05	4.13E+01	1.59E+01	6.03E−02
f_{12}	1.14E+02	1.10E+00	1.00E−08	1.67E+03	8.80E+02	4.22E+02	1.39E+04	6.90E+03	4.69E+03
f_{13}	5.45E−01	3.67E−01	7.14E−01	2.03E+00	2.05E+00	2.53E+00	3.15E+00	4.15E+00	4.70E+00
f_{14}	2.25E+00	3.40E+00	2.03E+00	1.25E+01	1.24E+01	1.10E+01	2.22E+01	2.15E+01	2.09E+01
f_{15}	2.24E+02	8.67E+01	3.32E+02	3.00E+02	1.37E+02	2.00E+02	3.72E+02	1.25E+02	2.00E+02
f_{16}	9.18E+01	9.28E+01	8.86E+01	1.26E+02	1.59E+01	1.11E+01	6.90E+01	1.62E+01	5.34E+00
f_{17}	1.01E+02	1.12E+02	9.34E+01	1.83E+02	9.15E+01	2.08E+02	1.47E+02	9.13E+01	6.36E+01
f_{18}	8.84E+02	3.60E+02	3.60E+02	8.98E+02	9.05E+02	9.04E+02	9.41E+02	8.70E+02	9.13E+02
f_{19}	8.78E+02	3.25E+02	3.20E+02	9.01E+02	8.85E+02	9.04E+02	9.38E+02	9.13E+02	9.13E+02
f_{20}	8.63E+02	3.43E+02	3.40E+02	8.96E+02	9.05E+02	9.04E+02	9.28E+02	9.09E+02	9.13E+02
f_{21}	7.94E+02	4.71E+02	5.00E+02	5.12E+02	5.00E+02	5.00E+02	5.00E+02	6.62E+02	7.05E+02
f_{22}	7.53E+02	7.46E+02	7.28E+02	8.80E+02	8.43E+02	8.17E+02	9.14E+02	8.63E+02	8.19E+02
f_{23}	8.88E+02	5.58E+02	5.59E+02	5.34E+02	5.34E+02	5.34E+02	5.39E+02	8.12E+02	7.30E+02
f_{24}	2.28E+02	2.00E+02	2.00E+02	2.00E+02	2.00E+02	2.00E+02	2.00E+02	2.00E+02	2.00E+02
f_{25}	4.55E+02	4.00E+02	4.03E+02	2.14E+02	2.10E+02	2.09E+02	2.21E+02	2.14E+02	2.13E+02
V.S.	(3, 4, 18)†	(8, 8, 9)		(7, 4, 14)†	(7, 6, 12)		(5, 2, 18)†	(6, 4, 15)†	
Optima	4	7	**9**	3	3	**7**	2	2	**7**
Rank	2.52	1.78	**1.7**	2.26	1.98	**1.76**	2.42	2.02	**1.56**

† denotes there is a significant difference over the distribution of mean errors between the corresponding algorithm with iCMA-ES-tsc by a two-sided Wilcoxon matched-pairs signed-ranks test at the 0,05 α-level.

Table D.6: Summary of the comparison with iCMA-ES-tsc on 10, 30 and 50 dimensions with respect to mean error values: (better, equal, worse). Error values lower than 10^{-8} are approximated to 10^{-8}.

	iCMA-ES-05 vs. iCMA-ES-tsc	Sep-iCMA-ES-tsc vs. iCMA-ES-tsc
10 Dim	(6, 10, 9)	(5, 4, 16)†
30 Dim	(4, 11, 10)†	(3, 6, 16)†
50 Dim	(7, 6, 12)	(3, 4, 18)†

† denotes there is a significant difference over the distribution of mean errors between the corresponding algorithm and iCMA-ES-tsc according to a two-sided Wilcoxon matched-pairs signed-rank test at the 0,05 α-level.

Table D.7: Given are for each algorithm the number of optima reached and the average rank on the CEC'05 benchmark problems of dimension 50. We use the Friedman test at significance level $\alpha = 0.05$ is used. ΔR_α is the minimum significant difference between the ranks of algorithms. The numbers in the last column are the differences of the sum of ranks relative to the best algorithm; if a difference is larger than ΔR_α, it is statistically significant.

ΔR_α	Algs	OptNum	Rank	ΔR
19.75	iCMA-ES-tsc	7	2.9	0
	iCMA-ES-$\oplus$	4	2.9	0
	iCMA-ES-uni	7	3.2	7.5
	iCMA-ES-dp	5	3.7	20.0
	iCMA-ES-uni,multi	6	3.8	22.5
	iCMA-ES-multi	4	4.5	40.0

Table D.8: Given are the tuned parameter settings for different subsets of the training functions taken from the SOCO benchmark set.

Factor	iCMA-ES			
	-$\oplus$	-uni,multi	-uni	-multi
a	8.349	5.977	8.419	3.235
b	1.647	3.749	2.324	4.537
c	0.5325	0.4666	0.5955	0.4582
d	3.809	2.571	2.499	1.618
e	-13.01	-10.87	-8.583	-8.178
f	-6.217	-8.703	-11.57	-14.83
g	-8.37	-17.46	-19.91	-19.61

Appendix E

Artificial bee colonies for continuous optimization: Experimental analysis and improvements

E.1 Introduction

The artificial bee colony (ABC) algorithm is a recent swarm intelligence algorithm that is loosely inspired by the foraging behavior of a honeybee swarm. It is one representative of a number of algorithmic approaches that exploit some type of behavior found in real bee colonies for tackling computational problems arising in computer science and optimization [Diwold et al., 2011b, Karaboga and Akay, 2009]. ABC was introduced by Karaboga [2005] applying it to continuous optimization problems. The algorithm can be summarized as follows. There are three types of (artificial) bees, namely employed bees, onlooker bees, and scout bees. Each employed bee is associated to a different food source—a food source corresponding to a solution of the optimization problem. At each iteration of the algorithm, an employed bee explores the neighborhood of the food source (solution) to which it is associated. Onlooker bees also explore the neighborhood of food sources; however, differently from employed bees, they are not associated to one fixed food source, but they choose the food source they explore in each algorithm iteration probabilistically based on its quality—corresponding to the quality of a solution in the optimization problem. If the neighborhood of a food source has been explored unsuccessfully for a given number of times (corresponding to a depleted food source), a new food source is chosen by a scout bee, which in the ABC algorithm corresponds to generating a random solution in the search space. In a sense, the random choice of a solution by a scout bee implements a form of partial restart, thus, adding an exploration feature to the algorithm.

The original ABC algorithm obtained encouraging results on some standard benchmark problems, but, being an initial proposal, still a considerable perfor-

mance gap with respect to state-of-the-art algorithms was observed. In particular, it was found to be relatively poor performing on composite and non-separable function as well as having a slow convergence rate towards high quality solutions [Akay and Karaboga, 2012]. Therefore, it is not surprising that in the following years, several modifications of the original ABC algorithm were introduced trying to improve performance. Unfortunately, so far there is no comprehensive comparative evaluation of the performance of ABC variants on a significantly large benchmark set available. Such a comparison would be useful for a number of reasons. Firstly, the selected benchmark functions differ between the papers; secondly, some papers test their algorithms on only a small set of benchmark problems; thirdly, the experimental settings differ in part quite substantially among the papers and several papers do not use optimum thresholds to avoid unreasonably low numbers; fourthly, some variants were tested with benchmark functions that have their optimum point in zero and possibly use features that exploit this fact; finally, no experiments are available that examine the scaling behavior of the algorithms across different dimensionalities. In addition, observed differences in few comparisons may be due to a different effort that has been spent on tuning algorithm parameters.

In this chapter, we try to fill this perceived gap. We first review in detail several variants of the original ABC algorithm. We have re-implemented these ABC variants under a same framework to allow their experimental study under same experimental conditions. For the evaluation of the ABC algorithms, we have chosen a recent benchmark set of continuous optimization problems that has been used as benchmark set for a special issue of the Soft Computing journal on the scalability of evolutionary algorithms and metaheuristics to large-scael continuous optimization problems [Lozano et al., 2011]. This benchmark set comprises 19 freely scalable functions that includes classical continuous optimization functions as well as combined functions [Herrera et al., 2010]; subsets of this benchmark set have also been used at earlier algorithm competitions for continuous optimization problems.

We evaluate the various ABC algorithms under three experimental conditions. First, we apply each of the algorithms using parameter settings that can be considered as default parameter settings from the literature. Second, we tune the parameters of each ABC variants using an automatic algorithm configuration tool, Iterated F-race [Balaprakash et al., 2007, Birattari et al., 2010] as implemented in the irace package [López-Ibáñez et al., 2011]. The rationale for this step is to examine the performance of the algorithms under a same tuning effort and, thus,

to eliminate the possible bias that is introduced by an uneven tuning effort for the algorithm variants in the original papers. Third, we consider the integration of local search algorithms into the ABC variants. In fact, many recent state-of-the-art algorithms for continuous optimization combine an effective local search mechanism with population-based search techniques [LaTorre et al., 2011, Molina et al., 2010a] and, thus, it is natural to consider this possibility also here. In this final step, we again use the help of automatic algorithm configuration tools to avoid a manuel re-design of the hybrid algorithms and to ensure an even tuning of the various ABC hybrid algorithms.

As maybe expected, the tuned ABC variants and the ABC variants that integrate local search significantly improve in several cases over the original ABC variants. In particular, the usage of a local search diminishes the differences among the various ABC algorithms. As a result, after the automatic tuning and the integration of the local search algorithms, the ranking of the ABC variants changes. This illustrates the necessity of a thorough design and tuning of the ABC algorithms before making conclusions about the relative performance of ABC variants or before actually claiming superior performance of a new variant over a basic variant.

The chapter is structured as follows. In Section E.2, we introduce the original ABC algorithm, its variants, and summarize the type of modifications available ABC variants introduce. Section E.3 describes the experimental setup and the experimental results are discussed in Section E.4. We conclude in Section E.5.

E.2 Artificial bee colony algorithm

In this section, we first introduce the original ABC algorithm, present in some detail the variants we study in the chapter, and give an overview of other ABC variants.

E.2.1 Original ABC algorithm

In a bee colony, each bee has a specialized task and a main aim of a colony is to maximize the amount of nectar in the hive. For this aim, bees use some form of division of labor and self-organization. Inspired by the foraging behavior of bee colonies, the artificial bee colony (ABC) algorithm was proposed by Karaboga [2005], Karaboga and Basturk [2007]. ABC was tested using its application to continuous function optimization and it is one of several algorithmic techniques that

Algorithm 6 The pseudo-code of the Artificial Bee Colony Algorithm

```
initialization
while termination condition is not met do
  Employed Bees Step
  Onlookers Step
  Scout Bees Step
end while
```

have recently been proposed based on some inspiration from bee colony behavior [Diwold et al., 2011b, Karaboga and Akay, 2009].

The ABC algorithm is a population-based optimization algorithm where three types of bees are used: employed bees, onlooker bees and scout bees. These bees try to find new, promising food sources or exploit already existing ones. In ABC, an analogy is made between the location of food sources and solutions for the problem to be solved and each food source corresponds to a solution of the problem. Furthermore, the amount of nectar at a food source location corresponds to the solution quality or fitness value of a solution to the problem. A particular choice in ABC is that the number SN of employed bees is equal the number of onlooker bees and that this number is the equal to the number of food sources (solutions) currently being exploited; SN is a parameter that is also called the population size. Employed bees and onlooker bees both exploit current food sources (solutions) by visiting its neighborhood. Their role, however, is slightly different. While there is a one-to-one correspondance between employed bees and food sources, that is, each employed bee is assigned to a different food source, the onlooker bees select randomly the food source to exploit, preferring better quality food sources. Scout bees explore the area for new food sources (solutions) if current food sources are deemed to be depleted. Algorithmically, this is implemented by associating to each food source a *limit* value. If more than *limit* times an employed bee or an onlooker bee has visited unsuccessfully a food source, a scout bee searches for a new (randomly located) food source.

An ABC algorithm consists of four phases: the initialization step, the search steps done by the employed bees, onlooker bees and scout bees in this sequence. In the initialization step of the algorithm, the initial food source locations are generated and other parameters of the algorithm are initialized. After initialization, the main loop is executed, which is comprised of the other three steps. Algorithm 6 gives a pseudo-code of these main steps. The details of these steps are explained next.

For the remainder of the chapter, we use an optimization-based terminology

to describe the ABC algorithm. In this chapter, we are dealing with continuous optimization problems, in which we are given a D-dimensional function $f : X \subseteq \mathbb{R}^D \rightarrow \mathbb{R}$, where X is the search space and D is the dimension of the search space. We assume here box-constrained problems where for each solution vector $x_i = (x_{i1}\, x_{i2}\, \ldots\, x_{iD})$ we have that $x_{ij} \in [x_j^{min}, x_j^{max}]$, where $[x_j^{min}, x_j^{max}]$ is the interval of feasible values in dimension $j, 1 \leq j \leq D$.

Initialization: In the algorithm initialization, SN solutions are chosen uniformly at random in the search space. For each solution $x_i, 1 \leq i \leq SN$, this is done by generating each solution's coordinate as

$$x_{i,j} = x_j^{min} + \varphi_{i,j}(x_j^{max} - x_j^{min}) \tag{E.1}$$

where $\varphi_{i,j}$ is a random number generated uniformly at random in $[0, 1]$. This generation of values is done for each dimension $j \in \{1, 2, \ldots, D\}$ independently.

Employed bees behavior: Each of the employed bees chooses deterministically its corresponding reference solution x_i and generates a solution in the vicinity of it. A new candidate solution x_i' is generated by modifying randomly one coordinate of x_i. To do so, another reference solution $x_k, k \neq i, k \in \{1, 2, \ldots, SN\}$ is chosen uniformly at random and x_{ij}' is set to

$$x_{i,j}' = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}),\ i \neq k, \tag{E.2}$$

where $j, j \in \{1, 2, \ldots, D\}$, is a dimension selected uniformly at random and $\phi_{i,j}$ is a random number chosen uniformly at random in $[-1, 1]$. If the new candidate solution x_i' is better than the reference solution x_i, then x_i' replaces x_i. In the original ABC algorithm the neighborhood search of an employed bee only changes the value of one dimension of a reference solution, which is claimed to be responsible for a possibly slow convergence of the original algorithm [Akay and Karaboga, 2012].

Onlooker bees behavior: Onlooker bees probabilistically select a solution and then explore its neighborhood using Equation E.2, that is, they search the neighborhood of a chosen solution in the same way as employed bees. The probabilistic choice of the solution to be explored depends on the solution quality, preferring better quality solutions. The selection probability p_i of a solution x_i is calculated as

$$p_i = \frac{fitness_i}{\sum_{n=1}^{SN} fitness_n}, \tag{E.3}$$

where $fitness_i$ is an evaluation function value assigned to x_i, which for minimization problems is defined as

$$fitness_i = \begin{cases} \frac{1}{1+f(x_i)}, & f(x_i) \geqslant 0, \\ 1 + abs(f(x_i)), & f(x_i) < 0, \end{cases} \tag{E.4}$$

Since better quality solutions have a higher probability of being selected, this step implements some type of intensification mechanism that prefers to examine more carefully the neighborhood of better quality solutions.

Scout bees behavior: If employed and onlooker bees cannot improve a solution for a given number of *limit* times, the solution reaches its visiting limit and they are abandoned. In this case, scout bees try to find a new food source in replacement of the abandoned food source. A new food source location is determined uniformly at random in the search space, that is, a scout bee applies the solution initialization as defined by Equation E.1. This abandoning and scouting mechanism increases the exploration capabilities of the algorithm. In this chapter, the value of *limit* is determined by the formula $lf \cdot SN \cdot D$, where lf is a real-valued parameter to be set.

E.2.2 Variants of the artificial bee colony algorithm

Starting from the original ABC algorithm, several variants of the algorithm have been proposed with the goal of improving its performance. In this chapter, we examine experimentally eight of these ABC variants. These were chosen among a larger number of ABC variants since they either were published in scientific journals or in the original papers they were shown to have experimental results that may make them potentially competitive with state-of-the-art heuristics. In what follows, we shortly describe the main features of the variants we tested.

Modified Artificial Bee Colony (MABC) Algorithm. Akay and Karaboga [2012] motivate their modified ABC (MABC) algorithm by the rather slow convergence speed of the original ABC algorithm, which they say is due to the fact that at each step of the employed and the onlooker bees only one dimension of a reference solution is changed. To overcome this problem, they introduced two

main modifications to the original ABC algorithm. First, they allow to modify a larger number of variables at each employed and onlooker bee step referring to this as increasing the perturbation frequency. To this aim, they introduce a parameter *MR* (modification rate), which gives the probability with which each variable x_{ij} of a reference solution x_i is modified. This is implemented by the equation

$$x_{ij} = \begin{cases} x_{ij} + \phi_{ij} \cdot (x_{ij} - x_{kj}) & \text{if } r_{ij} < MR \\ \\ x_{ij} & \text{otherwise,} \end{cases} \tag{E.5}$$

where r_{ij} is a random number that is drawn uniformly at random in $[0, 1]$. Hence, with a probability *MR* a variable undergoes variation and the particular value of *MR* has a strong impact on the search behavior. In particular, low values of *MR* favor exploitation, while large values of *MR* may lead to more exploration since more dimensions of a solution are modified.

The second modification concerns the magnitude of a change in one variable. In MABC, the modification of a variable is biased by a random number ϕ_{ij} drawn uniformly at random in the interval $[-SF, SF]$, where *SF* is a parameter called scaling factor. The original Equation E.2 is obtained by setting *SF* to one. Instead of using a fixed value for *SF*, Akay and Karaboga [2012] propose to adapt the value of *SF* at run-time using the Rechenberg's 1/5 mutation rule [Rechenberg, 1973]. In particular, every fixed number of iterations the ratio of solution modifications that led to an improvement is computed. Depending on whether this ratio is smaller, larger or equal to one fifth, the value of *SF* is multiplied (that is, decreased) by 0.85, divided (that is, increased) by 0.85, or it remains the same.

Gbest-guided Artificial Bee Colony (GbABC) Algorithm. The main idea of the GbABC algorithm [Diwold et al., 2011a, Zhu and Kwong, 2010] is to bias the modification of the reference solution by the best solution found so far, which we call global-best in what follows. The information of this global-best solution is incorporated into ABC by modifying Equation E.2 to

$$x'_{i,j} = x_{i,j} + \phi_{i,j}(x_{i,j} - x_{k,j}) + \psi_{i,j}(x_{gbest,j} - x_{i,j}),\ i \neq k\,, \tag{E.6}$$

where $x_{gbest,j}$ is the j-th element of the global-best solution, $\psi_{i,j}$ is a random number drawn uniformly at random in $[0, C]$ where C is a constant that is set to one by Diwold et al. [2011a] or used as a parameter to be set by Zhu and Kwong [2010]. This modification is inspired by the usage of the global-best solution to influence

particles in particle swarm optimization; it is a rather straightforward modification that, as we will see later, is important to obtain a significantly improved performance.

GbestDist-guided Artificial Bee Colony (GbdABC) Algorithm. Diwold et al. [2011a] have also proposed a second modification to the ABC algorithm, which uses the global-best solution as in the GbABC variant (with $C = 1$) but which additionally changes the selection of the neighboring solution x_k in Equation E.6. Let $d(x_i, x_k)$ be the Euclidean distance between two solutions x_i and x_k. Then, a solution x_k, $k \neq i$, is chosen with a probability

$$p_k = \frac{\frac{1}{d(x_i,x_k)}}{\sum_{l=1,l\neq i}^{SN} \frac{1}{d(x_i,x_l)}} \tag{E.7}$$

for the update through Equation E.6 instead of being chosen uniformly at random. Thus, the closer a solution x_k is to x_i, the higher is the probability of choosing it. The intuition of this choice is that it is more probable to fine a better solution by searching between two good solutions that are probably also close to each other in the solutions space [Diwold et al., 2011a].

Best-so-far selection Artificial Bee Colony (BsfABC) algorithm. As GbABC and GbdABC, BsfABC [Banharnsakun et al., 2011] exploits the global-best solution (called best-so-far solution in Bsf-ABC) but uses it in a different way. The global-best solution is only used to modify the onlooker bees step, thus the employed bee step is unaffected. The update is defined by

$$x'_{i,d} = x_{i,j} + \mathit{fitness}(x_{best})(\phi_{i,j}(x_{i,j} - x_{gbest,j})),\ i \neq k\ d = 1, 2, \ldots, D\,, \tag{E.8}$$

where j is a randomly selected dimension and $\mathit{fitness}(x_{best})$ is the fitness value of the global-best solution. BsfABC applies the position update in the randomly selected dimension j to each dimension. This has the effect that the variable values in all dimensions of a candidate solution get closer to each other. Such an approach works only well if the optimum point has the same variable values in all dimensions but it breaks down if this is not the case.

BsfABC also modifies the scout bees step. Instead of choosing a random new position, BsfABC introduces a random perturbation of the current food source by using the equation

$$x'_{i,j} = x_{i,j} + x_{i,j}\phi_{i,j}(w_{max} - \frac{itr}{itr_{max}}(w_{max} - w_{min})),\ j = 1, \ldots, D \qquad \text{(E.9)}$$

where w_{max} and w_{min} are control parameters that define the strength of the perturbation, itr is the number of algorithm iterations done so far and itr_{max} is the maximum number of iterations. The effect of this equation is that the strength of the perturbation decreases with an increasing number of iterations, making the algorithm more exploitative in later algorithm iterations. The experiments with the original BsfABC have been done on six benchmark functions, each of which has the global optimum at the position (0, 0, ..., 0) and for these problems BsfABC was shown to be very effective [Banharnsakun et al., 2011].

Chaotic Artificial Bee Colony (CABC) Algorithm. Alataş [2010] introduces two modifications to the original ABC algorithm. The first is to generate the initial solutions by using a chaotic map instead of a standard random number generator; this results in variant CABC_1. Alataş made tests with seven different chaotic maps for three chaotic ABC variants in his paper and here we consider the same chaotic maps in the parameter tuning. The second modification is to generate the solution of a scout bee by using a type of local search, resulting in variant CABC_2. In particular, if in CABC_2 a solution cannot be improved for $limit/2$ trials, the algorithm searches randomly for another $limit/2$ trials around the current solution; in each of the $limit/2$ trials it modifies one dimension and accepts a new solution if it improves over the current one. Finally, a third variant, CABC_3, combines the two modifications proposed for CABC_1 and CABC_2. Here, we only use variant CABC_3, also because variant CABC_1 did not appear to lead to improved results.

Improved Artificial Bee Colony (ImpABC) Algorithm. ImpABC [Gao and Liu, 2011] introduces three modifications to ABC. The first modification concerns the population initialization. The initial positions are generated using a chaotic random generator (as does CABC), which uses a logistic map. Once SN solutions are generated, for each solution the value of each variable is mirrored at the center of the search range for this variable, resulting in SN new solutions; this process is called opposition-based population initialization. From the resulting $2 \cdot SN$ solutions the best SN solutions are kept.

The other new algorithm features with respect to the original ABC algorithm

concern the solution modification. Similar to MABC, Gao and Liu [2011] propose to modify more than one variable. They introduce a parameter M, which refers to the number of variables that should be modified. In their paper, they explore values of M from 1 to D. They observe that the best value of M depends on the particular problem, but do not give a final indication how the parameter M should be set. As default, we therefore set it analogous to the parameter MR of MABC as a probability. This implies that for a same value of MR, the number of variables modified increases with D. The third algorithm feature concerns the solution modification to be applied. Their adaptation is inspired by the mechanisms that are used in differential evolution (DE) [Stern and Price, 1997] algorithms. In particular, the Imp-ABC algorithm proposes two new equations called $ABC/best/1$ and $ABC/rand/1$, which are inspired by the $DE/best/1$ and $DE/rand/1$ schemes in DE, respectively. The equation for $ABC/best/1$ is

$$x'_{i,j} = x_{best,j} + \phi_{i,j}(x_{i,j} - x_{r1,j}) \tag{E.10}$$

and the search equation for $ABC/rand/1$ is

$$x'_{i,j} = x_{r1,j} + \phi_{i,j}(x_{i,j} - x_{r2,j}) \tag{E.11}$$

where $r1$ and $r2$ are two random indices of solutions different from i, $x_{best,j}$ is the value of the variable j of the best solution found so far, and j is a randomly chosen variable. Equation E.10 biases the search towards the global-best solution while Equation E.11 is more explorative. To balance between the effects of the two equations, the authors propose to apply Equation E.11 with a probability p and Equation E.10 with a probability $1 - p$. Based on some limited experiments, they propose to set p to 0.25.

In a more recent paper, Gao et al. [2012] propose essentially the same ideas but use yet another modified search equation, where variable j of a chosen solution x_i is updated analogous to Equation E.10 but considering two or four random positions for updating the variable's position. Since they report that using four random solutions does not lead to better performance, we do not consider this variant here.

Rosenbrock Artificial Bee Colony (RABC). RABC [Kang et al., 2011] proposes two main modifications to ABC. The first replaces Equation E.4 with the rank-based fitness adaptation method defined as

$$fitness_i = 2 - SP + \frac{2(SP-1)(r_i-1)}{SN-1}, \tag{E.12}$$

where $SP \in [1.0, 2.0]$ is the selection pressure and r_i is the rank of solution x_i in the population.

The second modification is the integration of Rosenbrock's rotational direction method (RM) [Rosenbrock, 1960], a local search technique, into ABC. RM is applied every n_c iterations to the best-so-far solution. For integrating RM into ABC, some adaptations have been done to RM. Maybe the most noteworthy from an algorithmic side is the usage of an adaptive initial step size in RM. To do so, the authors propose to use the best m ranked solutions and to compute

$$\delta_j = 0.1 \cdot \frac{\sum_{i=1}^{m}(\bar{x}_{i,j} - x_{gbest,j})}{m}, \tag{E.13}$$

where δ_j is the step size in dimension j and $\bar{x}_i$ is the i-th best solution after ranking. The solution returned by RM replaces the middle ranked solution in the population, which, according to the authors, improves performance when compared to replacing the worst solution on multi-modal problems. For a description of the other modifications to RM, we refer to [Kang et al., 2011].

Incremental Artificial Bee Colony (IABC) with Local Search (IABCLS). Aydın et al. [2012] have proposed an algorithm that integrates the incremental social learning (ISL) framework [Montes de Oca, 2011] and local search procedures to ABC. The basic idea of ISL when applied to population-based algorithms for optimization problems is to start the algorithm with a small population size, and to add new agents after each g iterations (implementing the population growth), biased by members of the population (implementing the learning aspect). When the population is extended, one new employed bee location is generated as

$$x'_{new,j} = x_{new,j} + \varphi_{i,j}(x_{gbest,j} - x_{new,j}) \quad j = 1, \ldots, D \tag{E.14}$$

where $x_{new,j}$ is a variable value generated according to Equation E.1, $x'_{new,j}$ is the location of the new food source biased by the global-best solution, and $\varphi_{i,j}$ is a parameter. Apart from adapting ISL to ABC, IABC uses two other features. The first is to use information on the best-so-far solution for a randomly chosen dimension j, using

$$x'_{i,j} = x_{i,j} + \phi_{i,j}(x_{gbest,j} - x_{i,j}). \tag{E.15}$$

The second biases the scout bees towards the best-so-far solution. This is done by initializing the scout bee using

$$x'_{i,j} = x_{gbest,j} + R_{factor}(x_{gbest,j} - x_{new,j}) \quad j = 1, \ldots, D \tag{E.16}$$

where x'_i is the new food source location replacing an abandoned food source; R_{factor} is a parameter that determines the bias towards the best-so-far solution x_{gbest}; and $x_{new,j}$ is a variable value generated according to Equation E.1. This modification increases the exploitation behavior of the algorithm but it may increase chances of early convergence. Aydın et al. [2012] considered the hybridization of IABC with either Powell's conjugate direction set method [Powell, 1964] or Lin-Yu Tseng's Mtsls1 [Tseng and Chen, 2008] local search procedure resulting into a hybrid IABCLS algorithm. This was done by invoking a local search procedure in every algorithm iteration starting from the best-so-far solution.

Other Variants of Artificial Bee Colony. Since the initial proposal of ABC, a large number of ABC variants for numerical optimization has been proposed. One common theme has been the hybridization of ABC algorithms with procedures taken from other techniques. Fister et al. [2012] proposed memetic ABC algorithm, which is hybridized with two local search heuristics, the Nelder-Mead algorithm (NMA) and the random walk with direction exploitation (RWDE). Additionally, this algorithm takes inspiration from DE in the update equation. Yan et al. [2011] and Ming et al. [2010] hybridized ABC with a genetic algorithm. El-Abd [2011b] and Sharma et al. [2011] investigated the hybridization of ABC and particle swarm optimization, which essentially is done by modifying the update equations of ABC. Another hybrid approach was proposed by Zhong et al. [2011]; they introduce, inspired by the chemotaxis behavior of bacterial foraging, a stronger local search type behavior in the update of the employed and onlooker bees. Tsai et al. [2011] proposed Interactive ABC which introduces the concept of universal gravitation into the consideration of the affection between employed bees and the onlooker bees. El-Abd [2011a] explored the use of opposition-based learning in ABC. Abraham et al. [2012] introduced as others a type of DE strategy into the search equation of ABC. Some other variants emphasized on modifications in search equations of ABC steps. Zou et al. [2010] have proposed cooperative ABC algorithm which selects the global-best solution as the reference food source and mutates it. As chaotic ABC, Wu and Fan [2011] use a chaotic random number generator; however, they use it in the search equation instead of the initialization or the scout step. Alam et al. [2010] introduced an ABC algorithm with exponen-

ABC	original ABC	Karaboga and Basturk [2007]
GbABC	Gbest-guided ABC	Zhu and Kwong [2010] Diwold et al. [2011a]
BsfABC	Best-so-far selection ABC	Banharnsakun et al. [2011]
MABC	Modified ABC	Akay and Karaboga [2012]
ImpABC	Improved ABC	Gao and Liu [2011]
CABC	Chaotic ABC, version 3	Alataş [2010]
GbdABC	GbestDist-guided ABC	Diwold et al. [2011a]
RABC	Rosenbrock ABC	Kang et al. [2011]
IABC	Incremental ABC	Aydın et al. [2012]

Table E.1: Abbreviation, name, and main reference for the nine ABC algorithms compared in our study.

tially distributed mutation that uses exponential distributions to produce mutation steps with varying lengths and they adjust the step length at run time. Sharma and Pant [2012] proposed two new mechanisms for the movements of scout bees. The first method is based on a non-linear interpolated path while in the second one, scout bees follow a Gaussian movement. Rajasekhar et al. [2011] proposed an improved version of ABC algorithm with mutation based on Levy probability distributions. Global ABC algorihm, which was introduced by Guo et al. [2011], proposed three modified search equation by using global and individual best positions. To balance between exploring and exploiting, a diversity stategy have used by Lee and Cai [2011].

E.3 Experimental setup

We have re-implemented the original ABC algorithm and the eight variants we have presented in detail in Section E.2.2. Thus, in our experimental analysis, we will compare a total of nine algorithms. For convenience, we repeat the abbreviation, the name and the main reference to the particular algorithms in Table E.1.

We compare these nine ABC algorithms in three settings. A first setting uses the re-implementation of these algorithms using as much as possible the default parameter settings that were proposed in the original publications. In few cases, not all the default parameter settings have been specified unambigously in the original publications or several values have been tested without giving a final recommendation. In these cases, we have made an educated guess of a reasonable setting or used values that from the reported experimental results seemed to be appropriate.

A second setting considers the usage of an offline automatic algorithm configuration tool. By using an offline tool for parameter tuning and a same tuning effort for each ABC variant we intend (i) to make a fair comparison among the various ABC algorithms that is independent of the tuning spent by the original authors and (ii) to check whether and by how much the performance of ABC algorithms can be improved without changing any detail of their algorithmic structure.

In a third setting, we hybridize the ABC algorithms with local search procedures that are used to refine some of the generated solutions. For defining appropriate parameter settings, we directly tune the parameters of the resulting hybrid ABC algorithms using again an automatic algorithm configuration tool for deriving appropriate parameter settings. Note that we re-tune all the hybrid algorithms since there may be an interaction between the algorithm parameters and the fact that a local search is used and without re-tuning suboptimal performance may result.

E.3.1 Benchmark set

All experiments were performed on the benchmark functions that were proposed for a special issue of the Soft Computing journal on the scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems [Lozano et al., 2011]. This benchmark set, to which we refer as SOCO, contains 19 benchmark functions comprising well-known, widely used functions such as Rastrigin, Griewank, or Ackley as well as hybrid functions composed of two basic functions [Herrera et al., 2010]. A detailed description of the benchmark function set is given in the Table 2.2, page 26. All functions are freely scalable and here we conducted our experiments using functions of dimension $D \in \{10, 50, 100, 500\}$ to examine the scaling behavior of ABC algorithms. All algorithms were run 25 independent times on each SOCO function, each algorithm run being for a maximum of $5000 \cdot D$ function evaluations, as proposed in the original protocol of the SOCO benchmark set. As suggested by Herrera et al. [2010], error values lower than 10^{-14} are approximated to 10^{-14}, where the error value is defined as $f(x) - f(x^*)$ with x being a candidate solution and x^* being the optimal solution. (10^{-14} is therefore also referred to as optimum threshold.) To analyze the results, were necessary we used the non-parametric Wilcoxon test at a significance level $\alpha = 0.05$ and Holm's multiple test corrections if more than two algorithms are compared.

E.3.2 Local search

Over the past years, it has become very popular to include local search techniques into metaheuristics to improve their performance. It is also known that many high performing state-of-the-art algorithms for continuous optimization rely on the usage of effective local search algorithms. An example is the MOS-DE algorithm, the algorithm that was identified in the special issue as the best performing one on the SOCO benchmark functions [LaTorre et al., 2011]. Other examples include memetic algortihms [Molina et al., 2010a] or hybrid ant colony optimization algorithms [Liao et al., 2011b]. Also, few ABC algorithms such as IABC and RABC have made use of additional local search procedures. Given our previous experience, we believe that local search algorithms are useful to improve performance of ABC algorithms and may make them competitive with state-of-the-art algorithms. To support this idea, we have re-desinged the nine ABC variants by hybridizing them with a local search procedure and we have automatically tuned the parameters of the resulting hybrid algorithms.

For this hybridization, we have considered Powell's conjugate direction set method [Powell, 1964] and the Mtsls1 local search algorithm [Tseng and Chen, 2008]. The reason for selecting these methods is that they are relatively simple to implement but still rather powerful local search algorithms. Additionally, in some of our previous research, we were able to design with these two local searches some very high performing continuous optimization algorithms [Aydın et al., 2012, Liao et al., 2011b, Montes de Oca et al., 2011], making them also a natural choice for this task here. In fact, from our work on IABC, we already knew that both local search algorithms may contribute positively to performance. Because of the effectiveness of the hybridization mechanisms we have used previously, we followed the same design strategy which consists of the following design decisions.

- The local search procedure is applied to the global-best solution at each iteration. If the local search improves the solution, the best-so-far solution is updated. If $Failure_{max}$ successive local search applications fail to improve the global-best solution, local search is applied to a randomly chosen solution different from the global-best one.

- There are two options for determining the step size parameter of the local search: adaptive or fixed step size. In an adaptive step size, the maximum norm of the difference vector between a randomly selected solution in the population and the best-so-far solution is used. For the second option, half

the length of the search range is selected as the step size, which is not changed during progress.

- The number of iterations and other parameters related to the local search procedures are determined by using an automatic parameter configuration tool for each algorithm.

It is important to note that there is a trade-off between global and local search. If the effect of the local search is too strong, the impact of the ABC search mechanism may be considered insignificant after hybridization. To check for this possibility, we have implemented a random-restart local search algorithm (RLS), automatically tuned the local search parameters using the same tuning effort as spent for configuring the hybrid ABC algorithms, and then compared the resulting algorithms.

E.3.3 Tuner setup and parameter settings

For tuning the parameters of the nine ABC algorithms we used the iterated F-race procedure [Balaprakash et al., 2007, Birattari et al., 2010] implemented in the irace package [López-Ibáñez et al., 2011]. Iterated F-race is an offline automatic algorithm configuration tool. It iteratively applies F-race [Birattari et al., 2002] to a set of configurations that are generated using a probabilistic model. F-race is a racing method for selecting the best from a set of configurations. At each step of F-race, all surviving configurations are evaluated on a new benchmark function. The obtained error values are then ranked and the F-test checks after each step whether a significant difference arises between the configurations. If yes, the worst configurations are eleminated from the race using Friedman post-tests. F-race stops once only a single (or very few) configurations remain in the race. Then iterated F-race samples a new set of configurations; this sampling of the configurations is biased iteration by iteration more strongly around the best configurations found so far. The new configurations are the input for another F-race and the overall iterated F-race procedure terminates once a maximum number of algorithm runs, which corresponds to the tuning budget, has been executed.

Iterated F-race allows to tune real-valued, ordinal, and categorical parameters and it has been widely used in a number of tuning tasks. As input, iterated F-race requires the parameters to be tuned, their ranges and problem instances that are used as a training set in the tuning. The set of the parameters of the ABC algorithms, their ranges and types are listed in Table E.2. In the tuning, at most

Table E.2: Summary of ABC algorithm parameters, their type and the ranges that are considered in the tuning. All parameters are either of type numerical (N) or categorical (C); numerical parameters may be integers (int) or real numbers (real). For each parameter is given the range that is considered in the tuning and it is specified in which ABC algorithms the respective parameters occur.

General parameters

Para.	Type	Range	algorithm	Para.	Type	Range	algorithm
SN	N (int)	[5, 100]	all ABC	lf	N (real)	[0, 3]	all ABC

Local search related parameters

Para.	Type	Range	algorithm	Para.	Type	Range	algorithm
LS	C	Powell, Mtsls1	all ABC	ls_{itr}	N (int)	[1, 100]	all ABC
$Failure_{max}$	N (int)	[1, 20]	all ABC	SP	N (real)	[1, 2]	RABC
RM_{itr}	N (int)	[1, 100]	RABC	m	N (int)	[1, 100]	RABC

ABC algorithm specific parameters

Para.	Type	Range	algorithm	Para.	Type	Range	algorithm
MR	N (real)	[0, 1]	MABC, ImpABC	C	N (real)	[0, 4]	GbABC,GBdABC
chaoticMap	C	7 maps	CABC, ImpABC	p	N (real)	[0, 1]	ImpABC
SF	N (real)	[0, 1]	MABC	$adaptSF$	C	Yes / No	MABC
w_{min}	N (real)	[0, 0.5]	BABC	w_{max}	N (real)	[0.5, 1]	BABC
SN_{max}	N (int)	[10, 100]	IABC	R_{factor}	N (real)	$[10^{-14}, 0]$	IABC
g	N (int)	[1, 20]	IABC				

5 000 runs of an ABC algorithm are executed, that is, 5 000 is the tuning budget. As training instances, we use the 19 SOCO benchmark functions of dimension 10. First, a random order of the benchmark functions is determined and then at each step of F-race a benchmark function is used to evaluate the surviving configurations. Once all 19 benchmark functions have been tested, we re-use the benchmark functions in the same random order. The default parameter settings and the parameter settings that are obtained by the tuning are summarized in Table E.3.

The tuning method we use ranks the results of the different configurations. Due to the ranking, the magnitude of the differences between the configurations is not considered but only whether one configuration gives a better result as another one. This is different from using the absolute obtained error values for considering the significance of differences between configurations. The absolute errors values would, for example, be taken into account when using t-race, a racing method implemented in the irace package that makes use of Student's t-test instead of rank-based tests and that would tend to search for configurations with best mean performance. In fact, the biases between using rank-based tests and tests considering absolute error values were examined by Smit and Eiben [2010], where it is

shown that the evaluation method used in tuning (e.g. averages vs. ranks) has, as expected, an impact on the finally tuned configurations. Knowing about the possible biases, we opted here for the rank-based tests since a disadvantage of using averaging is that the computational results would have to be rescaled, as done by Smit and Eiben [2010] to account for differences in the order of magnitude of error values; otherwise the performance on very few functions would dominate the tuning. Given the possible bias incurred by ranking, we evaluate here the ABC variants according to their median performance; nevertheless, information on the full distribution of the algorithms results will be given in the form of solution quality distributions and run-time distributions on the chapter's supplementary pages `http://iridia.ulb.ac.be/supp/IridiaSupp2013-002`.

E.4 Experimental results and analysis

In this section, we presented the results and analysis of our experiments and discuss specific observations for the various ABC variants and we report on some additional experiments.

E.4.1 Main comparison

As a first step, we present the overall results of our comparison of various ABC variants. A main result of our experimental analysis is that the tuning and the addition of a local search phase to ABC algorithms has a major impact on the performance of the various ABC variants.

A summary of the overall results can be obtained from Figures E.1 to E.3. Figures E.1 and E.2 give box-plots that indicate the distribution of the median error values that each studied ABC variant obtained on the 19 benchmark functions. The number on top of each box indicates for how many functions the achieved median error value is below the optimum threshold. A "+" ("−") symbol on the top of a bar indicates that an ABC variant is performing significantly better (worse) than the original ABC algorithm. The detailed numerical data on which these box-plots are based are given in Tables E.4 to E.7; further data are available on the chapter's supplementary pages `http://iridia.ulb.ac.be/supp/IridiaSupp2013-002`. Additionally, we give in Figure E.3 the ranking of the variants across the various dimensions (from top to bottom, the results are given for 10, 50, 100, and 500 dimensions) for default parameters (left column), tuned parameter settings (middle column) and for the tuned ABC variants with

Table E.3: Summary of the ABC algorithm parameter settings. Given are for each ABC algorithm and its default, tuned, and tuned-with-local search parameter settings in a three field notation as *default / tuned / tuned+LS*. The parameter settings of the local search used by the ABC variants are given below the other parameters.

Alg.	SN	lf	Others					
ABC	62 / 8 / 37	1.0 / 2.734 / 2.982	—					
GbABC	15 / 12 / 40	1.0 / 1.12 / 1.171	C	1.0 / 1.507 / 2.069				
BsfABC	100 / 6 / 64	0.1 / 2.164 / 1.962	w_{min}	0.2 / 0.3327 / 0.2454	w_{min}	1.0 / 0.725 / 0.5843		
MABC	10 / 11 / 25	1.0 /1.978 / 0.2818	MR	0.4/ 0.774 /0.2743	SF	1.0 / 0.9707 / 0.6737	$adaptSF$	No / No / No
ImpABC	25 / 28 / 35	1.0 / 1.62 / 1.58	MR	1.0 / 0.4108 / 0.7269	p	0.25 / 0.4686/ 0.3837		
CABC	10 / 17 / 54	1.0 / 2.819 / 0.4039	chaoticMap	1 / 3 / 7				
GbdABC	15 / 11 / 84	1.0 / 2.031 / 2.03	C	1.0 / 2.176 / 1.469				
RABC	25 / 10 / 37	1.0 / 2.089 / 1.023	SP	1.5 / 1.864 / 1.618	RM_{itr}	15 / 17 / 34	m	5 / 2 / 64
IABC	5 / 6 / 6	1.0 / 2.272 / 0.0619	SN_{max}	50 / 12 / 93	R_{factor}	-1 / -3.47 / -3.868	g	1 / 12 / 5

Local search parameters

Parameter	ABC	GbABC	BsfABC	MABC	ImpABC	CABC	GbdABC	RABC	IABC	RLS
Local Search	Mtsls1	Mtsls1	Mtsls1	Mtsls1	Mtsls1	Mtsls1	Mtsls1	Mtsls1	Mtsls1	Mtsls1
LS_{itr}	76	81	76	61	4	66	59	88	85	56
$Failure_{max}$	1	5	1	20	9	11	9	2	12	—

local search (right column). To obtain these ranks, the results of all variants on a same benchmark function were ranked from best (rank 1) to worst (rank 9) and in Figure E.3 are given the so obtained average ranks for each algorithm.

Considering the box-plots for the default parameter settings in dimension 10 (top left plot in Figure E.1), we can observe that all variants except BsfABC reach typically lower median errors that the original ABC algorithm, which is confirmed by the average ranks (top left plot in Figure E.3). (The reasons for the poor performance of BsfABC are given in the next Section.) The overall best performing ABC variants in this setting and dimension are GbABC and GbdABC, which indicates that a strong emphasis on the global-best solution is important on the SOCO benchmark set.

The situation changes a bit when comparing the results obtained by the tuned parameter settings for dimension 10 (middle plots on top in Figures E.1 and E.3). In that case, the potential advantage of the ABC variants over the original ABC diminishes, with the original ABC reaching a ranking similar to that of MABC, BsfABC, and CABC. In addition, the gap to GbABC and GbdABC, which remain among the best performers, is reduced. Nevertheless, several ABC variants still reach statistically significantly better results than the original ABC algorithm, even if according to the rank information the performance gap is reduced. Among the tuned versions, ImpABC is the second best ranking, while for default settings it was the fifth ranked. The introduction of an effective local search procedure reduces further the gaps between the ABC variants and none of the variants emerges as a clear winner. Particularly high performance reaches ImpABC, which obtains on 15 of the 19 benchmark problems a median error value that is lower than the optimum threshold. Nevertheless, once local search is added no ABC variant reaches a statistically significanltly improved performance over the original ABC hybridized with local search.

Considering the scaling behavior of the ABC variants from the ten dimensional problems up to the 500 dimensional ones, the most noteworthy result is the poor scaling behavior of MABC and ImpABC, which are for high dimensional problems among the worst ranked ABC algorithms. In fact, considering default parameter settings, ImpABC ranks worse than ABC for dimensions 50, 100, and 500 and MABC ranks worse than ABC for dimension 500; often, the observed differences are also statistically significant. BsfABC remains the worst ranking ABC algorithm. Taking into account the differences in the absolute values as indicated by the boxplots or the numerical results in Tables E.4 to E.7, the differences in solution quality are substantial. The same poor scaling behavior for ImpABC, MABC

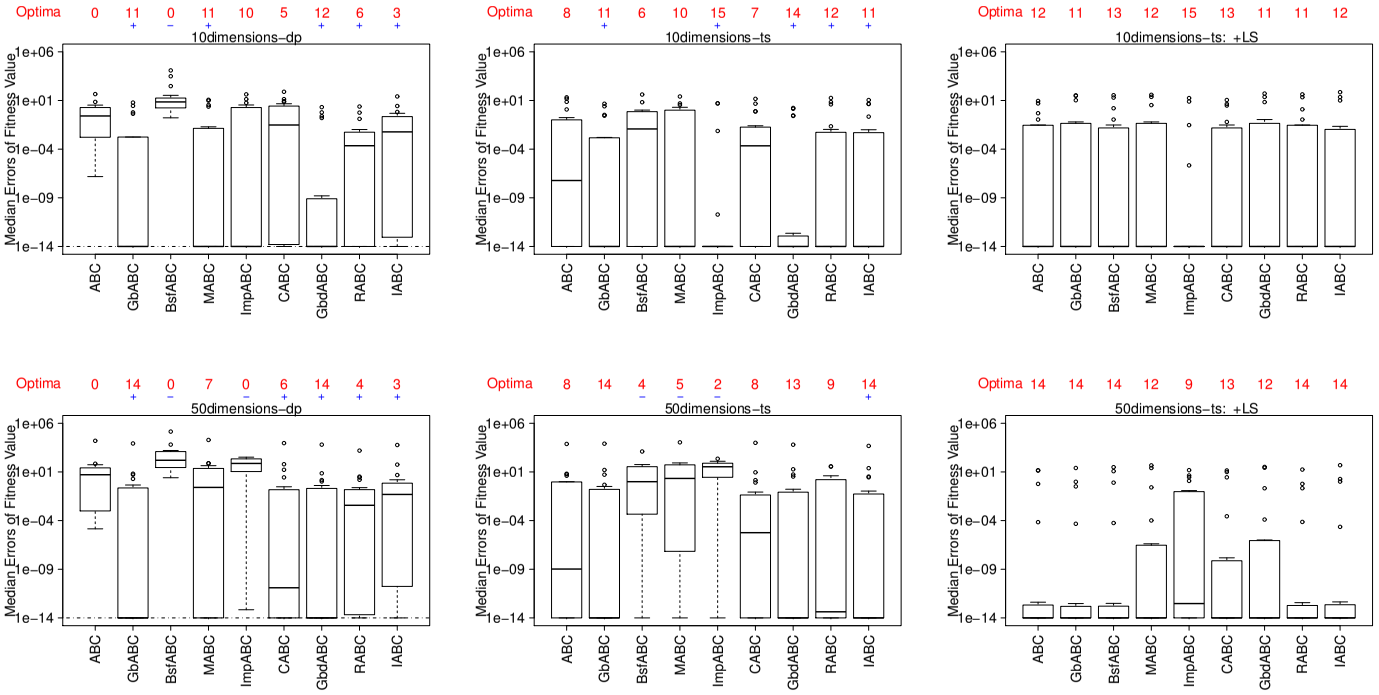


Figure E.1: Boxplots of the distribution of the median solution quality reached by each of the studied ABC variants on the 19 SOCO benchmark problems. The number on top of each box gives the number of functions on which the median error found was below the optimum threshold of 10^{-14}. The boxplot on the top row are for functions of dimension 10; the boxplots on the bottom row are for functions of dimension 50. A "+" or "−" symbol on top of a box-plot denotes a significant difference detected by a Wilcoxon's test at the 0.05 level with Holm's correction for multiple testing between the results obtained with indicated algorithm and the original ABC algorithm (first box)).

and BsfABC is observed for the tuned parameter settings. In section E.4.2, we identify the reason for this poor scaling behavior and indicate the remedy. GbABC and GbdABC remain among the best performing ABC algorithms also for higher dimensional problems, showing that the emphasis on the global-best solution is important for a good scaling behavior. Interestingly, for higher dimensional problems, IABC algorithm becomes the best ranking one and it is one of the ABC algorithms that profit most from tuning. Finally, considering the ABC variants with local search, their performance levels remain comparable and no statistically significant differences in favor of any of the ABC variants over the original ABC with local search are detected.

The main results of our comparison of ABC variants can be summarized as follows.

- Tuning has a major impact on the comparison results. In fact, especially on the higher dimensional benchmark problems several of the proposed ABC algorithm variants do not give a statistically significant improvement over the original ABC algorithm, once one considers reasonably tuned parameter settings.
- The introduction of a local search smoothes the differences between the different ABC variants and poor performing ones without local search become very competitive to the best ABC variants.
- Some of the variants such as BsfABC perform surprisingly poor when compared to the original papers. Some of the variants appear to have poor scaling behavior.

In the following section, we revist some of the above made conclusions and provide a more detailed analysis of several ABC variants.

E.4.2 Detailed analysis of ABC algorithms

Improvement over random restart local search

The best performing ABC algorithms typically include local search, as shown in the previous section. Hence, a first question that arises is whether the ABC algorithms contribute significantly to performance or whether the local search alone is enough to reach a same performance level. We explore this question by comparing the hybrid ABC algorithms to a random restart local search (RLS) algorithm, where

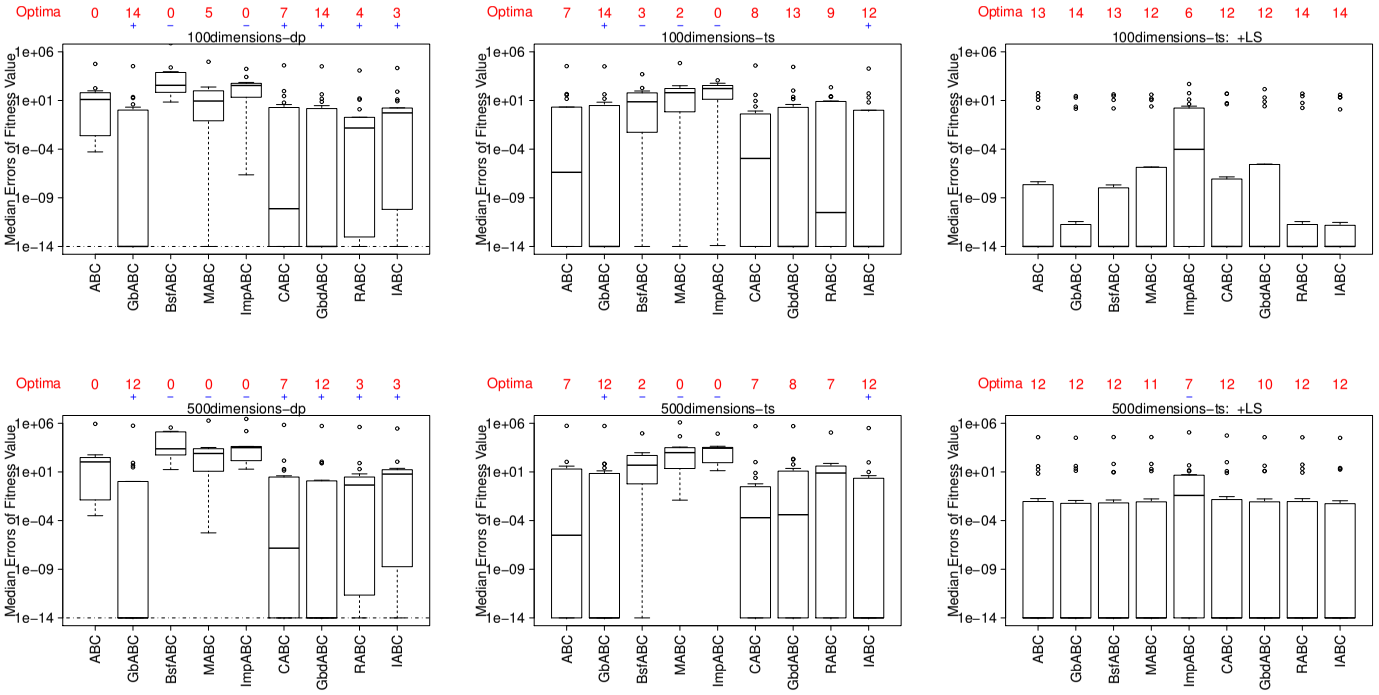


Figure E.2: Boxplots of the distribution of the median solution quality reached by each of the studied ABC variants on the 19 SOCO benchmark problems. The number on top of each box gives the number of functions on which the median error found was below the optimum threshold of 10^{-14}. The boxplot on the top row are for functions of dimension 100; the boxplots on the bottom row are for functions of dimension 500. A "+" or "−" symbol on top of a box-plot denotes a significant difference detected by a Wilcoxon's test at the 0.05 level with Holm's correction for multiple testing between the results obtained with indicated algorithm and the original ABC algorithm (first box)).

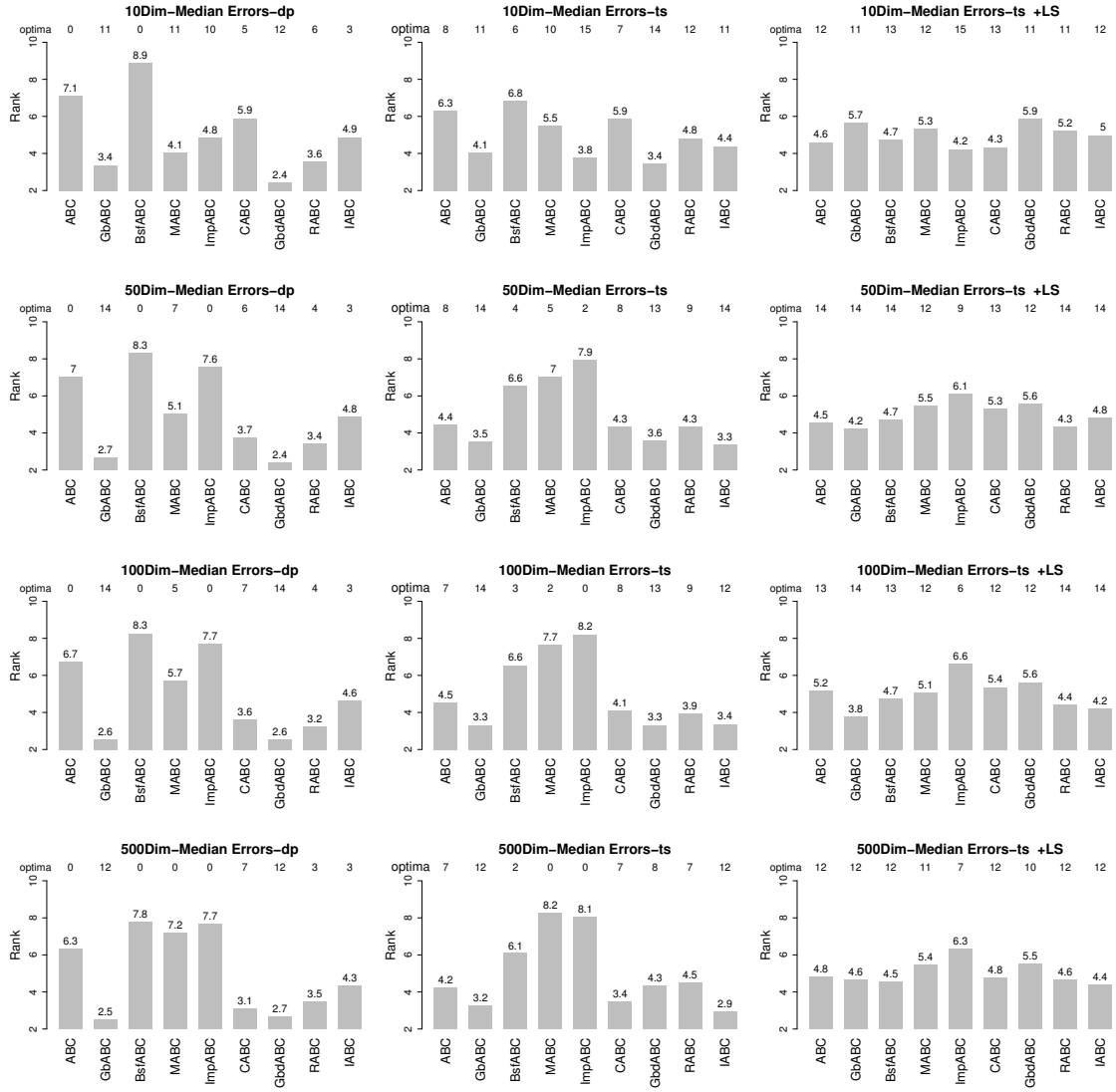


Figure E.3: Performance comparisons by the means of the average rank over 19 functions. The ranking was based on the median error value of obtained by each algorithm on the 19 SOCO benchmark functions.

the starting points for the local search are generated uniformly at random in the search space. To make the comparison fair, we tuned the parameters of the RLS algorithm, which are the parameters of the Mtsls1 local search algorithm [Tseng and Chen, 2008], using the same procedure and tuning budget as used when tuning the ABC algorithms. The tuned parameter settings of RLS are given in Table E.3, page 195.

Figure E.4 summarizes the results of this comparison across several dimensions of the benchmark functions. A "−" symbol on top of a box denotes that RLS performs statistically significantly worse than the respective ABC algorithm. As

we see in Figure E.4, RLS gives worse results in all dimensions. In fact, the local search alone is able on only few functions (f_1, f_4, f_5, and f_{10} of the SOCO benchmark set) to find solutions better then the optimum threshold, while for the ABC algorithms this is the case for often more than ten functions. Except for a few ABC variants, the observed differences are statistically significant. Therefore, we may conclude that there is an overall synergetic effect between the ABC algorithms and the local search.

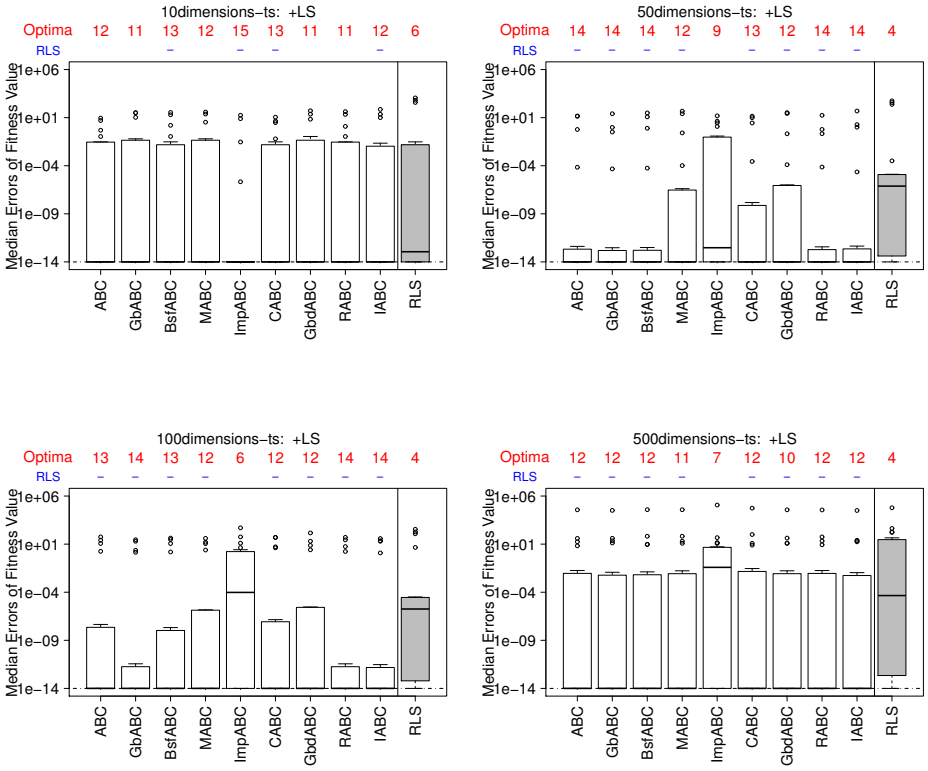


Figure E.4: Performance comparisons with RLS. A "−" symbol on top of a boxplot denotes a significant difference detected by a Wilcoxon's test at the 0.05 level (using Holm's correction) between the results obtained with the indicated algorithm and RLS.

Original ABC algorithm

As shown in the previous section, the original ABC algorithm can profit quite strongly from parameter tuning and reach performance that is similar to various of its extensions that have been proposed as improvements. This result also illustrates the danger of comparing to a default version of ABC and we recommend that

in future papers on potential improvements over ABC, the original version and the modified version is automatically tuned to avoid biases by the designer of a particular extension.

Already in earlier papers, the impact of parameter settings on the performance of ABC was studied [Akay and Karaboga, 2009, Diwold et al., 2011a, Karaboga and Basturk, 2008]. Conclusions were, for example, that a too low value of the limit parameter leads to poor performance, that ABC's performance is relatively robust given the limit value as determined through parameter lf is large enough [Akay and Karaboga, 2009, Diwold et al., 2011a], and that the population size does not need to be fine-tuned in order to obtain satisfactorily good results [Akay and Karaboga, 2009]. Our results indicate, however, that even the original ABC algorithm can profit substantially from a further fine-tuning of its parameters.

Global-best and global-best distance ABC

The GbABC and GbdABC algorithms did not profit much or at all from the additional parameter tuning. This can be seen in the bottom part of Tables E.4 to E.7 at the summary statistics on the number of functions that obtain better, same or worse median quality after tuning or the additional introduction of local search. This indicates that either their design makes them rather robust w.r.t. modified parameter settings or that the original authors have already very well fine-tuned the parameter settings. In fact, the tuned parameter settings at least for GbABC are rather similar to the ones proposed as default settings, giving some evidence for the latter. Interestingly, for GbABC and GbdABC the additional local search phase does actually slightly (though not significantly) worsen performance. This indicates that ABC algorithms that use appropriate algorithm features and that are well parameterized do not necessarily require an additional local search phase to reach very high performance on the SOCO benchmark set.

Best-so-far ABC

The poor performance of BsfABC we observed is in apparent contradiction to the excellent results reported in the original paper by Banharnsakun et al. [2011]. However, these differences can be explained by particularities in the design of BsfABC. As mentioned in Section E.2.2, BsfABC applies a position update to each dimension in such a way that the variable values in all dimensions get closer to each other. This induces a strong bias towards solving well problems where the optimum has in all dimensions the same variable values. In particular, in

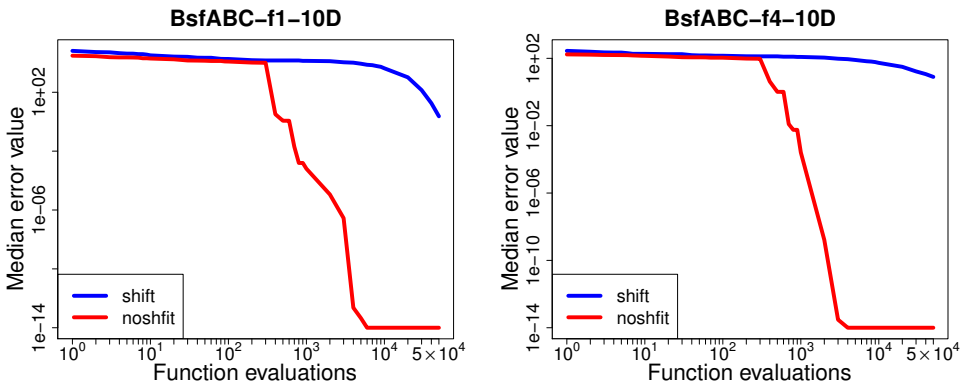


Figure E.5: Development of the median solution value over time for BsfABC on the ten dimensional SOCO functions f_1 (Sphere, left plot) and f_4 (Rastrigin, right plot) for the shifted and the unshifted versions. The unshifted version has the optimum at the point $(0, \ldots, 0)$.

the experiments reported in Banharnsakun et al. [2011] the benchmark problems have the optimum in the solution $x = (0, \ldots, 0)$, while the benchmark problems in the SOCO benchmark set have their optimum randomly shifted within the search range.

In Figure E.5 we show the development of the median error value over the number of function evaluations for the shifted and the not-shifted version of two SOCO benchmark functions (Sphere–f_1–and Rastrigin–f_4, respectively). As expected, the fact whether the optimum of a function is randomly shifted in the search range (the shift is independently done for each dimension) or not, has a huge impact in the performance of BsfABC, which is only effective for problems where the optimum solutions have a same variable value in each dimension. The same experiments with the other ABC variants did not show a significant influence of an optimum shift on performance. Finally, after tuning the performance of BsfABC is strongly improved; this is probably due to the larger setting of lf and the smaller population size.

Improved and Modified ABC

For functions of ten dimensions, MABC and, in particular, ImpABC show good results; in fact, on ten dimensions, the tuned version of ImpABC and ImpABC with local search are the best ranking ABC algorithms. However, the good performance of ImpABC and also MABC decline quickly with increasing dimensionality of the benchmark problems. For example, the tuned version of ImpABC is the worst

ranking ABC algorithm for 50 and 100 dimensional problems and the only ABC algorithms whose median is not below the optimum threshold of any function for 100 dimensional problems. Similary, the performance of ImpABC and MABC with local search declines with high dimensional functions, though less due to the mitigating effect of the local search. In other words, the scaling behavior of ImpABC and MABC is poor. As the main reason for this poor scaling behavior we identified the choice of increasing the number of variables that are modified with the dimension of the functions. In fact, if only a small, constant number of variables is modified, the scaling behavior of MABC and ImpABC improves strongly to an extent to which ImpABC and MABC become competitive with the other ABC algorithms (see also the supplementary pages `http://iridia.ulb.ac.be/supp/IridiaSupp2013-002`).

The improvements incurred by keeping the number of variables modified to a constant value is shown for MABC in Figure E.6 and for ImpABC in Figure E.7. In each of these plots we compare the median error values of MABC and ImpABC using default parameter settings except that the number of variables to be changed is set to eight for MABC and four for ImpABC, as suggested by the tuning on the ten dimensional problems. We refer to the latter variants as MABC-new and ImpABC-new.

Another explanation for the poor scaling behavior resides in the fact that the training set comprised only functions of a fixed dimensionality. Hence, the tuning cannot detect whether it is better to set the parameter to a constant or to scale it with the dimensionality of the problems. Thus, more advanced possibilities for scaling parameters would have to be considered.

Chaotic ABC

After tuning, the performance of CABC is improved compared to the default parameter settings. However, the original ABC benefits even more from tuning so that once tuned, CABC does not show anymore a significant improvement over the original ABC version, drawing some doubt about the real impact of the proposed modifications.

Rosenbrock ABC

RABC is the only ABC algorithm we included in the comparison that in its original form makes use of a local search procedure. While on dimension 10 it obtains on several functions better median performance than the tuned original ABC algo-

rithm (better on nine and worse on two), for larger dimensions it is roughly on par with the original ABC, despite making use of the Rosenbrock rotational direction method (RM). One reason may be that RM is not a very effective local search method for the benchmark problems tested here. In fact, after adding the Mtsls1 local search, the resulting hybrid RABC improves strongly its performance, giving some indication that this conjecture is true.

We can also test the effect RM has in RABC by removing it. Interestingly, once RM is removed from RABC, the only difference from the resulting ABC algorithm to the original ABC algorithm is the usage of the rank-based probabilistic selection of a solution by the onlooker bees instead of the usual fitness-based one through Equation E.4. The impact of this choice on ABC performance can be observed from the plots in Figure E.8, where the parameter settings correspond to the tuned settings of the original ABC algorithm. In fact, replacing the fitness-based selection by the rank-based probabilistic selection of solutions leads for all dimensions to improved performance (being statistically significant in dimensions 10 and 100).

Incremental ABC

IABC performs generally better than the original ABC algorithm, reaching on 11 of the 19 functions of dimension 10 better median results. Similarly, on the higher dimensional functions it improves on either 10 or 11 functions over the original ABC and is only worse on one function (f_2). The strong improvement through the tuning is probably due to the lower limit on the maximum population size in IABC and the slower increase of the population size when compared to the default parameter settings. Overall, it belongs to the best ranking ABC variants.

Solution behavior of the ABC variants

Next, we examine the development of the solution quality over time by so called SQT plots [Hoos and Stützle, 2005]. In particular, we give the development of the median error over computation time measured by the number of function evaluation as usual in continuous optimization. First we examine the impact of parameter tuning and the usage of an additional local search on the behavior of the ABC algorithms. In Figure E.9 are given representative SQT plots for the original ABC algorithm, GbABC, and IABC on three functions of dimension 50 (functions f_1, f_3, and f_{13} from the SOCO benchmark set, that is, Sphere, Rosenbrock, and a hybrid function). In these plots we can observe that the tuning improves strongly the convergence behavior of the original ABC algorithm and

IABC, while GbABC is almost not affected by the tuning, which can be noticed by the, in part, overlapping curves for the default and the tuned version. Additional local search may further speed up convergence towards high quality solutions as noticed by the fact that these curves are typically left-most in the plots. However, the additional local search does not always improve the final solution quality as can be noticed, for example, on the plots for function f_{13}.

In addition to SQT plots, we have generated run-time distributions (RTDs) for reaching the optimum threshold (or other very high quality solutions) and solution-quality distributions (SQDs) [Hoos and Stützle, 2005]. In Figure E.10 we give few RTDs for the same three algorithms with tuned parameter settings and hybridized with local search on benchmark functions f_5 and f_6. The RTDs on benchmark function f_5 indicate that the ABC variants suffer in part from severe stagnation behavior. The most extreme is IABC with local search: after less than 2000 function evaluations it finds in almost 50% of the runs a solution better than the optimum threshold but in the remaining, much longer runs it hardly finds additional optima. The RTDs on benchmark function f_5 are also interesting because they show how the ranking of the variants changes depending on whether local search is used or not: on f_5 actually the original ABC with local search performs better than the other two algorithms with local search, while without local search IABC and GbABC are clearly better than original ABC. The RTDs on f_6 are more representative for RTDs on the majority of the functions: with local search the various variants have rather similar performance, while without local search differences are still more apparent. Typcial is also that for many functions the ABC variants do not really show stagnation behavior (see also supplementary pages `http://iridia.ulb.ac.be/supp/IridiaSupp2013-002`).

Finally, Figure E.11 gives two SQDs for benchmark function f_5, which correspond to the distribution of the solution quality at the maximum number of function evaluations. These SQDs confirm the stagnation behavior showing that some runs are trapped in strongly sub-optimal solutions, an effect that is most visible in the SQDs for ABC variants with local search. In fact, we conjecture that by either changing the scheme of local search application to more frequently choose other than the global-best solution or some partial algorithm restart feature could be useful to enhance algorithm performance in such cases.

E.4.3 Comparison with SOCO special issue contributors

As a final step, we compare one of the best performing ABC variants with local search to all algorithms that have been contributed to the Soft Computing special issue Lozano et al. [2011]. All along our experiments in this chapter, we used actually the same experimental setup as proposed for this special issue and, hence, such a comparison is fair in this respect. The results of 13 algorithms have been published in the special issue. In addition, three benchmark algorithms had been included as reference algorithms to which each entry to the special issue had to be compared and which each competitor should have outperformed. These three reference algorithms were a differential evolution algorithm [Stern and Price, 1997], the real-coded CHC algorithm [Eshelman and Schaffer, 1993], and G-CMA-ES [Auger and Hansen, 2005]. In particular, G-CMA-ES was the best performing algorithm in the special session on real parameter optimization of the 2005 IEEE Congress on Evolutionary Computation (CEC'05) and the MA-SSW algorithm Molina et al. [2011], one of the 13 algorithms published in the SOCO special issue, was the best performing algorithm at the CEC 2010 special session on large scale global optimization. Hence, the algorithms to which we compare can be considered representatives of the state-of-the-art in real parameter optimization.

For this comparison we have chosen GbABC with local search as it is one of the best ABC algorithms across various dimensions and it is still a rather straightforward variant. We have computed for all the algorithms the distribution of the median error values found from the publicly available results tables and the boxplots in Figure E.12 compare these distributions across several dimensions. In fact, GbABC reaches a median performance that is competitive to the best continuous optimizers from the competition; for example, it reaches for 50 and 100 dimensions a median error that is below the optimum threshold for 14 functions, the same as the overall best performing algorithm in the benchmark competition, MOS-DE [LaTorre et al., 2011]. A further statistical analysis (using Wilcoxon's test with Holm's correction for multiple comparisons) shows that GbABC is statistically significantly better than reference algorithms DE, CHC and G-CMA-ES as well as some contributors such as SOUPDE, GODE, MA-SSW, RPSO-vm, EvoPROpt and EM323 for dimensions 100 and 500. Hence, when considering median performance, GbABC (and actually also several other ABC algorithms, such as IABC, which have very similar performance to GbABC) are competitive with state-of-the-art algorithms for large-scale continuous optimization problems.

The good median performance is in part due to the fact that the tuning is

done using F-race, a rank-based method. If the algorithms are compared based on mean error values, GbABC is still among the best performing ones but compares worse, for example, to MOS-DE than when using median error values. (A comparison of GbABC to the other algorithms from the SOCO special issue based on mean error values is available on the chapter's supplementary pages `http://iridia.ulb.ac.be/supp/IridiaSupp2013-002`.) The main reason for the worse mean performance is due to the stagnation of the algorithm in some runs on few functions. It would be interesting to re-tune the algorithms based on, for example, t-race which implicitly will favor algorithms having better average behavior.

E.5 Discussion and conclusions

In this chapter, we have reviewed artificial bee colony (ABC) optimization algorithms for continuous optimization and we have examined various of the proposed ABC experimentally using a recent benchmark function set for large-scale continuous optimization. Our experimental analysis is based on re-implementations of nine ABC algorithms including the original one, which are compared under same computational conditions. The ABC variants were compared using three setups. First, using their default parameter settings; second, using an automatic algorithm configuration tool to determine parameter settings; third, by combining the ABC algorithms with local search and re-tuning their parameter settings by an automatic algorithm configuration tool. The usage of an automatic algorithm configuration tool in our context has essentially three main reasons. The first is to potentially improve significantly the default algorithm parameter settings; this was actually achieved for most of the ABC variants. The second reason is to avoid the bias in results due to an uneven tuning of the different algorithm variants; this was achieved by using a same tuning setup and effort for all algorithms. A third reason is to help in the design of hybrid ABC algorithms that include an effective local search for continuous optimization; this was necessary due to the fact that nine hybrid algorithms had to be designed. The high performance of the final hybrid algorithms justifies *a posteriori* this procedure.

Some of the main results of our experimental analysis and comparison are the following. First, when considering default parameter settings and low dimensional benchmark problems, most of the ABC variants actually improve over the original ABC algorithm, as would be promised by the original papers where the variants

have been proposed.[1]

However, this conclusion changes as we move from to tuned parameter settings and to the ABC variants with local search or as we move to higher-dimensional problems. Considering tuned parameter settings, several of the proposed ABC variants do not result anymore in significantly improved performance (especially in higher dimensional problems). The situation gets even "worse" if we move to the ABC variants that include an effective local search algorithm. In fact, no statistically significant differences of the hybrid ABC variants to the original ABC with local search are detected anymore when measuring performance by the median error value. This result is due to the fact that an effective local search typically smoothes the performance differences between algorithms without local search. In fact, we could claim that as a side-product of our experimental analysis, we have derived a number of new state-of-the-art ABC algorithms, as most of the high-performing, hybrid ABC algorithms we examine have never been considered before. However, the particular scheme we used of adding local search to ABC algorithms has been examined before in other contexts [Aydın et al., 2012, Liao et al., 2011b, Montes de Oca et al., 2011] and the resulting high performance actually is rather an indication that this particular hybridization scheme is promising. A maybe surprising result is that few of the ABC variants that we examined do not profit really from the additional local search phase but reach very high performance without. This is the case for GbABC and GbdABC and it suggests that ABC algorithms, if appropriately configured and tuned have already a very strong local search behavior.

If we move from the low-dimensional problems to higher-dimensional ones, we observed that some of the proposed ABC variants have rather poor scaling behavior and become the worst performing ABC algorithms for the highest dimensional problems we considered. This poor scaling behavior affected mainly modified ABC [Akay and Karaboga, 2012] and improved ABC [Gao and Liu, 2011]. However, we could identify that the underlying reason was a particular choice in the setting of a specific parameter that scaled with the dimensionality (see Section E.4.2); by keeping the parameter simply to a constant, the scaling behavior of these two variants was largely improved making them competitive to the other variants for high dimensional functions.

In a final step, we compared the median error values obtained by one of the

[1] An exception is the best-so-far ABC algorithm [Banharnsakun et al., 2011] for which we have observed poor performance. In Section E.4.2 we have shown that this poor behavior is due to specific choices in the algorithm design.

best performing ABC algorithms, the hybrid GbABC, to the results of a recent benchmarking effort for high dimensional real function optimization. A result of this comparison is that the best performing ABC variants we examined are well competitive with state-of-the-art algorithms for continuous function optimization.

Finally, there is a large number of further work that could be done in the direction followed here. A straightforward one is to extend the analysis of the ABC algorithms on other benchmark sets, as, depending on the properties of the benchmark functions, the relative performance of algorithms may change. In fact, it would be also interesting to extend the study to other benchmark sets such as those from the CEC 2005 special session or the Black-Box Optimization Benchmarking (BBOB) suite. Another direction would be to re-consider the integration of local search algorithms into ABC algorithms and try to elaborate a more refined and potentially more performing scheme for integration. The fact that our hybrid algorithms have shown on some functions stagnation behavior is an indication that by a more refined design further improvements are possible. Given the existance of a number of other ABC variants we did not include in our experimental comparison, yet another direction would be to extend our experimental study to these. Nevertheless, we think that more interesting would be the elaboration of a flexible algorithm framework for ABC algorithms, where the various proposed modifications to ABC are implemented as algorithm components that may be combined. Automatic algorithm configuration could then be applied to such a framework to possibly obtain even more performing algorithms. Such an approach has been explored for other problems [KhudaBukhsh et al., 2009, López-Ibáñez and Stützle, 2012] and the high performance of our initial effort on configuring hybrid ABC algorithms with local search just confirm that such an approach is very promising.

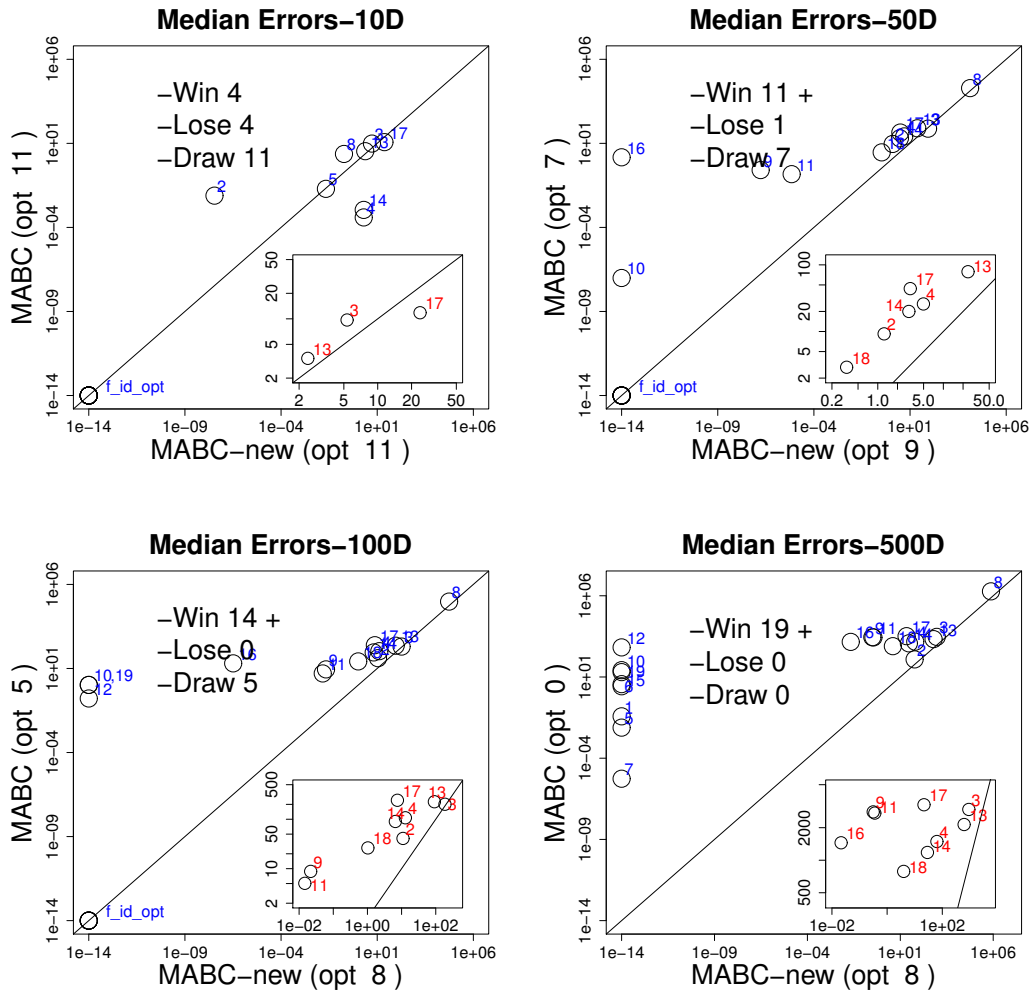


Figure E.6: Correlation plots of MABC and MABC-new on dimensions 10, 50, 100 and 500, respectively. Each point represents the median error value obtained by either of the two algorithms. A point on the upper triangle delimited by the diagonal indicates better performance for the algorithm on the x-axis; a point on the lower right triangle indicates better performance for the algorithm on the y-axis. The number labeled beside some outstanding points represent the index of the corresponding function. The comparison is conducted based on median error values and the comparison results of the algorithm on the x-axis are presented in form of -win, -draw, -lose, respectively. We marked with a + symbol those cases in which there is a statistically significant difference at the 0.05 α-level between the algorithms. The number of opt on the axes shows the number of medians lower than the zero threshold by the corresponding algorithm. (with default parameter settings)

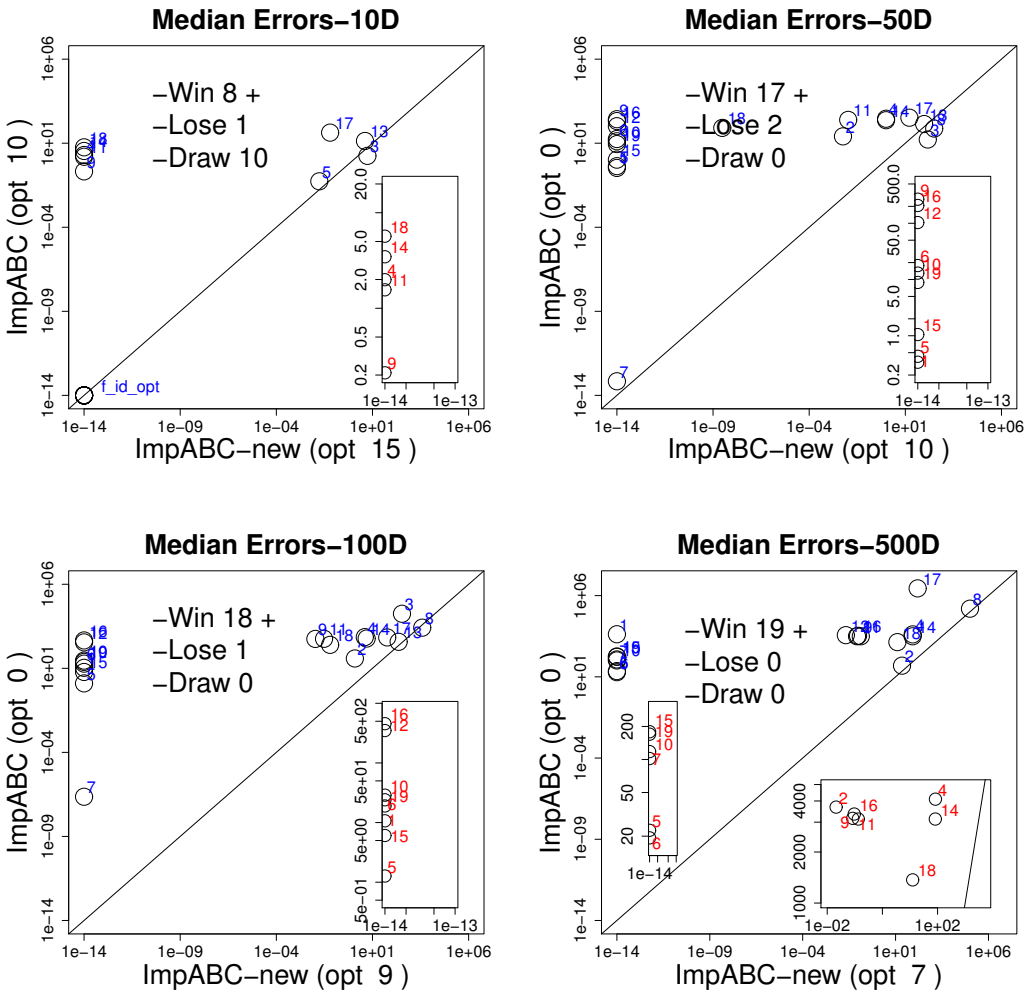


Figure E.7: Correlation plots of ImpABC and ImpABC-new on dimensions 10, 50, 100 and 500 respectively. For an explanation of the plots see caption of Figure E.6.

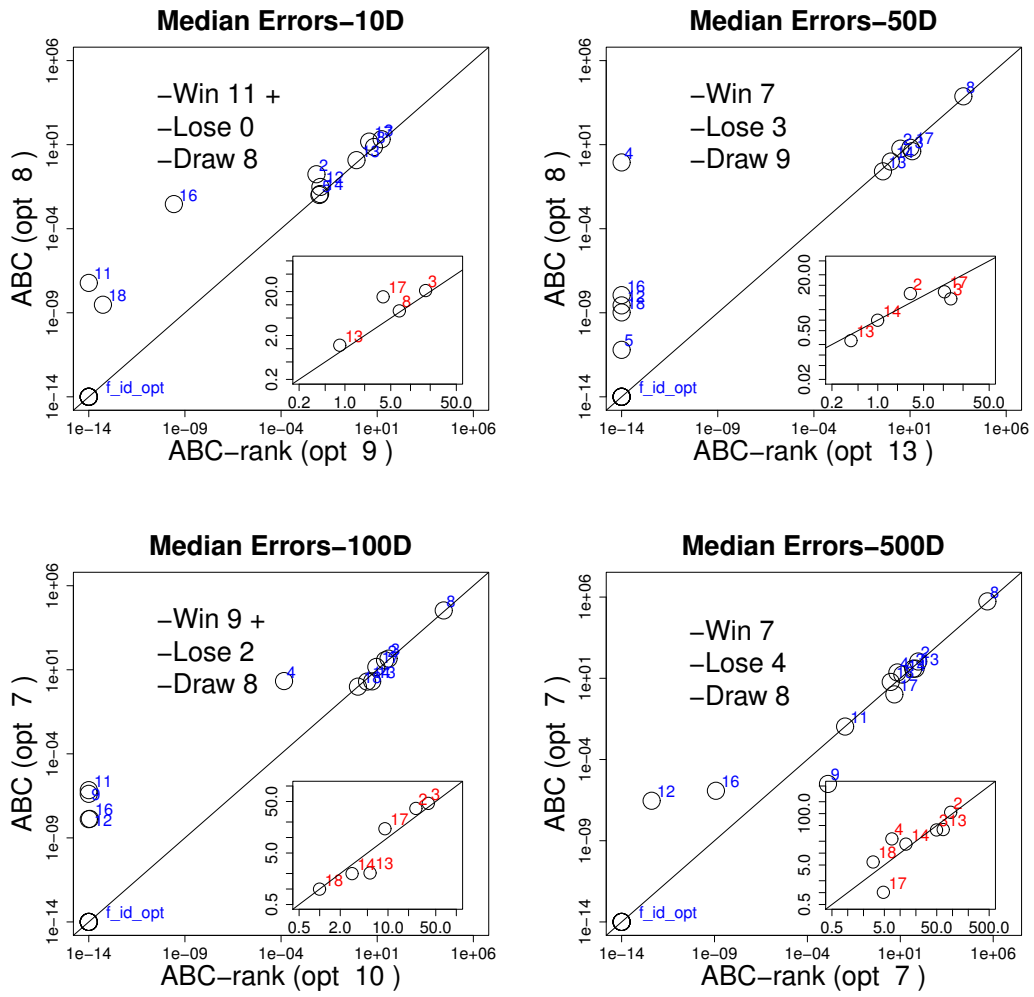


Figure E.8: Correlation plots of ABC and ABC with a rank-based selection of solutions by onlooker bees (ABC-rank) on dimensions 10, 50, 100 and 500, respectively. The parameter settings used are the tuned settings for ABC. For an explanation of the plots see caption of Figure E.6.

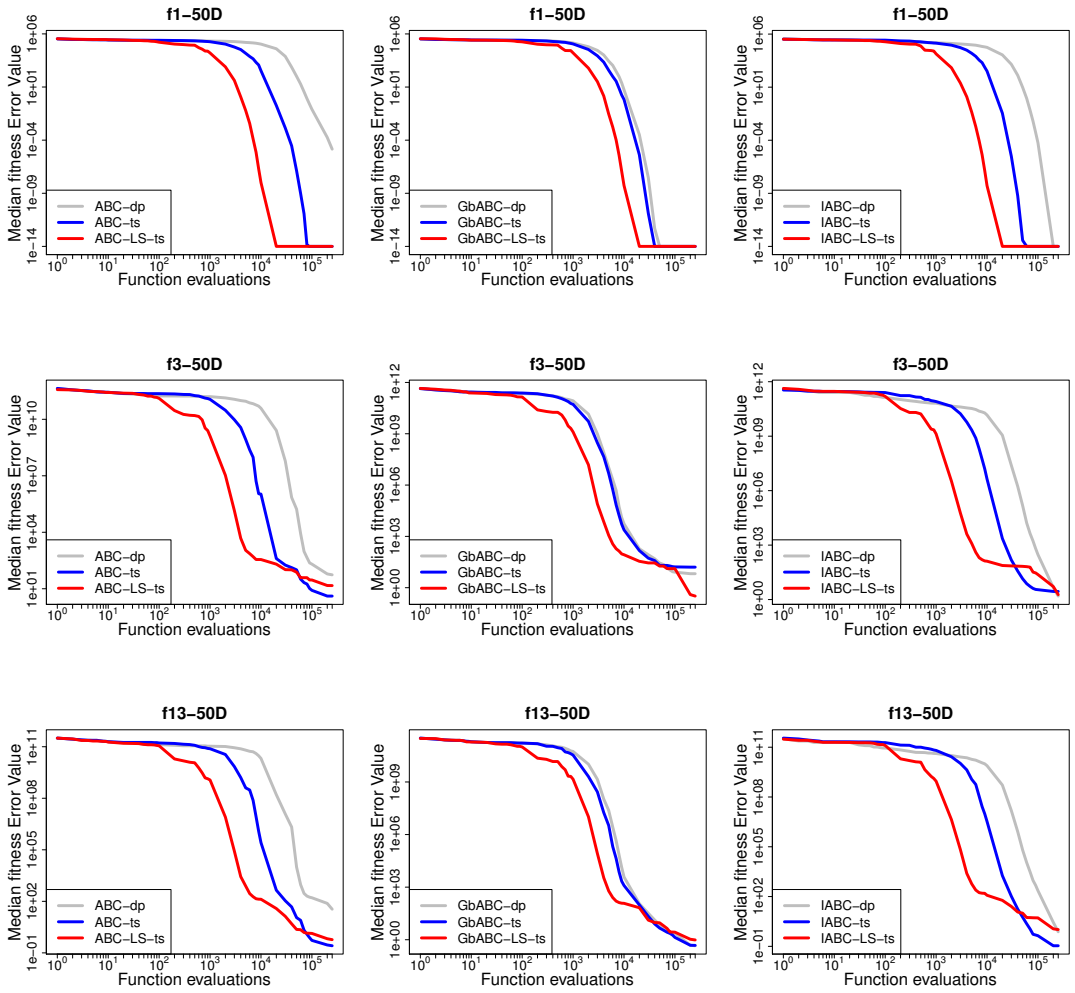


Figure E.9: SQD curves for the original ABC (left column), GbABC (middle column), and IABC (right column) for functions f_1 (Sphere, top), f_3 (Rosenbrock, middle) and f_{13} (a hybrid function, bottom).

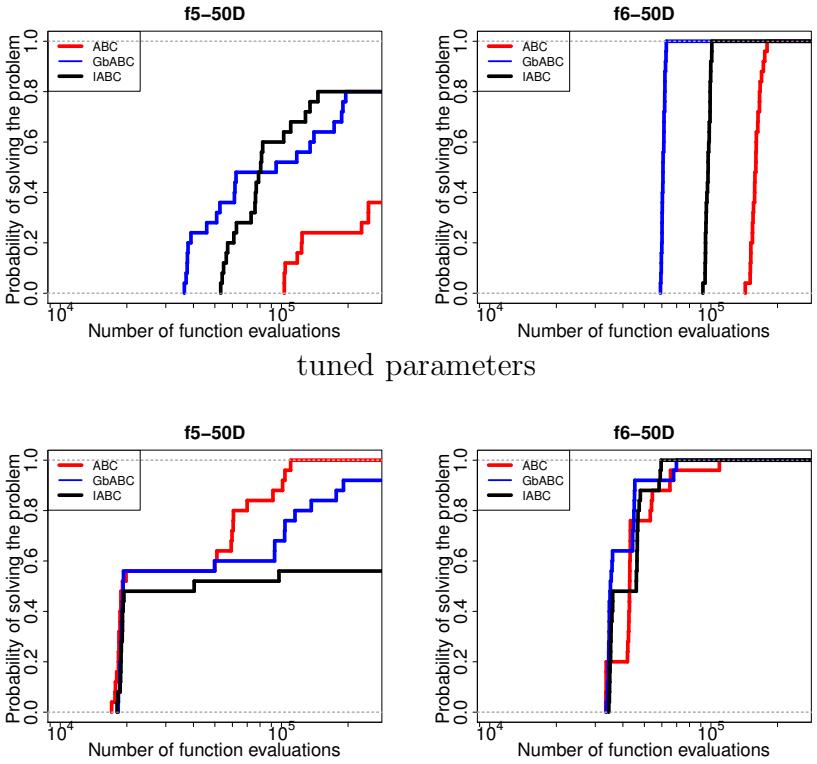


Figure E.10: RLDs for the original ABC, GbABC and IABC for functions f_5 (left column) and f_6 (right column) using tuned parameter settings (upper row) and in combination with local search (bottom row).

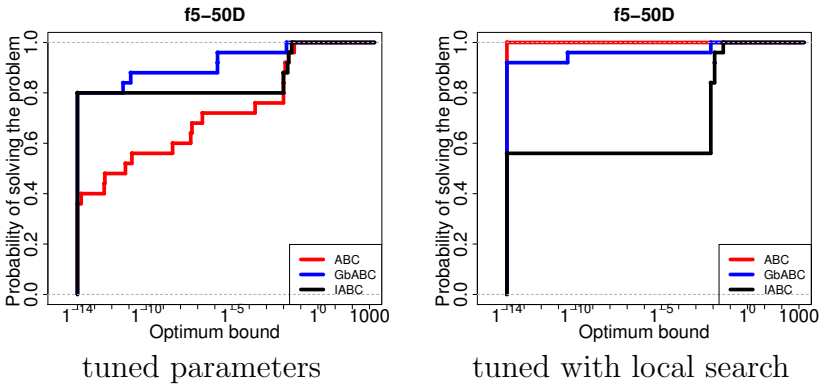


Figure E.11: SQDs for the original ABC, GbABC and IABC for function f_5 using tuned parameter settings (left) and in combination with local search (right).

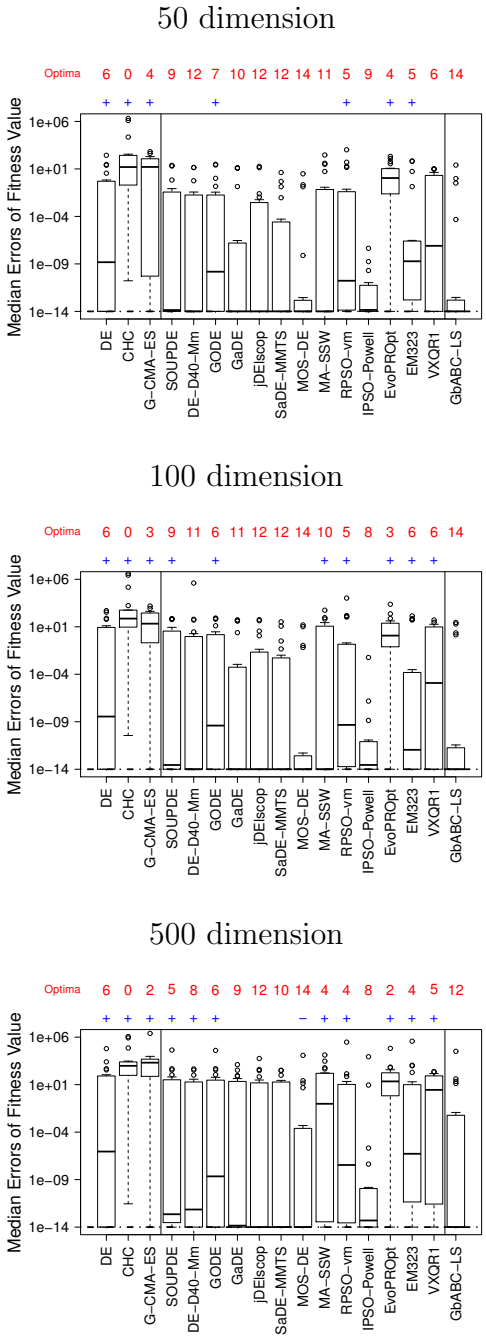


Figure E.12: Comparison of tuned GbABC with local search to the algorithms from the SOCO benchmark competition. A "+" symbol on top of a box-plot denotes a significant difference detected by a Wilcoxon's test at the 0.05 level between the results obtained GbABC and the indicated 16 reference algorithm. If an algorithm is significantly better than the indicated algorithm, a "−" symbol is put on top of a box-plot for indicating this difference.

Table E.4: Given are the medians of the various ABC variants on the 10 dimensional functions from the SOCO benchmark set for the default parameter settings (upper part), the tuned parameter settings (middle part) and the tuned hybrid ABC algorithms with local search (bottom part). At the bottom is indicated the number of times the tuned parameter settings give lower, same or worse median (win, draw, loss) than default parameter settings (ts vs dp) and the same information for the comparisons between the hybrids with local search versus default parameter settings (LS vs dp) and hybrids with local search versus tuned settings (LS vs ts).

default parameter settings

Function	ABC	GbABC	BsfABC	MABC	ImpABC	CABC	GbdABC	RABC	IABC
f_1	8.31E−07	1.00E−14	2.33E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	1.58E+00	1.02E−08	1.93E+01	7.75E−03	1.00E−14	7.18E+00	1.59E−09	9.50E−04	6.67E−01
f_3	7.16E+00	2.80E+00	1.23E+04	9.69E+00	1.77E+00	1.37E+01	2.07E+00	2.39E+00	2.71E+00
f_4	7.06E−03	1.00E−14	7.88E+00	3.92E−04	1.99E+00	1.27E−09	1.00E−14	1.00E−14	2.03E−14
f_5	3.29E−03	2.67E−08	8.38E−01	1.97E−02	5.41E−02	8.62E−03	6.71E−12	7.40E−03	2.34E−04
f_6	7.53E−03	1.00E−14	3.72E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.23E−13	7.85E−12
f_7	5.14E−05	1.00E−14	1.65E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.12E−13
f_8	4.55E+01	6.08E+00	2.99E+02	2.34E+00	1.00E−14	7.86E+01	6.00E−01	1.04E−02	2.61E+01
f_9	2.37E+00	1.00E−14	1.64E+01	1.00E−14	2.11E−01	5.44E−01	1.00E−14	2.92E−03	2.77E−02
f_{10}	1.51E−07	1.00E−14	1.14E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	1.97E+00	1.00E−14	1.50E+01	1.00E−14	1.56E+00	7.20E−01	1.00E−14	3.66E−03	3.37E−02
f_{12}	2.57E−01	1.76E−03	6.14E+00	1.00E−14	1.00E−14	3.87E−02	1.00E−14	2.25E−04	1.07E−02
f_{13}	3.35E+00	4.32E−01	3.10E+03	3.42E+00	1.36E+01	4.62E+00	1.59E−01	1.29E−01	4.07E−01
f_{14}	1.69E−01	1.75E−03	7.12E+00	1.10E−03	3.46E+00	3.03E−02	3.90E−14	2.05E−04	6.45E−03
f_{15}	3.15E−05	1.00E−14	1.90E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	6.60E−14
f_{16}	7.75E−01	1.00E−14	7.67E+00	1.00E−14	1.00E−14	3.05E−02	1.00E−14	4.53E−04	5.95E−03
f_{17}	1.80E+00	6.00E−01	3.39E+01	1.18E+01	4.26E+01	8.42E+00	2.44E−01	5.66E−01	4.83E−01
f_{18}	4.30E−01	1.00E−14	3.62E+00	1.00E−14	5.68E+00	1.06E−02	1.00E−14	1.36E−04	2.11E−03
f_{19}	2.03E−06	1.00E−14	4.94E−01	1.00E−14	1.00E−14	2.18E−14	1.00E−14	1.00E−14	1.00E−14

tuned parameter settings

Function	ABC-ts	GbABC-ts	BsfABC-ts	MABC-ts	ImpABC-ts	CABC-ts	GbdABC-ts	RABC-ts	IABC-ts
f_1	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	1.74E−01	4.21E−12	1.36E−01	9.50E−07	1.00E−14	2.52E−02	2.28E−13	4.26E−14	4.48E−05
f_3	2.11E+01	2.54E+00	4.09E+01	5.00E+00	5.06E+00	4.68E+00	1.41E+00	1.82E+01	1.11E+01
f_4	1.00E−14	1.00E−14	9.95E−01	2.00E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_5	1.01E−02	6.86E−09	1.48E−02	3.94E−02	7.40E−03	9.86E−03	1.00E−14	2.22E−02	9.86E−03
f_6	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_8	7.27E+00	4.62E+00	6.63E+00	9.62E−08	1.00E−14	1.48E+01	1.71E+00	1.00E−14	4.60E+00
f_9	1.00E−14	1.00E−14	5.59E−06	1.00E−14	1.00E−14	4.74E−05	1.00E−14	1.00E−14	1.00E−14
f_{10}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	5.99E−08	1.00E−14	2.51E−04	1.00E−14	1.00E−14	2.20E−04	1.00E−14	1.00E−14	1.00E−14
f_{12}	3.05E−02	1.32E−03	3.09E−02	1.00E−14	1.00E−14	1.15E−02	1.00E−14	1.07E−02	1.08E−07
f_{13}	1.20E+00	3.31E−01	6.13E−01	3.44E+00	5.03E+00	7.43E−01	2.59E−01	3.99E+00	2.20E−01
f_{14}	1.16E−02	1.61E−03	1.03E+00	1.99E+00	1.00E−14	9.62E−03	1.00E−14	5.30E−05	6.00E−08
f_{15}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{16}	2.92E−03	1.00E−14	1.19E−02	1.00E−14	1.00E−14	7.82E−04	1.00E−14	1.00E−14	1.00E−14
f_{17}	1.52E+01	2.73E−01	8.09E−01	2.57E+01	1.91E−11	7.71E−01	2.90E−01	6.86E+00	4.33E+00
f_{18}	3.01E−09	1.00E−14	3.23E−05	1.49E−12	1.00E−14	5.36E−06	1.00E−14	1.00E−14	1.00E−14
f_{19}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14

hybrids with local search

Function	ABC-LS	GbABC-LS	BsfABC-LS	MABC-LS	ImpABC-LS	CABC-LS	GbdABC-LS	RABC-LS	IABC-LS
f_1	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	1.42E−14	1.42E−14	1.42E−14	1.42E−14	1.00E−14	1.42E−14	1.42E−14	1.42E−14	1.24E−14
f_3	5.00E+00	3.04E+01	3.47E+01	3.82E+01	2.90E+00	1.15E+01	5.16E+01	4.23E+01	7.17E+01
f_4	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_5	1.00E−14	7.40E−03	1.00E−14	1.00E−14	2.71E−02	1.00E−14	1.23E−02	7.40E−03	2.21E−02
f_6	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_8	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_9	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{10}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{12}	3.03E−02	6.27E−02	3.03E−02	6.27E−02	1.00E−14	3.03E−02	6.27E−02	3.03E−02	1.07E−02
f_{13}	4.91E−01	1.06E+01	1.59E+00	3.35E+00	4.21E+00	4.90E−01	6.86E+00	1.23E+00	1.01E+01
f_{14}	1.10E−01	6.27E−02	1.10E−01	6.27E−02	1.00E−14	1.10E−01	1.10E−01	1.10E−01	1.07E−02
f_{15}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{16}	2.68E−02	2.68E−02	1.00E−14	2.68E−02	1.00E−14	2.68E−02	2.68E−02	2.68E−02	1.00E−14
f_{17}	8.95E+00	3.35E+01	1.97E+01	2.35E+01	2.84E−06	3.42E−01	2.31E+01	2.18E+01	2.25E+01
f_{18}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.21E−01	1.00E−14	1.00E−14	1.00E−14
f_{19}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14

(Win, Draw, Loss)

ts vs dp	(16, 0, 3)	(8, 11, 0)	(19, 0, 0)	(3, 10, 6)	(8, 10, 1)	(13, 5, 1)	(4, 12, 3)	(8, 6, 5)	(13, 3, 3)
LS vs dp	(18, 0, 1)	(2, 10, 7)	(19, 0, 0)	(5, 9, 5)	(8, 10, 1)	(12, 5, 2)	(2, 10, 7)	(6, 7, 6)	(10, 4, 5)
LS vs ts	(9, 8, 2)	(2, 10, 7)	(11, 6, 2)	(8, 8, 3)	(2, 15, 2)	(7, 7, 5)	(2, 10, 7)	(3, 11, 5)	(2, 11, 6)

Table E.5: Given are the medians of the various ABC variants on the 50 dimensional functions from the SOCO benchmark set for the default parameter settings (upper part), the tuned parameter settings (middle part) and the tuned hybrid ABC algorithms with local search (bottom part). At the bottom is indicated the number of times the tuned parameter settings give lower, same or worse median (win, draw, loss) than default parameter settings (ts vs dp) and the same information for the comparisons between the hybrids with local search versus default parameter settings (LS vs dp) and hybrids with local search versus tuned settings (LS vs ts).

default parameter settings

Function	ABC	GbABC	BsfABC	MABC	ImpABC	CABC	GbdABC	RABC	IABC
f_1	1.42E−05	1.00E−14	1.60E+03	1.00E−14	3.38E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	6.98E+01	1.93E+00	4.08E+01	9.22E+00	2.55E+01	6.20E+01	1.46E+00	3.30E+00	5.63E+01
f_3	5.48E+01	6.58E+00	7.66E+07	7.49E+01	1.66E+01	2.75E+00	7.24E+00	1.56E+00	1.56E+00
f_4	6.12E+00	1.00E−14	1.58E+02	2.59E+01	2.85E+02	2.32E−12	1.00E−14	4.33E−13	3.28E−02
f_5	2.83E−04	1.00E−14	1.54E+01	1.00E−14	4.31E−01	5.03E−14	1.00E−14	1.00E−14	1.00E−14
f_6	5.45E−02	1.00E−14	1.10E+01	1.00E−14	1.77E+01	1.00E−14	1.00E−14	3.01E−11	2.72E−09
f_7	1.22E−03	1.00E−14	2.53E+00	1.00E−14	6.89E−14	1.00E−14	1.00E−14	2.50E−14	2.16E−11
f_8	1.58E+04	8.40E+03	6.44E+03	1.93E+04	7.46E+01	9.15E+03	6.38E+03	1.55E+03	5.85E+03
f_9	2.37E+01	1.00E−14	1.79E+02	2.57E−01	2.68E+02	2.04E−07	1.00E−14	4.86E−02	6.53E−01
f_{10}	2.00E−05	1.00E−14	4.03E+01	9.79E−08	1.30E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	2.20E+01	1.00E−14	1.68E+02	1.47E−01	2.44E+02	4.46E−07	1.00E−14	6.00E−02	6.37E−01
f_{12}	2.05E+00	1.00E−14	8.64E+02	1.00E−14	1.03E+02	7.98E−07	1.00E−14	4.10E−03	4.87E−02
f_{13}	3.48E+01	4.53E−01	3.98E+07	7.89E+01	1.38E+02	2.98E−01	4.05E−01	2.43E−01	7.53E−01
f_{14}	6.77E+00	1.00E−14	1.23E+02	2.00E+01	2.33E+02	1.25E−11	1.00E−14	3.83E−04	2.79E−02
f_{15}	7.47E−04	1.00E−14	4.60E+00	1.00E−14	1.05E+00	1.00E−14	1.00E−14	1.76E−14	1.30E−11
f_{16}	5.15E+00	1.00E−14	2.87E+02	1.48E+00	2.07E+02	3.46E−11	1.00E−14	1.55E−02	1.78E−01
f_{17}	2.86E+01	5.22E+00	1.40E+05	4.40E+01	3.29E+02	1.66E+01	1.05E+00	2.68E+00	4.80E+00
f_{18}	4.79E+00	1.00E−14	4.48E+01	2.91E+00	8.15E+01	2.66E−12	1.00E−14	3.73E−03	8.33E−02
f_{19}	1.55E−04	1.00E−14	1.63E+01	1.00E−14	8.87E+00	1.00E−14	1.00E−14	1.00E−14	4.12E−13

tuned parameter settings

Function	ABC-ts	GbABC-ts	BsfABC-ts	MABC-ts	ImpABC-ts	CABC-ts	GbdABC-ts	RABC-ts	IABC-ts
f_1	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	5.79E+00	3.23E−01	3.45E+01	2.13E+01	8.24E+00	9.08E+00	1.67E−01	1.20E−03	2.43E+01
f_3	4.08E+00	1.59E+01	5.70E+01	8.44E+01	7.82E+01	1.43E+00	1.87E+01	3.53E+01	2.80E+00
f_4	8.78E−01	1.00E−14	4.58E+01	6.07E+01	1.09E+02	1.00E−14	1.00E−14	9.95E−01	1.00E−14
f_5	6.23E−12	1.00E−14	2.70E−02	1.00E−14	6.86E−02	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_6	1.00E−14	1.00E−14	1.00E−14	1.00E−14	2.48E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_8	7.51E+03	8.00E+03	1.26E+03	1.10E+04	1.38E+01	9.79E+03	6.30E+03	2.22E+00	4.72E+03
f_9	1.00E−14	1.00E−14	3.03E−02	2.15E+00	4.90E+01	1.85E−04	1.00E−14	1.00E−14	1.00E−14
f_{10}	1.00E−14	1.00E−14	1.05E+00	1.39E−07	6.30E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	1.00E−14	1.00E−14	3.07E−02	2.69E+00	4.13E+01	3.92E−04	1.00E−14	1.00E−14	1.00E−14
f_{12}	2.75E−09	1.00E−14	6.23E−04	1.07E−02	5.12E+01	2.60E−05	1.00E−14	4.24E−14	1.00E−14
f_{13}	2.59E−01	4.75E−01	5.60E+00	7.72E+01	1.25E+02	8.53E−02	3.36E+00	4.01E+00	1.08E−01
f_{14}	9.95E−01	1.00E−14	3.48E+01	4.64E+01	7.93E+01	6.51E−07	5.45E−14	3.98E+00	1.00E−14
f_{15}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	4.13E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{16}	1.12E−08	1.00E−14	2.92E−04	3.82E−01	9.70E+01	1.03E−04	1.00E−14	7.23E−11	1.00E−14
f_{17}	6.50E+00	5.22E+00	4.10E+01	7.32E+01	2.27E+02	8.11E−01	5.40E+00	3.75E+01	3.33E+00
f_{18}	1.05E−09	1.00E−14	9.95E−01	1.66E+01	3.40E+01	5.73E−06	1.00E−14	9.95E−01	1.00E−14
f_{19}	1.00E−14	1.00E−14	1.52E+00	4.70E−01	3.15E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14

hybrids with local search

Function	ABC-LS	GbABC-LS	BsfABC-LS	MABC-LS	ImpABC-LS	CABC-LS	GbdABC-LS	RABC-LS	IABC-LS
f_1	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	4.12E−13	2.98E−13	3.13E−13	4.41E−13	1.56E−13	3.13E−13	4.69E−13	3.69E−13	4.41E−13
f_3	1.47E+01	3.24E−01	1.20E+01	2.53E+01	7.05E+00	1.19E+01	3.27E+01	5.93E−01	1.87E+00
f_4	1.00E−14	1.00E−14	1.00E−14	1.00E−14	9.95E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_5	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_6	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_8	6.92E−05	4.69E−05	5.64E−05	1.03E−04	2.40E+00	3.54E−05	1.28E−04	7.28E−05	2.27E−05
f_9	1.00E−14	1.00E−14	1.00E−14	1.79E−07	2.15E−02	1.00E−14	7.44E−07	1.00E−14	1.00E−14
f_{10}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	1.00E−14	1.00E−14	1.00E−14	4.12E−07	2.16E−02	2.98E−08	1.02E−06	1.00E−14	1.00E−14
f_{12}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.82E−01	1.00E−14	1.00E−14	1.00E−14
f_{13}	5.88E−01	9.73E−01	7.89E−01	2.59E−01	1.46E+00	1.54E+00	2.10E−01	2.12E−01	1.02E+00
f_{14}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	7.28E−12	1.82E−01	1.00E−14	1.00E−14	1.00E−14
f_{15}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{16}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	3.79E−07	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{17}	1.39E+01	2.54E+01	3.08E+01	4.71E+01	6.92E+00	2.82E+01	2.71E+01	1.75E+01	4.90E+01
f_{18}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	9.95E−01	1.19E+00	1.00E−14	1.00E−14	1.00E−14
f_{19}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14

(Win, Draw, Loss)

ts vs dp	(19, 0, 0)	(2, 15, 2)	(19, 0, 0)	(3, 5, 11)	(18, 0, 1)	(6, 6, 7)	(2, 13, 4)	(9, 4, 6)	(15, 3, 1)
LS vs dp	(19, 0, 0)	(3, 14, 2)	(19, 0, 0)	(11, 7, 1)	(19, 0, 0)	(7, 6, 6)	(3, 12, 4)	(14, 4, 1)	(13, 3, 3)
LS vs ts	(8, 8, 3)	(3, 14, 2)	(15, 4, 0)	(15, 4, 0)	(17, 2, 0)	(5, 8, 6)	(4, 11, 4)	(10, 9, 0)	(3, 14, 2)

Table E.6: Given are the medians of the various ABC variants on the 100 dimensional functions from the SOCO benchmark set for the default parameter settings (upper part), the tuned parameter settings (middle part) and the tuned hybrid ABC algorithms with local search (bottom part). At the bottom is indicated the number of times the tuned parameter settings give lower, same or worse median (win, draw, loss) than default parameter settings (ts vs dp) and the same information for the comparisons between the hybrids with local search versus default parameter settings (LS vs dp) and hybrids with local search versus tuned settings (LS vs ts).

default parameter settings

Function	ABC	GbABC	BsfABC	MABC	ImpABC	CABC	GbdABC	RABC	IABC
f_1	5.20E−05	1.00E−14	9.10E+03	1.00E−14	1.07E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	9.84E+01	1.80E+01	4.61E+01	4.09E+01	3.85E+01	9.24E+01	1.48E+01	1.59E+01	8.05E+01
f_3	1.80E+02	2.27E+01	6.62E+08	2.01E+02	1.76E+04	2.85E+01	4.29E+01	1.22E+01	1.27E+01
f_4	1.59E+01	1.00E−14	3.63E+02	1.07E+02	7.18E+02	1.70E−07	1.00E−14	2.88E−08	1.29E+00
f_5	1.05E−04	1.00E−14	6.78E+01	1.00E−14	1.27E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_6	1.66E−01	1.00E−14	1.38E+01	1.00E−14	1.90E+01	1.00E−14	1.00E−14	6.24E−11	1.00E−08
f_7	2.77E−03	1.00E−14	6.91E+00	1.00E−14	2.29E−07	1.00E−14	1.00E−14	1.30E−13	8.03E−11
f_8	5.71E+04	3.36E+04	2.47E+04	9.54E+04	2.62E+03	4.07E+04	3.22E+04	1.21E+04	2.17E+04
f_9	5.55E+01	1.00E−14	4.12E+02	8.81E+00	5.61E+02	1.20E−07	1.00E−14	1.81E−01	1.60E+00
f_{10}	7.98E−05	1.00E−14	1.35E+02	1.05E+00	2.88E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	5.65E+01	1.00E−14	4.10E+02	5.05E+00	5.69E+02	3.79E−07	1.00E−14	1.61E−01	1.77E+00
f_{12}	4.92E+00	1.00E−14	6.03E+03	1.64E−01	3.49E+02	7.58E−11	1.00E−14	1.54E−02	1.84E−01
f_{13}	8.28E+01	2.07E+00	2.80E+08	2.26E+02	3.77E+02	3.81E+00	7.34E+00	1.91E−01	1.45E+00
f_{14}	1.54E+01	1.00E−14	2.92E+02	9.01E+01	5.92E+02	7.70E−12	1.00E−14	5.02E−03	1.18E+00
f_{15}	2.07E−03	1.00E−14	1.60E+01	1.00E−14	6.07E+00	1.00E−14	1.00E−14	5.94E−14	4.94E−11
f_{16}	1.27E+01	1.00E−14	2.49E+03	2.01E+01	4.55E+02	1.94E−10	1.00E−14	5.00E−02	5.29E−01
f_{17}	6.93E+01	4.02E+00	7.73E+06	2.42E+02	6.88E+02	6.04E+00	2.95E+00	1.25E+00	8.49E+00
f_{18}	1.19E+01	1.00E−14	1.17E+02	2.63E+01	2.49E+02	6.72E−11	1.00E−14	1.49E−02	2.72E−01
f_{19}	4.98E−04	1.00E−14	7.00E+01	1.05E+00	2.41E+01	1.00E−14	1.00E−14	1.00E−14	4.18E−12

tuned parameter settings

Function	ABC-ts	GbABC-ts	BsfABC-ts	MABC-ts	ImpABC-ts	CABC-ts	GbdABC-ts	RABC-ts	IABC-ts
f_1	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.08E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	3.67E+01	6.05E+00	4.56E+01	6.32E+01	2.73E+01	3.70E+01	3.92E+00	2.87E−01	5.33E+01
f_3	4.60E+01	4.31E+01	1.92E+02	1.74E+02	1.68E+02	8.95E+00	1.04E+02	2.39E+01	2.04E+01
f_4	2.13E+00	1.00E−14	8.16E+01	1.90E+02	4.36E+02	1.00E−14	1.00E−14	8.95E+00	9.95E−01
f_5	1.00E−14	1.00E−14	5.57E−02	1.00E−14	9.31E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_6	1.00E−14	1.00E−14	1.07E−11	8.19E−08	8.96E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	1.53E−14	1.24E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_8	3.39E+04	3.20E+04	4.98E+03	6.96E+04	1.18E+03	3.90E+04	2.82E+04	2.27E+02	1.90E+04
f_9	4.18E−07	1.00E−14	1.39E+01	1.66E+02	3.43E+02	8.53E−04	1.00E−14	1.00E−14	1.00E−14
f_{10}	1.00E−14	1.00E−14	4.20E+00	2.10E+00	2.10E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	6.91E−07	1.00E−14	2.29E−01	1.54E+02	3.51E+02	9.34E−04	1.00E−14	1.00E−14	1.00E−14
f_{12}	1.30E−08	1.00E−14	3.29E−05	1.37E+00	1.90E+02	9.58E−05	1.00E−14	3.07E−11	1.00E−14
f_{13}	2.05E+00	6.70E+00	6.96E+01	2.09E+02	3.48E+02	8.28E−01	2.34E+01	2.71E+01	9.38E−01
f_{14}	1.99E+00	1.00E−14	9.45E+01	1.42E+02	3.23E+02	4.20E−06	3.14E−11	6.96E+00	9.95E−01
f_{15}	1.00E−14	1.00E−14	1.00E−14	1.06E−14	3.15E+00	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{16}	1.35E−08	1.00E−14	1.08E−02	2.13E+01	3.44E+02	2.24E−04	1.00E−14	3.94E−09	1.00E−14
f_{17}	1.48E+01	1.52E+01	2.97E+01	3.30E+02	5.93E+02	2.41E+00	1.39E+01	2.38E+01	6.30E+00
f_{18}	9.95E−01	1.00E−14	5.18E+01	9.57E+01	1.18E+02	1.11E−05	1.00E−14	2.98E+00	1.00E−14
f_{19}	1.00E−14	1.00E−14	7.35E+00	3.15E+00	1.78E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14

hybrids with local search

Function	ABC-LS	GbABC-LS	BsfABC-LS	MABC-LS	ImpABC-LS	CABC-LS	GbdABC-LS	RABC-LS	IABC-LS
f_1	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	6.45E−12	3.62E−12	5.05E−12	2.58E−11	2.59E−12	7.39E−12	4.09E−11	3.60E−12	2.98E−12
f_3	5.63E+01	2.86E+01	4.25E+01	3.97E+01	5.38E+01	8.64E+01	1.46E+02	5.06E+01	3.83E+01
f_4	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_5	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_6	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_8	1.79E+00	1.42E+00	1.53E+00	2.44E+00	4.88E+02	4.45E−01	2.56E+00	1.72E+00	1.22E+00
f_9	1.00E−14	1.00E−14	1.00E−14	1.21E−06	2.72E−01	1.14E−07	2.94E−06	1.00E−14	1.00E−14
f_{10}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	4.48E−08	1.00E−14	2.11E−08	1.49E−06	1.09E+00	3.76E−07	2.38E−06	1.00E−14	1.00E−14
f_{12}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	8.10E−06	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{13}	1.17E+01	2.29E+00	1.15E+01	1.13E+01	8.91E+00	5.56E+00	7.38E+00	5.60E+00	2.00E+01
f_{14}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	5.55E−04	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{15}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{16}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	2.74E−04	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{17}	2.56E+01	1.98E+01	3.01E+01	1.57E+01	2.81E+01	3.03E+01	2.01E+01	2.78E+01	2.23E+01
f_{18}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	7.69E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{19}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14

(Win, Draw, Loss)

ts vs dp	(19, 0, 0)	(2, 14, 3)	(19, 0, 0)	(3, 2, 14)	(19, 0, 0)	(6, 7, 6)	(2, 13, 4)	(9, 4, 6)	(15, 3, 1)
LS vs dp	(19, 0, 0)	(2, 14, 3)	(19, 0, 0)	(14, 5, 0)	(19, 0, 0)	(9, 7, 3)	(2, 12, 5)	(12, 4, 3)	(13, 3, 3)
LS vs ts	(9, 7, 3)	(4, 14, 1)	(15, 3, 1)	(17, 2, 0)	(19, 0, 0)	(8, 8, 3)	(4, 11, 4)	(8, 9, 2)	(4, 12, 3)

Table E.7: Given are the medians of the various ABC variants on the 500 dimensional functions from the SOCO benchmark set for the default parameter settings (upper part), the tuned parameter settings (middle part) and the tuned hybrid ABC algorithms with local search (bottom part). At the bottom is indicated the number of times the tuned parameter settings give lower, same or worse median (win, draw, loss) than default parameter settings (ts vs dp) and the same information for the comparisons between the hybrids with local search versus default parameter settings (LS vs dp) and hybrids with local search versus tuned settings (LS vs ts).

default parameter settings

Function	ABC	GbABC	BsfABC	MABC	ImpABC	CABC	GbdABC	RABC	IABC
f_1	4.68E−04	1.00E−14	1.49E+05	3.78E−02	4.19E+03	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	1.49E+02	8.46E+01	4.95E+01	1.21E+02	4.87E+01	1.44E+02	8.28E+01	8.09E+01	1.12E+02
f_3	5.72E+02	2.99E+01	3.26E+10	2.95E+03	4.14E+08	1.40E+01	7.94E+01	1.95E+01	1.64E+01
f_4	1.19E+02	1.00E−14	2.38E+03	1.49E+03	4.10E+03	1.99E+00	1.00E−14	3.08E+00	1.74E+01
f_5	3.17E−04	1.00E−14	1.20E+03	7.63E−03	2.26E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_6	7.38E−01	1.00E−14	1.74E+01	2.68E+00	1.93E+01	1.00E−14	1.00E−14	4.31E−10	9.15E−08
f_7	1.74E−02	1.00E−14	5.25E+01	5.46E−06	1.02E+02	1.00E−14	1.00E−14	2.98E−12	2.35E−09
f_8	8.66E+05	5.56E+05	3.63E+05	1.87E+06	1.57E+05	6.79E+05	5.25E+05	4.14E+05	2.89E+05
f_9	3.84E+02	1.00E−14	2.49E+03	2.80E+03	3.13E+03	1.70E−07	1.00E−14	1.15E+00	1.29E+01
f_{10}	1.40E−03	1.00E−14	1.79E+03	2.63E+01	1.18E+02	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	3.94E+02	1.00E−14	2.48E+03	2.73E+03	3.16E+03	1.47E−07	1.00E−14	1.17E+00	1.33E+01
f_{12}	4.68E+01	1.00E−14	1.12E+05	6.63E+02	3.68E+03	8.83E−10	1.00E−14	1.89E−01	1.73E+00
f_{13}	5.34E+02	4.55E+01	2.03E+10	2.14E+03	2.16E+08	1.94E+01	1.13E+02	6.30E+00	2.34E+01
f_{14}	1.08E+02	1.03E+00	1.89E+03	1.18E+03	3.13E+03	1.99E+00	9.95E−01	2.17E+00	1.32E+01
f_{15}	9.30E−03	1.00E−14	1.77E+02	3.62E+00	1.79E+02	1.00E−14	1.00E−14	1.45E−12	1.31E−09
f_{16}	1.07E+02	1.00E−14	6.45E+04	1.45E+03	3.35E+03	5.58E−09	1.00E−14	4.49E−01	4.12E+00
f_{17}	2.17E+02	1.06E+00	3.07E+09	3.26E+03	2.79E+06	4.00E+00	1.42E+00	2.92E+00	1.80E+01
f_{18}	8.43E+01	2.28E−03	8.62E+02	7.88E+02	1.37E+03	3.03E−02	9.95E−01	2.66E−01	6.01E+00
f_{19}	2.95E−03	1.00E−14	2.51E+02	1.99E+01	1.70E+02	1.00E−14	1.00E−14	1.79E−13	1.63E−10

tuned parameter settings

Function	ABC-ts	GbABC-ts	BsfABC-ts	MABC-ts	ImpABC-ts	CABC-ts	GbdABC-ts	RABC-ts	IABC-ts
f_1	1.00E−14	1.00E−14	1.00E−14	4.57E+00	3.09E+03	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	1.08E+02	6.72E+01	4.94E+01	1.35E+02	4.71E+01	1.02E+02	5.83E+01	2.21E+01	9.61E+01
f_3	3.86E+01	2.57E+01	4.41E+02	4.58E+04	2.46E+08	2.46E+00	2.21E+02	5.07E+01	9.79E+00
f_4	2.29E+01	1.00E−14	9.40E+02	2.08E+03	3.56E+03	3.43E−08	1.13E+00	7.56E+01	2.98E+00
f_5	1.00E−14	1.00E−14	3.48E−10	7.43E−01	2.24E+01	1.00E−14	4.01E−04	1.00E−14	1.00E−14
f_6	1.00E−14	1.00E−14	1.01E+00	1.98E+01	1.91E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	1.26E−02	1.34E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_8	5.43E+05	5.27E+05	9.15E+04	1.23E+06	8.50E+04	5.06E+05	5.06E+05	1.12E+05	3.20E+05
f_9	3.13E−06	1.00E−14	3.04E+02	3.48E+03	2.96E+03	6.63E−03	1.00E−14	7.72E+00	1.00E−14
f_{10}	1.00E−14	1.00E−14	4.20E+01	4.20E+01	1.30E+02	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	1.07E−02	1.00E−14	1.54E+02	3.47E+03	2.94E+03	6.29E−03	1.00E−14	8.96E+00	1.00E−14
f_{12}	2.89E−07	1.00E−14	6.28E−02	9.64E+02	2.68E+03	7.45E−04	2.26E−01	1.10E−01	1.00E−14
f_{13}	3.95E+01	6.83E+01	6.59E+02	2.75E+03	7.09E+07	7.86E+00	1.88E+02	5.48E+01	4.14E+00
f_{14}	1.69E+01	9.95E−01	6.42E+02	1.66E+03	2.75E+03	8.86E−05	1.35E+00	4.88E+01	9.95E−01
f_{15}	1.00E−14	1.00E−14	1.05E+00	7.54E+00	6.15E+01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{16}	1.19E−06	1.00E−14	1.84E−01	1.97E+03	2.97E+03	3.01E−03	1.08E−01	3.03E−02	1.00E−14
f_{17}	1.02E+00	1.29E+01	1.15E+02	3.60E+03	4.35E+03	5.99E−01	2.30E+01	3.20E+01	1.49E+00
f_{18}	5.97E+00	7.02E−07	5.46E+02	9.39E+02	1.21E+03	1.98E−04	1.97E−06	1.79E+01	1.00E−14
f_{19}	1.00E−14	1.00E−14	2.10E+00	2.52E+01	1.17E+02	1.00E−14	1.00E−14	1.00E−14	1.00E−14

hybrids with local search

Function	ABC-LS	GbABC-LS	BsfABC-LS	MABC-LS	ImpABC-LS	CABC-LS	GbdABC-LS	RABC-LS	IABC-LS
f_1	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_2	1.82E−02	1.18E−02	1.31E−02	1.69E−02	7.89E−03	1.04E−02	1.69E−02	1.80E−02	1.09E−02
f_3	6.63E+00	2.21E+01	9.05E+00	1.22E+01	4.62E+00	9.84E+00	1.26E+01	2.12E+01	2.52E+01
f_4	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_5	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_6	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_7	1.00E−14	1.00E−14	1.00E−14	3.57E−14	1.00E−14	1.00E−14	7.52E−14	1.00E−14	1.00E−14
f_8	3.69E+04	3.23E+04	3.83E+04	3.78E+04	1.15E+05	1.06E+04	3.70E+04	3.59E+04	3.15E+04
f_9	4.23E−05	2.73E−06	6.40E−06	1.49E−05	4.46E+00	4.10E−06	2.76E−05	1.72E−06	8.16E−06
f_{10}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{11}	1.19E−05	2.38E−05	1.09E−05	1.57E−05	3.71E+00	6.77E−06	2.98E−05	1.86E−05	5.10E−06
f_{12}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.08E−01	1.00E−14	1.00E−14	1.00E−14	1.00E−14
f_{13}	3.93E+01	4.05E+01	6.60E+01	6.62E+01	1.89E+01	4.63E+01	4.26E+01	5.60E+01	2.16E+01
f_{14}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.10E−01	6.60E−05	1.00E−14	1.00E−14	1.00E−14
f_{15}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	4.16E−14	1.00E−14	1.00E−14
f_{16}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	5.95E−01	1.34E+00	1.00E−14	1.00E−14	1.00E−14
f_{17}	1.75E+01	1.32E+01	9.70E+00	2.00E+01	2.35E+01	1.87E+01	1.21E+01	8.51E+00	1.74E+01
f_{18}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.10E+01	1.27E+01	1.00E−14	1.00E−14	1.00E−14
f_{19}	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14	1.00E−14

(Win, Draw, Loss)

ts vs dp	(19, 0, 0)	(5, 12, 2)	(19, 0, 0)	(1, 0, 18)	(18, 0, 1)	(8, 7, 4)	(3, 8, 8)	(8, 3, 8)	(15, 3, 1)
LS vs dp	(19, 0, 0)	(6, 10, 3)	(19, 0, 0)	(19, 0, 0)	(19, 0, 0)	(6, 7, 6)	(6, 8, 5)	(13, 3, 3)	(15, 3, 1)
LS vs ts	(10, 7, 2)	(6, 10, 3)	(17, 2, 0)	(19, 0, 0)	(18, 0, 1)	(7, 7, 5)	(11, 3, 4)	(11, 7, 2)	(4, 10, 5)

Bibliography

K. Abhishek, S. Leyffer, and J. T. Linderoth. Modeling without categorical variables: a mixed-integer nonlinear program for the optimization of thermal insulation systems. *Optimization and Engineering*, 11(2):185–212, 2010.

A. Abraham, R. K. Jatoth, and A. Rajasekhar. Hybrid differential artificial bee colony algorithm. *Journal of Computational and Theoretical Nanoscience*, 9(2): 249–257, 2012.

M. A. Abramson. *Pattern search algorithms for mixed variable general constrained optimization problems*. PhD thesis, École Polytechnique de Montréal, Canada, 2002.

M. A. Abramson. Mixed variable optimization of a load-bearing thermal insulation system using a filter pattern search algorithm. *Optimization and Engineering*, 5(2):157–177, 2004.

M. A. Abramson, C. Audet, J. W. Chrissis, and J. G. Walston. Mesh adaptive direct search algorithms for mixed variable optimization. *Optimization Letters*, 3(1):35–47, 2009.

D. H. Ackley. *A connectionist machine for genetic hillclimbing*. Kluwer, Boston, 1987.

B. Adenso-Diaz and M. Laguna. Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, 54:99–114, 2006.

B. Akay and D. Karaboga. Artificial bee colony algorithm for large-scale problems and engineering design optimization. *Journal of Intelligent Manufacturing*. In press.

B. Akay and D. Karaboga. Parameter tuning for the artificial bee colony algorithm. In N. T. Nguyen, R. Kowalczyk, and S.-M. Chen, editors, *Proceedings of the 1st International Conference on Computational Collective Intelligence. Semantic*

Web, Social Networks and Multiagent Systems, ICCCI'09., LNCS 5796, pages 608–619. Springer Verlag, Heidelberg, Germany, 2009.

B. Akay and D. Karaboga. A modified artificial bee colony algorithm for real-parameter optimization. *Information Sciences*, 192:120–142, 2012.

M. S. Alam, M. W. Ul Kabir, and M. M. Islam. Self-adaptation of mutation step size in artificial bee colony algorithm for continuous function optimization. In *Proceedings of International Conference on Computer and Information Technology, ICCIT'10*, pages 69–74. IEEE Press, Piscataway, NJ, 2010.

B. Alataş. Chaotic bee colony algorithms for global numerical optimization. *Expert Systems with Applications*, 37(8):5682–5687, 2010.

N. Andréasson, A. Evggrafov, and M. Patriksson. *An introduction to continuous optimization*. Lightning Source Incorporated, 2005.

C. Audet and J. E. Dennis, Jr. Pattern search algorithms for mixed variable programming. *SIAM Journal on Optimization*, 11(3):573–594, 2001.

C. Audet and J. E. Dennis Jr. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2006.

A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Proceeding of IEEE Congress on Evolutionary Computation, CEC'05*, pages 1769–1776, Piscataway, NJ, USA, 2005. IEEE Press.

D. Aydın, T. Liao, M. A. Montes de Oca, and T. Stützle. Improving performance via population growth and local search: The case of the artificial bee colony algorithm. In J.-K. Hao, P. Legrand, P. Collet, N. Monmarché, E. Lutton, and M. Schoenauer, editors, *Artificial Evolution*, LNCS 7401, pages 85–96. Springer Verlag, Heidelberg, Germany, 2012.

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

A. Banharnsakun, T. Achalakul, and B. Sirinaovakul. The best-so-far selection in artificial bee colony algorithm. *Applied Soft Computing*, 11(2):2888–2901, 2011.

T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation: the New Experimentalism*. Springer, 2006.

R. Battiti. First-and second-order methods for learning: between steepest descent and newton's method. *Neural computation*, 4(2):141–166, 1992.

H. Bersini, M. Dorigo, S. Langerman, G. Seront, and L. Gambardella. Results of the first international contest on evolutionary optimisation (1st ICEO). In *Proceedings of IEEE International Conference on Evolutionary Computation, ICEC'96*, pages 611–615, Piscataway, NJ, 1996. IEEE Press.

H.-G. Beyer and H.-P. Schwefel. Evolution strategies - a comprehensive introduction. *Natural Computing*, 1:3–52, 2002.

G. Bilchev and I. Parmee. The Ant Colony Metaphor for Searching Continuous Design Spaces. In T. Fogarty, editor, *AISB Workshop on Evolutionary Computing*, pages 25–39. Springer-Verlag, 1995.

M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*. Springer, Berlin, Germany, 1st ed. 2005. 2nd printing edition, 2009.

M. Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'02*, pages 11–18, San Francisco, CA, USA, 2002. Morgan Kaufmann.

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

B. Bischl, O. Mersmann, H. Trautmann, and M. Preuß. Algorithm selection based on exploratory landscape analysis and cost-sensitive learning. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO'12*, pages 313–320, New York, NY, USA, 2012. ACM.

G. E. Box. Evolutionary operation: A method for increasing industrial productivity. *Applied Statistics*, pages 81–101, 1957.

D. Brockhoff, A. Auger, and N. Hansen. On the impact of active covariance matrix adaptation in the CMA-ES with mirrored mutations and small initial

population size on the noiseless BBOB testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'12 (Companion)*, pages 291–296, New York, NY, USA, 2012. ACM.

S. Brooks. A discussion of random methods for seeking maxima. *Operations Research*, 6(2):244–251, 1958.

M. R. Bussieck and A. Pruessner. Mixed-integer nonlinear programming. *SIAG/OPT Newsletter: Views & News*, 14(1):19–22, 2003.

Y. Cao and Q. Wu. Mechanical design optimization by mixed-variable evolutionary programming. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pages 443–446. IEEE Press, Piscataway, NJ, 1997.

J. Chen and Y. Tsao. Optimal design of machine elements using genetic algorithms. *Journal of the Chinese Society of Mechanical Engineers*, 14(2):193–199, 1993.

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

C. A. Coello Coello. Use of a self-adaptive penalty approach for engineering optimization problems. *Computers in Industry*, 41(2):113–127, 2000.

C. A. Coello Coello and R. L. Becerra. Efficient evolutionary optimization through the use of a cultural algorithm. *Engineering Optimization*, 36(2):219–236, 2004.

C. A. Coello Coello and E. Mezura Montes. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3):193–203, 2002.

A. R. Conn, N. I. Gould, and P. L. Toint. *Trust region methods*, volume 1. SIAM, 1987.

A. R. Conn, K. Scheinberg, and L. N. Vicente. *Introduction to derivative-free optimization*. Society for Industrial and Applied Mathematics, 2009.

W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, USA, 1998.

G. B. Dantzig and M. N. Thapa. *Linear Programming 1: Introduction*. Springer, 1997.

G. B. Dantzig and M. N. Thapa. *Linear Programming 2: Theory and Extensions*. Springer, 2003.

S. Das, A. Abraham, U. K. Chakraborty, and A. Konar. Differential evolution using a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553, 2009.

S. Das, P. N. Suganthan, and C. A. Coello Coello. Guest editorial special issue on differential evolution. *IEEE Transactions on Evolutionary Computation*, 15(1): 1–3, 2011.

D. Datta and J. Figueira. A real-integer-discrete-coded differential evolution algorithm: A preliminary study. In P. Cowling and P. Merz, editors, *Evolutionary Computation in Combinatorial Optimization*, LNCS 6022, pages 35–46. Springer Verlag, Heidelberg, Germany, 2010.

W. C. Davidon. Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17, 1991.

C. De Boor and A. Ron. On multivariate polynomial interpolation. *Constructive Approximation*, 6(3):287–302, 1990.

K. Deb and M. Goyal. A combined genetic adaptive search (GeneAS) for engineering design. *Computer Science and Informatics*, 26:30–45, 1996.

K. Deb and M. Goyal. A flexible optimization procedure for mechanical component design based on genetic adaptive search. *Journal of Mechanical Design*, 120(2): 162–164, 1998.

J. E. Dennis, Jr and V. Torczon. Direct search methods on parallel machines. *SIAM Journal on Optimization*, 1(4):448–474, 1991.

G. G. Dimopoulos. Mixed-variable engineering optimization based on evolutionary and social metaphors. *Computer Methods in Applied Mechanics and Engineering*, 196(4-6):803 – 817, 2007.

K. Diwold, A. Aderhold, A. Scheidler, and M. Middendorf. Performance evaluation of artificial bee colony optimization and new selection schemes. *Memetic Computing*, 3(3):149–162, 2011a.

K. Diwold, M. Beekman, and M. Middendorf. Honeybee optimisation-an overview and a new bee inspired optimisation scheme. In B. K. Panigrahi, Y. Shi, and

M.-H. Lim, editors, *Handbook of Swarm Intelligence-Concepts, Principles and Application*, volume 8 of *Adaptation, Learning, and Optimization*, pages 295–328. Springer Verlag, Berlin, Germany, 2011b.

M. Dorigo and G. Di Caro. The Ant Colony Optimization meta-heuristic. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK, 1999.

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

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

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

B. Dorronsoro and P. Bouvry. Improving classical and decentralized differential evolution with new mutation operator and population topologies. *IEEE Transactions on Evolutionary Computation*, 15(1):67–98, 2011.

J. Dréo and P. Siarry. Continuous interacting ant colony algorithm based on dense heterarchy. *Future Generation Computer Systems*, 20(5):841–856, 2004.

J. Dubois-Lacoste, M. López-Ibáñez, and T. Stützle. Automatic configuration of state-of-the-art multi-objective optimizers using the TP+PLS framework. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'11*, pages 2019–2026, New York, NY, USA, 2011. ACM.

A. Eiben and T. Bäck. Empirical investigation of multiparent recombination operators in evolution strategies. *Evolutionary Computation*, 5(3):347–365, 1997.

M. El-Abd. Opposition-based artificial bee colony algorithm. In N. Krasnogor and P. L. Lanzi, editors, *Proceedings of Genetic and Evolutionary Computation Conference, GECCO'11*, pages 109–116. ACM, 2011a.

M. El-Abd. A hybrid ABC-SPSO algorithm for continuous function optimization. In *Proceedings of IEEE Symposium on Swarm Intelligence*, pages 1–6. IEEE Press, Piscataway, NJ, 2011b.

M. Epitropakis, D. Tasoulis, N. Pavlidis, V. Plagianakos, and M. Vrahatis. Enhancing differential evolution utilizing proximity-based mutation operators. *IEEE Transactions on Evolutionary Computation*, 15(1):99–119, 2011.

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

I. Fister, I. Fister Jr., J. Brest, and V. Zumer. Memetic artificial bee colony algorithm for large-scale global optimization. In *IEEE Congress on Evolutionary Computation, CEC'12*, pages 1–8. IEEE Press, Piscataway, NJ, 2012.

J.-F. Fu, R. Fenton, and W. Cleghorn. A mixed integer-discrete-continuous programming method and its application to engineering design optimization. *Engineering Optimization*, 17(4):263–280, 1991.

L. Gao and A. Hailu. Comprehensive learning particle swarm optimizer for constrained mixed-variable optimization problems. *International Journal of Computational Intelligence Systems*, 3(6):832–842, 2010.

W. Gao and S. Liu. Improved artificial bee colony algorithm for global optimization. *Information Processing Letters*, 111(17):871–882, 2011.

W. Gao, S. Liu, and L. Huang. A global best artificial bee colony algorithm for global optimization. *Journal of Computational and Applied Mathematics*, 236 (11):2741–2753, 2012.

S. Ghosh, S. Das, S. Roy, S. K. Minhazul Islam, and P. N. Suganthan. A differential covariance matrix adaptation evolutionary algorithm for real parameter optimization. *Information Science*, 182(1):199–219, 2012.

J. Gimmler. Metaheuristiken zur kontiuierlichen, globalen optimierung. Master's thesis, Computer Science Department, TU, Darmstadt, Germany, 2005.

J. Gimmler, T. Stützle, and T. E. Exner. Hybrid particle swarm optimization: An examination of the influence of iterative improvement algorithms on performance. In M. Dorigo et al., editors, *Proceedings of International Conference on Ant Colony Optimization and Swarm Intelligence, ANTS'06*, LNCS 4150, pages 436–443. Springer, Heidelberg, Germany, 2006.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley, Boston, MA, USA, 1989.

C. P. Gomes and B. Selman. Algorithm portfolios. *Artificial Intelligence*, 126(12): 4–62, 2001.

T. Gönen. *Electric power distribution system engineering.* McGraw-Hill, New York, USA, 1986.

A. Griewank. Generalized descent for global optimization. *Journal of optimization theory and applications*, 34(1):11–39, 1981.

I. Griva, S. G. Nash, and A. Sofer. *Linear and nonlinear optimization.* Society for Industrial and Applied Mathematics, 2009.

M. Guntsch and M. Middendorf. A population based approach for ACO. In S. Cagnoni et al., editors, *Proceedings of EvoWorkshops 2002: EvoCOP, EvoIASP, EvoSTim*, volume 2279 of *LNCS*, pages 71–80. Springer, Berlin, Germany, 2002.

C. Guo, J. Hu, B. Ye, and Y. Cao. Swarm intelligence for mixed-variable design optimization. *Journal of Zhejiang University Science*, 5(7):851–860, 2004.

P. Guo, W. Cheng, and J. Liang. Global artificial bee colony search algorithm for numerical function optimization. In *Proceedings of International Conference on Natural Computation*, volume 3, pages 1280–1283. IEEE Press, Piscataway, NJ, 2011.

N. Hansen. Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'12 (Companion)*, pages 2389–2396, New York, NY, USA, 2009. ACM.

N. Hansen. The CMA evolution strategy: A tutorial, 2010. Online: http://www.lri. fr/hansen/cmatutorial. pdf.

N. Hansen and A. Ostermeier. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In *Proceedings of IEEE International Conference on Evolutionary Computation, CEC'96*, pages 312–317, Piscataway, NJ, 1996. IEEE Press.

N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

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

N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. 2009a. Technical Report RR-6829, INRIA.

N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions. Technical Report Research Report RR-6869, INRIA, 2009b.

N. Hansen, A. S. P. Niederberger, L. Guzzella, and P. Koumoutsakos. A method for handling uncertainty in evolutionary optimization with an application to feedback control of combustion. *IEEE Transactions on Evolutionary Computation*, 13(1):180–197, 2009c.

Q. He and L. Wang. An effective co-evolutionary particle swarm optimization for constrained engineering design problems. *Engineering Applications of Artificial Intelligence*, 20(1):89–99, 2007a.

Q. He and L. Wang. A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. *Applied Mathematics and Computation*, 186 (2):1407–1422, 2007b.

F. Herrera and M. Lozano. Two-loop real-coded genetic algorithms with adaptive control of mutation step sizes. *Applied Intelligence*, 13(3):187–204, 2000.

F. Herrera, M. Lozano, and J. Verdegay. Tackling real-coded genetic algorithms: Operators and tools for behavioural analysis. *Artificial Intelligence Review*, 12: 265–319, 1998.

F. Herrera, M. Lozano, and A. Sánchez. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. *International Journal of Intelligent Systems*, 18(3):309–338, 2003.

F. Herrera, M. Lozano, and A. Sánchez. Hybrid crossover operators for real-coded genetic algorithms: an experimental study. *Soft Computing*, 9(4):280–298, 2005.

F. Herrera, M. Lozano, and D. Molina. Test suite for the special issue of soft computing on scalability of evolutionary algorithms and other meta-

heuristics for large scale continuous optimization problems, 2010. URL: `http://sci2s.ugr.es/eamhco/`.

R. Hinterding. Gaussian mutation and self-adaption for numeric genetic algorithms. In *Proceeding of IEEE Congress on Evolutionary Computation, CEC'95*, page 384, Piscataway, NJ, 1995. IEEE Press.

J. H. Holland. *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. University of Michigan Press, 1975.

R. Hooke and T. A. Jeeves. "Direct Search" solution of numerical and statistical problems. *Journal of the ACM*, 8(2):212–229, 1961.

H. H. Hoos. Programming by optimization. *Communications of the ACM*, 55(2): 70–80, 2012.

H. H. Hoos and T. Stützle. *Stochastic Local Search: Foundations & Applications*. Morgan Kaufmann, San Francisco, CA, USA, 2005.

X.-M. Hu, J. Zhang, and Y. Li. Orthogonal methods based ant colony search for solving continuous optimization problems. *Journal of Computer Science and Technology*, 23:2–18, 2008.

X.-M. Hu, J. Zhang, H. S.-H. Chung, Y. Li, and O. Liu. SamACO: Variable sampling ant colony optimization algorithm for continuous optimization. *IEEE Transactions on Systems, Man, and Cybernetics - Part B: Cybernetics*, 40:1555–1566, 2010.

B. A. Huberman, R. M. Lukose, and T. Hogg. An economics approach to hard computational problems. *Science*, 275(5296):51–54, 1997.

F. Hutter, D. Babic, H. Hoos, and A. Hu. Boosting verification by automatic tuning of decision procedures. In *Proceedings of Formal Methods in Computer Aided Design, FMCAD'07*, pages 27–34, Piscataway, NJ, 2007. IEEE Press.

F. Hutter, H. Hoos, K. Leyton-Brown, and K. Murphy. An experimental investigation of model-based parameter optimisation: SPO and beyond. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'09*, pages 271–278, New York, NY, USA, 2009a. ACM.

F. Hutter, H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, 36:267–306, 2009b.

S. G. Johnson. The nlopt nonlinear-optimization package., 2008. URL: http://ab-initio.mit.edu/nlopt.

N. C. Jones and P. Pevzner. *An introduction to bioinformatics algorithms*. MIT Press, 2004.

K. D. Jong. *An analysis of the behavior of a class of genetic adaptive systems*. PhD thesis, Department of Computer and Communication Sciences, University of Michigan, 1975.

F. Kang, J. Li, and Z. Ma. Rosenbrock artificial bee colony algorithm for accurate global optimization of numerical functions. *Information Sciences*, 181(16):3508–3531, 2011.

D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, Engineering Faculty, Computer Engineering Departmen, 2005.

D. Karaboga and B. Akay. A survey: Algorithms simulating bee swarm intelligence. *Artificial Intelligence Review*, 31(1–4):61–85, 2009.

D. Karaboga and B. Basturk. A powerful and efficient algorithm for numerical function optimization: Artificial bee colony (ABC) algorithm. *Journal of Global Optimization*, 39(3):459–471, 2007.

D. Karaboga and B. Basturk. On the performance of artificial bee colony (ABC) algorithm. *Applied Soft Computing*, 8(1):687–697, 2008.

A. Kayhan, H. Ceylan, M. Ayvaz, and G. Gurarslan. PSOLVER: A new hybrid particle swarm optimization algorithm for solving continuous optimization problems. *Expert Systems with Applications*, 37(10):6798–6808, 2010.

C. Kelley. Detection and remediation of stagnation in the nelder–mead algorithm using a sufficient decrease condition. *SIAM Journal on Optimization*, 10(1): 43–55, 1999.

J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948, Piscataway, NJ, USA, 1995. IEEE Press.

J. Kennedy and R. C. Eberhart. *Swarm intelligence*. Morgan Kaufmann, San Francisco, CA, USA, 2001.

J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proceedings of IEEE Congress on Evolutionary Computation, CEC'02*, volume 2, pages 1671–1676, Piscataway, NJ, 2002. IEEE Press.

A. R. KhudaBukhsh, L. Xu, H. H. Hoos, and K. Leyton-Brown. SATenstein: Automatically building local search SAT solvers from components. In *Proceedings of International Joint Conference on Artificial Intelligence, IJCAI'09*, pages 517–524, 2009.

M. Kokkolaras, C. Audet, and J. Dennis Jr. Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system. *Optimization and Engineering*, 2(1):5–29, 2001.

T. G. Kolda, R. M. Lewis, and V. Torczon. Optimization by direct search: New perspectives on some classical and modern methods. *SIAM review*, 45(3):385–482, 2003.

O. Kramer. Iterated local search with Powell's method: a memetic algorithm for continuous global optimization. *Memetic Computing*, 2:69–83, 2010.

J. Lampinen and I. Zelinka. Mixed integer-discrete-continuous optimization by differential evolution - part 1: the optimization method. In P. Ošmera, editor, *Proceedings of 5th International Mendel Conference of Soft Computing*, pages 71–76. Brno University of Technology, Brno, Czech Republic, 1999a.

J. Lampinen and I. Zelinka. Mixed integer-discrete-continuous optimization by differential evolution. Part 2: a practical example. In P. Ošmera, editor, *Proceedigns of 5th International Mendel Conference of Soft Computing*, pages 77–81. Brno University of Technology, Brno, Czech Republic, 1999b.

J. Lampinen and I. Zelinka. Mechanical engineering design optimization by differential evolution. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 127–146. McGraw-Hill, London, UK, 1999c.

P. Larrañaga and J. A. Lozano, editors. *Estimation of distribution algorithms: A new tool for evolutionary computation*. Kluwer Academic Publishers, Boston, MA, 2002.

A. LaTorre, S. Muelas, and J.-M. Peña. A MOS-based dynamic memetic differential evolution algorithm for continuous optimization: a scalability test. *Soft Computing*, 15(11):2187–2199, 2011.

W.-P. Lee and W.-T. Cai. A novel artificial bee colony algorithm with diversity strategy. In *Proceedings of International Conference on Natural Computation*, volume 3, pages 1441–1444. IEEE Press, Piscataway, NJ, 2011.

G. Leguizamón and C. Coello. An alternative ACOR algorithm for continuous optimization problems. In M. Dorigo et al., editors, *Proceedings of the Seventh International Conference on Swarm Intelligence, ANTS'10*, LNCS 6234, pages 48–59. Springer, Berlin, Germany, 2010.

H.-L. Li and C.-T. Chou. A global approach for nonlinear mixed discrete programing in design optimization. *Engineering Optimization*, 22:109–122, 1994.

J. J. Liang, A. Qin, P. N. Suganthan, and S. Baskar. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. *IEEE Transactions on Evolutionary Computation*, 10(3):281–295, 2006.

T. Liao and T. Stützle. Bounding the population size of IPOP-CMA-ES on the noiseless BBOB testbed. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO'13(Companion)*, New York, NY, USA, 2013a. ACM. Accepted.

T. Liao and T. Stützle. Testing the impact of parameter tuning on a variant of IPOP-CMA-ES with a bounded maximum population size on the noiseless BBOB testbed. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO'13(Companion)*, New York, NY, USA, 2013b. ACM. Accepted.

T. Liao and T. Stützle. Expensive optimization scenario: IPOP-CMA-ES with a population bound mechanism for noiseless function testbed. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO'13(Companion)*, New York, NY, USA, 2013c. ACM. Accepted.

T. Liao and T. Stützle. A simple and effective cooperative-competitive hybrid algorithm for continuous optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 2013. Submitted.

T. Liao, D. Molina, M. Montes de Oca, and T. Stützle. A note on the effects of enforcing bound constraints on algorithm comparisons using the IEEE CEC'05

benchmark function suite. Technical Report TR/IRIDIA/2011-010, IRIDIA, Université Libre de Bruxelles, Belgium, 2011a.

T. Liao, M. A. Montes de Oca, D. Aydın, T. Stützle, and M. Dorigo. An incremental ant colony algorithm with local search for continuous optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'11*, pages 125–132, New York, NY, USA, 2011b. ACM.

T. Liao, M. A. Montes de Oca, and T. Stützle. Tuning parameters across mixed dimensional instances: A performance scalability study of Sep-G-CMA-ES. In *Proceedings of the Workshop on Scaling Behaviours of Landscapes, Parameters and Algorithms of the Genetic and Evolutionary Computation Conference, GECCO'11*, pages 703–706, New York, NY, USA, 2011c. ACM.

T. Liao, D. Molina, T. Stützle, M. A. Montes de Oca, and M. Dorigo. An ACO algorithm benchmarked on the BBOB noiseless function testbed. In *Proceedings of Conference on Genetic and Evolutionary Computation Conference, GECCO'12(Companion)*, pages 159–166, New York, NY, USA, 2012. ACM.

T. Liao, M. A. Montes de Oca, and T. Stützle. Computational results for an automatically tuned CMA-ES with increasing population size on the CEC'05 benchmark set. *Soft Computing*, 17(6), 2013a.

T. Liao, K. Socha, M. A. Montes de Oca, T. Stützle, and M. Dorigo. Ant colony optimization for mixed-variable optimization problems. *IEEE Transactions on Evolutionary Computation*, 2013b. Conditionally accepted.

H. Loh and P. Papalambros. Computation implementation and test of a sequential linearization approach for solving mixed-discrete nonlinear design optimization. *Journal of Mechanical Design*, 113(3):335–345, 1991.

M. López-Ibáñez and T. Stützle. Automatic configuration of multi-objective aco algorithms. In M. Dorigo et al., editors, *Proceedings of the Seventh International Conference on Swarm Intelligence, ANTS'10*, LNCS 6234, pages 95–106. IEEE Press, Piscataway, NJ, 2010.

M. López-Ibáñez and T. Stützle. The automatic design of multi-objective ant colony optimization algorithms. *IEEE Transactions on Evolutionary Computation*, 16(6):861–875, 2012.

M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. The irace package, iterated race for automatic algorithm configuration. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université Libre de Bruxelles, Belgium, 2011. URL `http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf`.

M. López-Ibáñez, T. Liao, and T. Stützle. On the anytime behavior of IPOP-CMA-ES. In C. A. Coello Coello et al., editors, *PPSN 2012, Part I*, volume 7491 of *Lecture Notes in Computer Science*, pages 357–366. Springer, Heidelberg, Germany, 2012.

I. Loshchilov, M. Schoenauer, and M. Sebag. Alternative restart strategies for CMA-ES. In C. A. C. Coello et al., editors, *Proceedings of Parallel Problem Solving from Nature, PPSN'12*, LNCS 7491, pages 296–305, Heidelberg, Germany, 2012a. Springer.

I. Loshchilov, M. Schoenauer, and M. Sebag. Black-box optimization benchmarking of NIPOP-aCMA-ES and NBIPOP-aCMA-ES on the BBOB-2012 noiseless testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'12 (Companion)*, pages 269–276, New York, NY, USA, 2012b. ACM.

I. Loshchilov, M. Schoenauer, and M. Sebag. Black-box optimization benchmarking of IPOP-saACM-ES and BIPOP-saACM-ES on the BBOB-2012 noiseless testbed. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'12 (Companion)*, pages 175–182, New York, NY, USA, 2012c. ACM.

I. Loshchilov, M. Schoenauer, and M. Sebag. Self-adaptive surrogate-assisted covariance matrix adaptation evolution strategy. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'12*, pages 321–328, New York, NY, USA, 2012d. ACM.

I. Loshchilov, M. Schoenauer, and M. Sebag. Bi-population cma-es algorithms with surrogate models and line searches. In *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO'13 (Companion)*, New York, NY, USA, 2013. ACM. Accepted.

H. R. Lourenço, O. Martin, and T. Stützle. Iterated local search: Framework and applications. In M. Gendreau and J.-Y. Potvin, editors, *Handbook of Metaheuristics*, volume 146 of *International Series in Operations Research & Management Science*, chapter 9, pages 363–397. Springer, New York, NY, 2 edition, 2010.

M. Lozano, F. Herrera, N. Krasnogor, and D. Molina. Real-coded memetic algorithms with crossover hill-climbing. *Evolutionary computation*, 12(3):273–302, 2004.

M. Lozano, D. Molina, and F. Herrera. Editorial scalability of evolutionary algorithms and other metaheuristics for large-scale continuous optimization problems. *Soft Computing*, 15(11):2085–2087, 2011.

C. B. Lucasius and G. Kateman. Application of genetic algorithms in chemometrics. In *Proceedings of the third international conference on Genetic algorithms*, pages 170–176. Morgan Kaufmann Publishers Inc., 1989.

S. Lucidi, V. Piccialli, and M. Sciandrone. An algorithm model for mixed variable programming. *SIAM Journal on Optimization*, 15(4):1057–1084, 2005.

M. H. Mashinchi, M. A. Orgun, and W. Pedrycz. Hybrid optimization with improved tabu search. *Applied Soft Computing*, 11(2):1993–2006, 2011.

R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

O. Mersmann, B. Bischl, H. Trautmann, M. Preuß, C. Weihs, and G. Rudolph. Exploratory landscape analysis. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO'11*, pages 829–836, New York, NY, USA, 2011. ACM Press.

E. Mezura Montes and C. A. Coello Coello. Useful infeasible solutions in engineering optimization with evolutionary algorithms. In A. Gelbukh et al., editors, *Advances in Artificial Intelligence*, LNCS 3789, pages 652–662. Springer, Berlin, Germany, 2005.

Z. Michalewicz. *Genetic algorithms + data structures = evolution programs*. Springer-Verlag New York, Inc., 1992.

H. Ming, J. Baohui, and L. Xu. An improved bee evolutionary genetic algorithm. In *IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 1, pages 372–374. IEEE Press, Piscataway, NJ, 2010.

D. Molina, M. Lozano, C. García-Martínez, and F. Herrera. Memetic algorithms for continuous optimisation based on local search chains. *Evolutionary Computation*, 18(1):27–63, 2010a.

D. Molina, M. Lozano, and F. Herrera. MA-SW-Chains: Memetic algorithm based on local search chains for large scale continuous global optimization. In *Proceeding of IEEE Congress on Evolutionary Computation, CEC'10*, pages 1–8. IEEE Press, 2010b.

D. Molina, M. Lozano, A. Snchez, and F. Herrera. Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains. *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, 15:2201–2220, 2011.

N. Monmarché, G. Venturini, and M. Slimane. On how Pachycondyla apicalis ants suggest a new search algorithm. *Future Generation Computer Systems*, 16(9): 937–946, 2000.

M. A. Montes de Oca. *Incremental Social Learning in Swarm Intelligence Systems*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2011.

M. A. Montes de Oca, D. Aydın, and T. Stützle. An incremental particle swarm for large-scale continuous optimization problems: An example of tuning-in-the-loop (re)design of optimization algorithms. *Soft Computing*, 15(11):2233–2255, 2011.

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

R. W. Morrison and K. A. D. Jong. A test problem generator for non-stationary environments. In *Proceedings of the Congress on Evolutionary Computation, CEC'99*, pages 2047–2053, Piscataway, NJ, 1999. IEEE Press.

P. Moscato. Memetic algorithms: a short introduction. In *New ideas in optimization*, pages 219–234. McGraw-Hill Ltd, UK, 1999.

C. Müller, B. Baumgartner, and I. Sbalzarini. Particle swarm CMA evolution strategy for the optimization of multi-funnel landscapes. In *Proceeding of IEEE Congress on Evolutionary Computation, CEC'09*, pages 2685–2692, Piscataway, NJ, 2009. IEEE Press.

V. Nannen and A. E. Eiben. Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proceedings of International Joint Conference on Artifical Intelligence, IJCAI'07*, pages 975–980, San Francisco, CA, USA, 2007. Morgan Kaufmann.

J. A. Nelder and R. Mead. A simplex method for function minimization. *The computer journal*, 7(4):308–313, 1965.

G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.

J. Ocenasek and J. Schwarz. Estimation distribution algorithm for mixed continuous-discrete optimization problems. In *Proceedings of the 2nd Euro-International Symposium on Computational Intelligence*, pages 227–232. IOS Press, Amsterdam, The Netherlands, 2002.

U.-M. O'Reilly and F. Oppacher. Hybridized crossover-based search techniques for program discovery. In *Proceeding of IEEE Congress on Evolutionary Computation, CEC'95*, volume 2, pages 573–578, Piscataway, NJ, 1995. IEEE Press.

H.-O. Peitgen. *Newton's method and dynamical systems*. Kluwer Academic Pub, 1989.

F. Peng, K. Tang, G. Chen, and X. Yao. Population-based algorithm portfolios for numerical optimization. *IEEE Transactions on Evolutionary Computation*, 14(5):782–800, 2010.

Y. Petalas, K. Parsopoulos, and M. Vrahatis. Memetic particle swarm optimization. *Annals of Operations Research*, 156(1):99–127, 2007.

R. Poli, J. Kennedy, and T. Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007.

M. J. D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7(2): 155, 1964.

M. J. D. Powell. UOBYQA: unconstrained optimization by quadratic approximation. *Mathematical Programming*, 92(3):555–582, 2002.

M. J. D. Powell. The NEWUOA software for unconstrained optimization without derivativesn. In G. Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, pages 255–297. Springer, 2006.

M. J. D. Powell. The BOBYQA algorithm for bound constrained optimization without derivatives. *Cambridge NA Report NA2009/06, University of Cambridge, Cambridge*, 2009.

S. Praharaj and S. Azarm. Two-level non-linear mixed discrete-continuous optimization-based design: An application to printed circuit board assemblies. *Advances in Design Automation*, 1(44):307–321, 1992.

W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge Univ. Press, 1992.

C. J. Price, I. Coope, and D. Byatt. A convergent variant of the nelder–mead algorithm. *Journal of Optimization Theory and Applications*, 113(1):5–19, 2002.

A. Qin, V. Huang, and P. Suganthan. Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, 13(2):398–417, 2009.

A. Rajasekhar, A. Abraham, and M. Pant. Levy mutated artificial bee colony algorithm for global optimization. In *Proceedings of 2011 IEEE International Conference on Systems, Man, and Cybernetics*, pages 655–662. IEEE Press, Piscataway, NJ, 2011.

R. V. Rao, V. J. Savsani, and D. P. Vakharia. Teaching-learning-based optimization: A novel method for constrained mechanical design optimization problems. *Computer-Aided Design*, 43(3):303–315, 2011.

S. S. Rao and Y. Xiong. A hybrid genetic algorithm for mixed-discrete design optimization. *Journal of Mechanical Design*, 127(6):1100–1112, 2005.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. PhD thesis, Technical University of Berlin, 1971.

I. Rechenberg. *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzbook, Stuttgart, Germany, 1973.

J. R. Rice. The algorithm selection problem. *Advances in Computers*, 15:65–118, 1976.

R. Ros. Benchmarking sep-CMA-ES on the BBOB-2009 noisy testbed. In *Proceedings of Genetic and Evolutionary Computation Conference, GECCO'09(Companion)*, pages 2441–2446, New York, NY, USA, 2009. ACM.

R. Ros and N. Hansen. A simple modification in CMA-ES achieving linear time and space complexity. In G. Rudolph et al., editors, *Parallel Problem Solving from Nature, PPSN'08*, LNCS 5199, pages 296–305, Heidelberg, Germany, 2008.

H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *The Computer Journal*, 3(3):175–184, 1960.

F. Sambo, M. A. Montes de Oca, B. Di Camillo, G. Toffolo, and T. Stützle. MORE: Mixed optimization for reverse engineering: An application to modeling biological networks response via sparse systems of nonlinear differential equations. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 9(5): 1459–1471, 2012.

E. Sandgren. Nonlinear integer and discrete programming in mechanical design optimization. *Journal of Mechanical Design*, 112:223–229, 1990.

H. Schmidt and G. Thierauf. A combined heuristic optimization technique. *Advances in Engineering Software*, 36:11–19, 2005.

H.-P. Schwefel. *Evolutionsstrategie und Numerische Optimierung*. PhD thesis, Technical University of Berlin, 1975.

H. P. Schwefel. *Numerical optimization of computer models*. Wiley & Sons, 1981.

T. Sharma and M. Pant. Enhancing scout bee movements in artificial bee colony algorithm. In K. Deep, A. Nagar, M. Pant, and J. C. Bansal, editors, *Proceedings of the International Conference on Soft Computing for Problem Solving*, volume 130 of *Advances in Intelligent and Soft Computing*, pages 601–610. Springer India, 2012.

T. Sharma, M. Pant, and T. Bhardwaj. PSO ingrained artificial bee colony algorithm for solving continuous optimization problems. In *International Conference on Computer Applications and Industrial Electronics*, pages 108–112. IEEE Press, Piscataway, NJ, 2011.

Y. Shi and R. Eberhart. A modified particle swarm optimizer. In *Proceedings of IEEE Congress on Evolutionary Computation, CEC'98*, pages 69–73, Piscataway, NJ, 1998. IEEE Press.

S. Smit and A. Eiben. Comparing parameter tuning methods for evolutionary algorithms. In *IEEE Congress on Evolutionary Computation, CEC'09*, pages 399–406, Piscataway, NJ, 2009. IEEE Press.

S. Smit and A. Eiben. Beating the world champion evolutionary algorithm via REVAC tuning. In *Proceeding of IEEE Congress on Evolutionary Computation, CEC 2010*, pages 1–8, Piscataway, NJ, 2010. IEEE Press.

J. E. Smith and T. C. Fogarty. Operator and parameter adaptation in genetic algorithms. *Soft Computing*, 1(2):81–87, 1997.

K. Socha. *Ant Colony Optimization for Continuous and Mixed-Variable Domains*. PhD thesis, Université Libre de Bruxelles, Belgium, May 2008.

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

M. Stelmack and S. Batill. Concurrent subspace optimization of mixed continuous/discrete systems. In *Proceedings of AIAA/ASME/ASCE/AHS/ASC 38th Structures, Structural Dynamic and Materials Conference*. AIAA, Kissimmee, Florida, 1997.

R. Stern and K. Price. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11 (4):341–359, 1997.

J. Stoer, R. Bulirsch, R. Bartels, W. Gautschi, and C. Witzgall. *Introduction to numerical analysis*. Springer-Verlag New York, Inc., 1993.

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

P. N. Suganthan, N. Hansen, J. Liang, K. Deb, Y. Chen, A. Auger, and S. Tiwari. Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization. Technical Report 2005005, Nanyang Technological University, 2005.

E.-G. Talbi. A taxonomy of hybrid metaheuristics. *Journal of Heuristics*, 8(5): 541–564, 2002.

K. Tang, X. Yao, P. N. Suganthan, C. MacNish, Y. Chen, C. Chen, and Z. Yang. Benchmark functions for the CEC 2008 special session and competition on large scale global optimization. Technical report, Nature Inspired Computation and Applications Laboratory, USTC, China, 2007. URL: `http://nical.ustc.edu.cn/cec08ss.php`.

G. Thierauf and J. Cai. Evolution strategies—parallelization and application in engineering optimization. In B. Topping, editor, *Parallel and distributed processing for computational mechanics: systems and tools*, pages 329–349. Saxe-Coburg Publications, Edinburgh, UK, 2000.

V. Torczon. On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, 7:1–25, 1997.

A. Törn and A. Zilinskas. *Global Optimization*. Springer-Verlag New York, Inc., 1989.

P. Tsai, J. Pan, B. Liao, and S. Chu. Enhanced artificial bee colony optimization. *International Journal of Innovative Computing, Information and Control*, 5(12): 1–12, 2011.

L. Tseng and C. Chen. Multiple trajectory search for large scale global optimization. In *Proceeding of the IEEE Congress on Evolutionary Computation,CEC'08*, pages 3052–3059, Piscataway, NJ, 2008. IEEE Press.

P. Tseng. Fortified-descent simplicial search method: A general approach. *SIAM Journal on Optimization*, 10(1):269–288, 1999.

N. Turkkan. Discrete optimization of structures using a floating point genetic algorithm. In *Proceedings of the Annual Conference of the Canadian Society for Civil Engineering*, pages 4–7. Moncton, N.B., Canada, 2003.

M. Črepinšek, S.-H. Liu, and L. Mernik. A note on teaching-learning-based optimization algorithm. *Information Sciences*, 212(0):79–93, 2012.

F. Vanden Berghen and H. Bersini. CONDOR, a new parallel, constrained extension of Powell's UOBYQA algorithm: Experimental results and comparison with the DFO algorithm. *Journal of computational and applied mathematics*, 181(1):157–175, 2005.

M. Wagner, J. Day, D. Jordan, T. Kroeger, and F. Neumann. Evolving pacing strategies for team pursuit track cycling. In *Proceedings of Metaheuristics International Conference, MIC 2011*. MIC Conference Organization, 2011.

J. Wang and Z. Yin. A ranking selection-based particle swarm optimizer for engineering design optimization problems. *Structural and Multidisciplinary Optimization*, 37:131–147, 2008.

L. Wang and L.-p. Li. An effective differential evolution with level comparison for constrained engineering design. *Structural and Multidisciplinary Optimization*, 41(6):947–963, 2010.

D. Whitley, S. Rana, J. Dzubera, and K. E. Mathias. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85(12):245 – 276, 1996.

L. A. Wolsey. *Integer programming*. Wiley, New York, 1998.

A. H. Wright. Genetic algorithms for real parameter optimization. *Foundations of genetic algorithms*, 1:205–218, 1991.

B. Wu and S.-h. Fan. Improved artificial bee colony algorithm with chaos. In Y. Yu et al., editors, *Computer Science for Environmental Engineering and EcoInformatics*, CCIS 158, pages 51–56. Springer Berlin Heidelberg, 2011.

S.-J. Wu and P.-T. Chow. Genetic algorithms for nonlinear mixed discrete-integer optimization problems via meta-genetic parameter optimization. *Engineering Optimization*, 24(2):137–159, 1995.

L. Xu, H. H. Hoos, and K. Leyton-Brown. Hydra: Automatically configuring algorithms for portfolio-based selection. In M. Fox and D. Poole, editors, *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010*. AAAI Press, 2010.

X. Yan, Y. Zhu, and W. Zou. A hybrid artificial bee colony algorithm for numerical function optimization. In *Proceedings of International Conference on Hybrid Intelligent Systems*, pages 127–132. IEEE Press, Piscataway, NJ, 2011.

Z. Yuan, M. Montes de Oca, M. Birattari, and T. Stützle. Modern continuous optimization algorithms for tuning real and integer algorithm parameters. In M. Dorigo et al., editors, *Proceedings of the Seventh International Conference on Swarm Intelligence, ANTS'10*, LNCS 6234, pages 204–215. Springer, Berlin, Germany, 2010.

E. Zahara and Y. Kao. Hybrid Nelder-Mead simplex search and particle swarm optimization for constrained engineering design problems. *Expert Systems with Applications*, 36(2):3880–3886, 2009.

S. A. Zenios. *Financial optimization*. Cambridge University Press, 1996.

J. Zhang and A. C. Sanderson. Jade: adaptive differential evolution with optional external archive. *IEEE Transactions on Evolutionary Computation*, 13(5):945–958, 2009.

Y. Zhong, J. Lin, J. Ning, and X. Lin. Hybrid artificial bee colony algorithm with chemotaxis behavior of bacterial foraging optimization algorithm. In *Proceedings of International Conference on Natural Computation*, pages 1171–1174. IEEE Press, Piscataway, NJ, 2011.

G. Zhu and S. Kwong. Gbest-guided artificial bee colony algorithm for numerical function optimization. *Applied Mathematics and Computation*, 217(7):3166–3173, 2010.

M. Zlochin, M. Birattari, N. Meuleau, and M. Dorigo. Model-based search for combinatorial optimization: A critical survey. *Annals of Operations Research*, 131(1–4):373–395, 2004.

W. Zou, Y. Zhu, H. Chen, and Z. Zhu. Cooperative approaches to artificial bee colony algorithm. In *Proceedings of International Conference on Computer Application and System Modeling*, volume 9, pages 44–48. IEEE Press, Piscataway, NJ, 2010.