JYVÄSKYLÄ STUDIES IN COMPUTING
139

Giovanni Iacca

Memory-Saving Optimization Algorithms for Systems with Limited Hardware

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 139

Giovanni Iacca

Memory-Saving Optimization Algorithms for Systems with Limited Hardware

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen auditoriossa 2
joulukuun 5. päivänä 2011 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in the building Agora, auditorium 2, on December 5, 2011 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2011

Memory-Saving Optimization Algorithms for Systems with Limited Hardware

JYVÄSKYLÄ STUDIES IN COMPUTING 139

Giovanni Iacca

Memory-Saving Optimization Algorithms for Systems with Limited Hardware

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2011

Editors
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä
Pekka Olsbo, Ville Korkiakangas
Publishing Unit, University Library of Jyväskylä

URN:ISBN:978-951-39-4538-1
ISBN 978-951-39-4538-1 (PDF)

ISBN 978-951-39-4537-4 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2011

ABSTRACT

Iacca, Giovanni
Memory-Saving Optimization Algorithms for Systems with Limited Hardware
Jyväskylä: University of Jyväskylä, 2011, 100 p.(+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 139)
ISBN 978-951-39-4537-4 (nid.)
ISBN 978-951-39-4538-1 (PDF)
Finnish summary
Diss.

During the past ten years, there has been a dramatic increase of industrial applications in which some sort of ”intelligence” is plugged into embedded systems, such as mobile robots, remote sensors, etc., to perform specific optimization processes (e.g. training/learning tasks) despite the limited hardware resources. These situations can be addressed in different ways, including compact Evolutionary Algorithms (cEAs) and single-solution (population-less) Optimization Algorithms. cEAs save memory using a statistic representation of a population, instead of storing and process an entire population and all its individuals therein, while population-less Optimization Algorithms save memory using a single solution exploring the search space according to some logic. The main drawback of these algorithms is that, since they do not use a population, they are subject to premature convergence, especially when the dimensionality grows. Nevertheless, it is still possible to design very robust and flexible memory-saving algorithms, which guarantee good results despite a low memory footprint. Among cEAs, compact Differential Evolution (cDE) has proven successful for a broad set of problems. This work presents a few novel cDE-based algorithmic structures, aimed at improving the performance of cDE. As an alternative to cDE-based algorithms, a novel promising single-solution Memetic Computing approach, designed having in mind the *lex parsimoniae* inspired by the Ockham's Razor, is introduced. The main finding of this work is that, for optimization problems plagued by limited hardware, an extremely simple algorithm, if carefully designed, should be preferred to overwhelmingly complicated algorithms having large memory requirements and a high computational overhead.

Keywords: global optimization, compact optimization, single-solution optimization, population-less optimization, evolutionary computing, differential evolution, memetic computing, algorithmic enhancements

Author

Giovanni Iacca
Department of Mathematical Information Technology
University of Jyväskylä
Finland
giovanni.iacca@[jyu.fi][gmail.com]

Supervisors

Adjunct Professor Ferrante Neri
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Doctor Ernesto Mininno
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Professor Tuomo Rossi
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Professor Raino A. E. Mäkinen
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Reviewers

Professor Jarosław Arabas
Faculty of Electronics and Information Technology
Warsaw University of Technology
Poland

Professor Daniela Zaharie
Faculty of Mathematics and Computer Science
West University of Timisoara
Romania

Opponent

Associate Professor Carlos Cotta
Dept. de Lenguajes y Ciencias de la Computación
Universidad de Málaga
Spain

ACKNOWLEDGEMENTS

This dissertation would not have been possible without the help of many people who, in different ways, provided an invaluable support during my Ph.D. studies. Without them, I would have never completed my research and written this thesis.

First and foremost, I want to express my deepest gratitude to Ferrante Neri and Ernesto Mininno, who guided me throughout my research, believed in me, and always allowed me to work in my own way. If now I am a researcher, it is only because of their gentle guidance, their continuous inspiration, and their precious teachings. Most of all, I thank them for the informal atmosphere they created and all the great time we spent together in the last two years. I could not have wished for better or friendlier mentors for my research.

I would like to sincerely thank Prof. Tuomo Rossi and Prof. Raino Mäkinen, who gave me precious advices regarding the Finnish system and helped me in solving many practical problems in all the time of my doctoral studies. I am also grateful to Prof. Tommi Kärkkäinen, who always gave me encouraging words.

I want to thank the University of Jyväskylä, in particular Rector Aino Sallinen, for having believed in me and generously supported my Ph.D. studies. Without Rector's financial support, this dissertation would have been impossible. I especially thank the Department of Mathematical and Information Technology, for having given me incredible opportunities during my studies, and all the people working at the department, for the kind help they gave me in many situations. I would also like to show my gratitude to all the supervisors of the courses I attended during these years, for having greatly enriched my knowledge.

I owe my deepest gratitude to the international coauthors of some of the papers presented in this work, not just for the precious collaboration, but also for being so kind and friendly during my stay in Singapore. I would like to thank all the anonymous reviewers of various journals and conferences, who helped me a lot to improve my work and sometimes provided inspiring ideas. Special thanks go to Fabio, who worked on his Master thesis in our group. It was nice to supervise him and spend funny moments together. Moreover, global thanks go to all my friends in Jyväskylä and worldwide, for their love and support.

Last but not the least, my deepest thanks go to the most important people of my life: my parents Maria and Francesco, my brother Aldo, and my sister Deborah, for always encouraging me and letting me feel their love, and my beloved girlfriend Silvia, for making me feel alive in a way nobody else can. My family is my beacon, which, despite the distance, always sheds its brilliant light on my life and reminds me of my origins. My girlfriend is the beauty of the vast world, she is the place where I belong and all the places I will ever want to see.

"For small creatures such as we the vastness is bearable only through love."
Carl Sagan

LIST OF FIGURES

CONTENTS

LIST OF INCLUDED ARTICLES

PI Ferrante Neri, Giovanni Iacca and Ernesto Mininno. Disturbed Exploitation compact Differential Evolution for Limited Memory Optimization Problems. In *Information Sciences*, volume 181 (2011), issue 12, pages 2469-2487, 2011.

PII Giovanni Iacca, Ernesto Mininno and Ferrante Neri. Composed compact differential evolution. In *Evolutionary Intelligence*, volume 4, number 1, pages 17-29, 2011.

PIII Giovanni Iacca, Ferrante Neri and Ernesto Mininno. Noise Analysis Compact Differential Evolution. In *International Journal of Systems Science*, to appear, 2011.

PIV Giovanni Iacca, Rammohan Mallipeddi, Ernesto Mininno, Ferrante Neri and Ponnuthurai Nagaratnam Suganthan. Global Supervision for Compact Differential Evolution. In *2011 IEEE Symposium on Differential Evolution Proceedings*, pages 1-8, 2011.

PV Giovanni Iacca, Rammohan Mallipeddi, Ernesto Mininno, Ferrante Neri and Ponnuthurai Nagaratnam Suganthan. Super-fit and Population Size Reduction in Compact Differential Evolution. In *2011 IEEE Workshop on Memetic Computing Proceedings*, pages 1-8, 2011.

PVI Giovanni Iacca, Ferrante Neri and Ernesto Mininno. Opposition-Based Learning in Compact Differential Evolution. In *Application of Evolutionary Computation*, volume 6624/2011 of Lecture Notes in Computer Science, pages 264-273, 2011.

PVII Rammohan Mallipeddi, Giovanni Iacca, Ponnuthurai Nagaratnam Suganthan, Ferrante Neri and Ernesto Mininno. Ensemble Strategies in Compact Differential Evolution. In *2011 IEEE Congress on Evolutionary Computation Proceedings*, pages 1972 - 1977, 2011.

PVIII Giovanni Iacca, Ferrante Neri, Ernesto Mininno, Yew-Soon Ong and Meng-Hiot Lim. Ockham's Razor in Memetic Computing: Three Stage Optimal Memetic Exploration. In *Information Sciences*, accepted, 2011.

The author's contribution in the articles listed above was as follows.

The paper which introduced the author to global optimization, and more particularly compact Differential Evolution, was article **PI**, where an unconventional cDE-based Memetic Algorithm, namely Disturbed Exploitation compact Differential Evolution (DEcDE), is presented. The author contributed to the design and the implementation of DEcDE and implemented most of the optimization algorithms used for comparisons. He also executed part of the numerical experiments, and compiled the relative results in figures and tables. Furthermore,

he independently performed the literature review relative to the space robot application, and contributed to the design of the Simulink model of the robot. Finally, he took part in the writing of the Numerical Results chapter in the article.

Articles **PII** and **PIV** form together a study of parallel/distributed variants of cDE, where multiple DEcDE units are used as local searchers to explore the search space from different perspectives. In both the articles, the author wrote the Numerical Results chapters and contributed to the design and the implementation of the proposed algorithms, as well as the benchmark functions and algorithms used for comparisons.

A similar idea is presented in article **PVII**, where a pool of cDE units employing different mutation and crossover strategies with different parameters is introduced. This article was for the author the first opportunity to work with international collaborators, namely R. Mallipeddi Prof. P.N. Suganthan; together with them, he designed and implemented the proposed algorithm and the algorithm used for comparisons, and carried out most of the numerical experiments.

Article **PIII** introduces a Noise Analysis scheme for compact Differential Evolution. In this case the standard cDE is used, instead of DEcDE, to address problems plagued by noise on fitness function. This choice was made to perform a fair comparison between cDE and NAcDE. For this article, the author contributed to the implementation of the proposed algorithm, as well as some of the algorithms used for tests, and to the writing of the Numerical Results chapter.

Articles **PV** and **PVI** investigate the application of some of the most promising ideas recently proposed to improve Differential Evolution, i.e. Super-Fit, Population Size Reduction, and Opposition Based Learning, to the compact Differential Evolution framework. Both articles focus on the standard cDE, and try to improve its performance adding components successfully used in DE, analyzing the contribution of each of these components for a broad set of test functions. In these two articles the author independently designed and implemented both the algorithms proposed, namely SFcDE-PSR and cODE, designed and carried out all the numerical experiments, and wrote most of the contents.

Finally, article **PVIII** represents a relative drift of the author's interest, since it is the first paper in which the compact Differential Evolution is not used. This paper proposes indeed an alternative to cDE-based algorithms, namely a novel single-solution Memetic Computing approach, designed having in mind the Ockham's Razor *lex parsimoniae*. For this article, the author independently carried out the whole algorithmic design, as well as the implementation of the whole set of algorithms and test functions, the relative experiments, and the compilation of results. He also designed, developed and tested the digital filter application. The writing of the article was instead carried out jointly with the coauthors.

1 INTRODUCTION

In our every day life, we always have to make choices. For example, we may need to choose a flight schedule, the shortest path to go back home from work, which car to buy, etc. Regardless the context, all these decisions share a common idea: they are made according to some "optimality" criteria, or, in other words, they are related to some goal we want to reach or necessity we want to satisfy. As a matter of fact, optimization plays a crucial role in basically every aspect of our life, in all the fields of technology and applied sciences, e.g. in engineering, finance, industrial design, bio-informatics, etc. Biological evolution itself can be considered a huge, everlasting optimization process aiming at improving the fitness of living beings, and this is one of the reasons for being so many optimization algorithms inspired by nature, as we will see in the next chapter. Broadly speaking, optimization algorithms are methods that aim at detecting the optimal solution of a given problem within a (continuous or discrete[1]) set of possible candidate solutions. More formally, given a *fitness* (*objective*, or *cost*) *function* in the form:

$$f : D \in \mathbb{R}^n \rightarrow \mathbb{R} \tag{1}$$

a (minimization) optimization problem can be defined as follows:

$$\text{find } x^* : f(x^*) = \min_{x \in D} \{f(x)\} . \tag{2}$$

where D is called *decision* (*design*, or *search*) *space*, $x \in D$ is a *solution* (also called *individual* or *point*), whose i-th component, $i = 1, \ldots, n$ (where n is the problem dimension) is called *decision variable*, and x^* denotes the (*global*) *optimum*. It must

[1] Intuitively, a set is considered to be discrete if it is composed of isolated elements, whereas it is considered to be continuous (dense) if it is composed of infinite and contiguous elements without "holes". More formally, a set $S \subseteq \mathbb{R}$ is said to be *dense* if:

$$\forall x_1, x_2 \in S \colon \exists x_3 \colon x_1 \leqslant x_3 \leqslant x_2.$$

If the property above is not satisfied for all the points, the set is said to be *discrete*. It must be remarked that, although on a computer all the sets are in principle discrete, if the distance between each pair of consecutive points in a set is not bigger than the machine precision ϵ, the set can still be considered dense, see Neri et al. (2011).

be remarked that, in the most general formulation, an optimization problem may have a more complicated decision space (i.e. bounded by non-linear constraints) and multiple conflicting objectives. In the remainder of this thesis, only single-objective unconstrained continuous optimization problems will be considered.

This work focuses on a specific class of optimization applications, that is applications plagued by limited hardware. The main reason to investigate this problem, is that, despite the every day increasing availability of powerful computational devices, nowadays there are still several applications which make use of "intelligent" systems characterized by severely limited hardware. These systems are often required to perform some specific optimization operations, such as an online training procedure, or the solution of a more general optimization problem. In the first case, the system is required to "learn" what is the best "behaviour" to improve some performance metrics; in the latter, the system is supposed to find the optimal solution within a set of candidate solutions of a given problem, e.g. find the shortest path among different possible options. In a nutshell, the main research problem which this work tries to address could be summarized by the following question:

is it possible to perform an optimization process on board of a system with limited hardware?

Typical examples of limited-hardware applications are: 1) home automation systems (such as cleaning devices, cooking robots, lawnmowers etc.), which must perform in an "intelligent" way some relatively simple task, still being affordable for low budget consumers; and 2) real-time control systems, where the optimization must be carried out on a micro-controller as quickly as possible, in order to leave a larger time slot for real-time communication with sensors and actuators.

In all these cases, costs and physical space requirements impose the employment of embedded systems characterized by very simple hardware structures. In order to perform optimization tasks on such a hardware, specific algorithms must be designed. Due to the hardware limitations, traditional population-based optimization methods, i.e. algorithms relying on the concurrent evaluation of multiple solutions, as well as algorithms employing complex learning structures, would be indeed unacceptable. On the contrary, a "memory-saving" approach must be applied, i.e. a method requiring a very limited amount of run-time memory and computational resources.

This thesis is organized as follows. The present chapter introduces some classical methods used for local and global continuous optimization. Chapter 2 describes the most popular population-based meta-heuristics used in global continuous optimization. Chapter 3 focuses more deeply on a specific meta-heuristic, namely the Differential Evolution. Chapter 4 introduces the class of compact algorithms, which simulate the behaviour of population-based algorithms by employing, instead of a population of solutions, its probabilistic representation. The two major algorithms belonging to this class, namely compact Genetic Algorithm (cGA) and compact Differential Evolution (cDE) are also described. Finally, chapter 5 presents the main contributions of this work. A few novel cDE-based struc-

tures are introduced, aimed at improving the performance of the standard cDE. As an alternative to cDE-based algorithms, a novel single-solution Memetic Computing approach, designed having in mind the *lex parsimoniae* inspired by the Ockham's Razor, is also presented. As we will see, the main finding of this work is that, for optimization problems plagued by limited hardware, an extremely simple algorithm, if carefully designed, is to be preferred to overwhelmingly complicated algorithms characterized by a large memory footprint and a high computational overhead.

1.1 Local Continuous Optimization

If the fitness function is Cauchy-continuous, for any given point, its closest points are expected to have a similar performance with respect to that point. Equivalently, all the points in the *neighborhood* of a given point, i.e. those points characterized by distance ϵ from it, have a similar fitness value. This concept is fundamental in continuous optimization because, unlike the discrete optimization case, it allows us to introduce the idea of "small" search movements and search directions. In other words, it implies the concept of *local search*. Local search methods can be classified as "derivative-based" (see section 1.1.1) and "derivative-free" (see section 1.1.2). According to the order of the derivatives used for exploring the search space, derivative-based methods can be classified as follows: (1) *first-order methods*, which rely on the objective function and to its (approximated) gradient; and (2) *second-order methods*, which use the objective function, its (approximated) gradient, and its (approximated) Hessian. On the other hand, derivative-free (or *zeroth-order*) methods base their search solely on ordinal relations between objective function values, without any model of higher order derivatives of the objective function. For the sake of completeness, it should be remarked that some authors, see e.g. Neri et al. (2011), call a method "derivative-free" if it uses the zeroth-order information to approximate higher order derivatives of the objective function. For example, the approximation of the partial derivative of the fitness function f within the neighborhood of radius ϵ of a given point, with respect to the decision variable $x[i]$, can be defined as follows:

$$\frac{\partial f}{\partial x[i]} \approx \frac{f\left(x[i]+\epsilon\right)-f\left(x[i]\right)}{\epsilon}. \tag{3}$$

Whether approximated or not, the information derived from the gradient (or higher derivatives) can be obviously exploited, within an optimizer, to select the most promising neighbor of a given starting point, and thus to identify a promising search direction, as we will see in the next section.

1.1.1 Derivative-based Local Search

If the objective function f has an analytical expression, and it is unimodal, continuous and twice-differentiable, it is possible to find its minimum using an analytic

method, based on the second order Taylor expansion of f around a given point x_0. More specifically, we can write:

$$f(x) \simeq h(x) = f(x_0) + \nabla f(x_0) \cdot (x - x_0) + \frac{1}{2}(x - x_0)^T \cdot H(f)(x_0) \cdot (x - x_0)$$

where $\nabla f(x)$ and $H(f)(x)$ are, respectively, the gradient and the Hessian of f. Since the minimum x^* of f is a stationary point where $\nabla h(x) = 0$, it follows that:

$$x^* = d(x_0) + x_0 \tag{4}$$

where $d(x_0) = -H^{-1}(f)(x_0)\nabla f(x_0)$ is called Newton direction, or Newton step at $x = x_0$. Equation (4) provides a second-order iterative method, called Newton's method, to find the minimum of a function, given an initial guess x_0. The pseudo-code of the Newton's method is shown in Fig. 1, where α is a step size control parameter whose choice heavily depends on the objective function. It is worthwhile to notice that if the inverse of the Hessian is replaced with the identity matrix, the search direction can be calculated as $d(x_0) = -\nabla f(x_0)$. Due to the use of a first-order approximation, however, this latter method, called *steepest descent* (or *gradient descent*), converges more slowly than the Newton's method.

Although extremely intuitive, the Newton's method has some serious drawbacks, mostly related to its high computational cost and to numerical instability (because of the inversion of the Hessian). Alternative Newton methods have been proposed to overcome these issues, which for example scale the Hessian or use iterative methods (based on factorization) to compute its inverse. Quasi-Newton and conjugate gradient methods have also been proposed: the first methods rely on an approximation of the Hessian (or its inverse), dynamically build up from changes in the gradient; the latter perform a line search along conjugate directions. But even if these methods have good convergence properties on quadratic, twice-differentiable, unimodal functions, they are not sufficiently robust to handle more complex functions (e.g. highly multivariate or multi-modal).

counter $k = 0$
generate an initial guess x_0
while *stop condition* **do**
  $x_{k+1} = -\alpha \cdot \left(H^{-1}(f)(x_k)\nabla f(x_k)\right) + x_k$
  $k = k + 1$
end

FIGURE 1 Newton's method pseudo-code

1.1.2 Derivative-free Local Search

In practical applications, the operation of derivation is not always possible because, for example, the objective function is not differentiable within the entire decision space, is affected by noise, or is not even available in an explicit analytical form (being, for instance, an experiment measurement). In situations where the objective function is the output of a simulation or the execution of a

program whose code is available, Automatic Differentiation (AD), see Griewank and Walther (2008), can still be applied to compute accurate derivatives. AD has been successfully applied in many engineering applications, see e.g. Toivanen et al. (2009) and Toivanen and Mäkinen (2011). In more general cases, where derivatives are not available at all, derivative-free optimization methods must be used. Also known as *direct search* methods, they consist in generating, according to some logic, a solution x and then testing its fitness by computing $f(x)$. This process is then repeated until a satisfactory solution is found, or until a predefined budget of fitness evaluations is exhausted. It is worth noting that even though these algorithms do not require any knowledge about the fitness function and its derivative, they still make use of some form of gradient given by the difference of the fitness between two neighbor points, similarly to eq. (3). If the direction suggested by this "gradient" leads to an improvement, these algorithms make optimistic attempts to follow it in the hope to find yet a better point. Except the Rosenbrock Algorithm, the methods here described are considered *heuristics*, since they can converge to non-stationary points on some specific problems. In addition, it must be remarked that these methods are *deterministic*, in the sense that, starting from a given point, their logic employs a predictable sequence of search moves, without applying any kind of stochastic perturbation or any form of randomization. To some extent, *stochastic* algorithms can be considered instead inherently global optimizers, due to their implicit ability to escape local minima (see the next section for further details). Nevertheless, there exist stochastic algorithms that can be used both as local or global optimizers, see e.g. the Simulated Annealing, Kirkpatrick et al. (1983), or the Tabu Search, Glover (1989a,b).

Hooke-Jeeves Method

The main idea of the Hooke-Jeeves algorithm, also called Pattern Search, see Hooke and Jeeves (1961), is to explore, along each of the axes of the search space, a neighborhood of radius h of a given initial solution x_0, trying to find whether a step of size h towards the positive or the negative direction is leading to a better fitness (if no improvement is found after exploring both directions, the original position of x_0 on that axis is retained). Once every axis has been probed, a new point x_1, obtained applying to x_0 an offset of $\pm h$ along the relevant axes, is evaluated. If $f(x_1) \geq f(x_0)$, the step size is decreased (e.g. $h/2$), and the new neighborhood of x_0 is explored. Otherwise, a new base point x_2 is chosen by taking one step further from x_1 in the direction defined by x_0 and x_1, i.e. $x_2 = x_1 + (x_1 - x_0)$, optimistically assuming that the direction is leading towards a better fitness. The algorithm is then applied again on x_2.

Nelder-Mead Method

While the Hooke-Jeeves algorithm relies on a single point and the systematic exploration of its neighborhood, the Nelder-Mead algorithm, or downhill simplex, see Nelder and Mead (1965), makes use of a set of $n + 1$ points, $x_0, x_1, \ldots, x_n$,

forming an n-dimensional polytope, or simplex, in D. At each iteration of the algorithm, these points are sorted according to their fitness, so that x_0 has the best fitness and x_n presents the worst fitness. The procedure then consists in constructing a candidate replacement point x_r for x_n by reflection of x_n in respect with the barycenter x_m of the other $x_0, x_1, \ldots, x_{n-1}$ points. Depending on the performance of x_r compared to x_0 and x_{n-1}, an extension point may be created in an optimistic attempt to explore further in the same direction, or on the contrary a contraction point may be computed closer to x_m. If none of the above attempts lead to a better solution, the simplex is contracted around its best point in order to reduce the exploration range in the next iteration of the algorithm.

Rosenbrock Method

The Rosenbrock Algorithm, see Rosenbrock (1960), is a classic deterministic local search which, under specific conditions, has been proved to always converge to a local optima (see Bazaraa et al. (2006)). Like the Hooke-Jeeves method, at the beginning this method probes each of the n base directions, with a step size h. In case of success, the step size is increased, otherwise it is decreased and the opposite direction is tried. Once a success has been found and exploited in each base direction, the coordinate system is rotated towards the approximated gradient, the step size is reinitialized and the procedure is repeated, until a stop criterion is met, using the rotated coordinate system. The main flaw of this algorithm is related to the creation of the new rotated coordinate system: this operation, which is usually performed by means of orthogonalization procedures, is indeed computational expensive, and in some cases may even become numerically instable.

1.2 Global Continuous Optimization

The problem of the methods described so far is that, if the objective function is multi-modal, i.e. contains multiple local minima, they tend to detect the *local minimum* closest to the starting point. However, the goal in optimization is to detect the *global optimum*, that is the solution exhibiting the lowest function value over the entire search space. In principle, in order to detect it, we should find all the null gradient points and then select the global optimum. On the other hand, as we have seen before, this operation is often impractical, if not impossible. In order to guarantee an extensive search, and in many cases avoid that the search gets stuck within local optima, an efficient *global optimizer* should not be based, therefore, only on gradient information, but also on direct fitness comparisons among solutions, regardless their position within the decision space. Here we describe the two simplest, but rather inefficient, global search methods, one purely deterministic and one completely stochastic. In the next chapter we will see how global optimization can be performed in a more efficient way, using meta-heuristics.

1.2.1 Brute-force Method

The simplest deterministic global optimization algorithm one may think of is the *brute-force method*, also known as enumeration. It simply consists in generating a finite number of points in the search space and evaluating all of them, keeping the best one as the solution to the problem. One way to generate these points is, for example, to construct a grid of k^n points covering the whole search space, sampling k equally spaced points in each interval $D_i = [x_i^L, x_i^U]$, $i = 1, 2, \ldots, n$. The main flaw in the brute-force search is that most of the tested points are far away from the optimum; moreover, this method highly depends on the sampling step-size chosen, as it affects the trade-off between complexity (number of solutions to evaluate) and performance (the probability of finding the global optimum).

1.2.2 Stochastic Global Search

The simplest stochastic way to perform the global optimal search of a generic function is the progressive perturbation of one or more solutions in order to improve upon their performance. The search can be performed by various search rules, for example randomly generating a new solution within the decision space or adding a randomized perturbation vector to a trial solution, see e.g. the Random Walk proposed in Gross and Harris (1985). These algorithms, called Stochastic Global Search or simply Stochastic Search methods, Spall (2003), can be considered the ancestors of all the modern computational intelligence optimization algorithms. As we will see in the next chapter, basically all the modern nature-inspired global optimization algorithms can be considered in the end stochastic search algorithms which differ one from another on the mechanism used for generating the trial solutions and/or the strategy for selecting them.

2 META-HEURISTICS

When hypotheses on the optimization problem cannot be made, a general purpose global optimization algorithm must be implemented for solving it, or at least detecting some sufficiently good solutions. Algorithms of this kind are usually referred as *meta-heuristics*, from the ancient Greek words μετα and ευρισκω, i.e. "*I search beyond*" or "*beyond the search*", as they apply a heuristic method in a controlled way to guide the search into obtaining a (sub) optimal solution.

A huge variety of meta-heuristics has been developed during the last years, many of which taking inspiration from nature, e.g. evolutionary principles, physical phenomena, animal behaviour, diffusion of ideas etc. These nature-inspired methods, also known as Computational Intelligence Optimization methods, can be classified in many different ways, according to their properties and their functioning principles. Although an exhaustive survey of all the CI optimization methods is out of the scope of this work, in this chapter we briefly review three major classes of population-based meta-heuristics, namely Evolutionary Algorithms (EAs), Swarm Intelligence (SI) and Memetic Computing (MC). As it will be more clear in the next chapters, most of the concepts described here can be applied, with limited changes, in memory-saving optimization algorithms.

2.1 Evolutionary Algorithms

In population-based algorithms, the search is carried out exploring multiple solutions concurrently. As a metaphor, Evolutionary Algorithms (EAs) consider these solutions as a population of individuals which breed with each other in order to improve their adaptation (or, in other words, their fitness) to environment. Using the same mechanisms that subtend biological evolution, EAs are characterized by four phases, namely: 1) parent selection, 2) crossover, 3) mutation, 4) survivor selection. The pseudo-code of a generic EA is given in Fig. 2, where N is the number of individuals in the population and pop_t is the population at the t-th generation. Here we describe the basic definitions of some of the most popular optimization

EAs in use nowadays, i.e. Genetic Algorithms, Evolutionary Programming and Evolution Strategies.

counter $t = 0$
// ** initial population pop_0 **
generate and evaluate N random individuals to create pop_0
while *budget condition* **do**
  // ** recombination and mutation **
  select parents for recombination
  recombine parents to create offsprings
  mutate and evaluate offsprings
  // ** update population pop_{t+1} **
  replace some individuals of pop_t with the offsprings to create pop_{t+1}
  $t = t + 1$
end

FIGURE 2 EA pseudo-code

2.1.1 Genetic Algorithms

The very first version of Genetic Algorithms (GAs), see Holland (1975) and Goldberg (1989), was initially developed for solving combinatorial problems. Nevertheless, the original framework has been successfully extended to mixed and continuous optimization and nowadays, due to a plethora of algorithmic enhancements and applications in many heterogeneous fields, GAs are undoubtedly the most popular Evolutionary Algorithms. The inspiring principles of GAs are to be found in the biological mechanisms behind sexual reproduction, where two parent individuals recombine their genomes in order to produce offsprings. In the original implementation of the GA, the genome of an individual is represented by a string of bits, i.e. 0-1 values, thus requiring a binary encoding of real-valued parameters (e.g. by means of Gray codes). In most of the modern implementations instead, GAs use a more natural real-valued (or in some cases integer) representation of individuals. In other words, each individual is considered a vector in $\mathbb{R}^n$, where n is the problem dimension:

$$x = (x[1], \ldots, x[n]) .$$

Parent selection can be performed according to different logics. Two examples of selection methods are (1) the proportionate selection, where each parent is assigned a selection probability proportional to its fitness (so that the best individuals have higher chance to be selected); (2) the k-tournament selection, where each parent is the best solution out of a sample of k randomly selected individuals.

Recombination usually takes place applying on the two parents a crossover function, which concatenates sequences of genes (i.e. decision variables) from one parent with other sequences from the other parent. There exist many different crossover schemes, among which the most used are the single-point and multi-point crossovers. More complex crossover schemes are also possible, such as mask crossover or partially mapped crossover, see Eiben and Smith (2003).

After recombination, the offsprings undergo a mutation process, which consists in perturbing, with a very low probability, one or more of their components. It is important to observe that in GAs the mutation operator has actually an exploratory function: if offsprings were created only by means of crossover, the "reachable" search space would be limited only to those solutions that can be generated recombining the initial population. In other words, there would be parts of the search space totally unreachable for the algorithm. On the other hand, the introduction of random mutations provide instead a larger exploration ability, allowing the creation of points outside this limited region of the search space.

Once the mutation has been performed, finally survival selection occurs: in GAs, survival is usually performed according to the generational strategy, i.e. the parents are completely replaced by their offsprings.

2.1.2 Evolutionary Programming

In Evolutionary Programming (EP), see Fogel et al. (1966), each individual is a real-valued vector composed of its candidate solution representation x and a set of self-adaptive parameters σ:

$$(x, \sigma) = (x[1], \ldots, x[n], \sigma[1], \ldots, \sigma[n]) .$$

Trying to emulate the way new features appear in living beings through mutation of their genome, EP does not employ a recombination scheme, but relies solely on mutation. More specifically, at each iteration of the algorithm, for each individual (x, σ), an offspring is generated according to the following formulas:

$$\begin{cases} \sigma[i] = \sigma[i](1 + \alpha \cdot \mathcal{N}(0,1)) \\ x[i] = x[i] + \sigma[i] \cdot \mathcal{N}(0,1) \end{cases}$$

where the index i refers to the index of the variable, $i = 1, \ldots, n$ (n being the dimension of the problem), $\mathcal{N}(0,1)$ is a Gaussian random variable with mean 0 and standard deviation 1, and α is a control parameter used to scale the perturbation. After mutation, the offsprings are evaluated, and merged with the parents. Finally, survivor selection is performed in such a way that each individual is compared against a set of randomly selected individuals, and the individuals which have won the highest number of comparisons survive.

2.1.3 Evolution Strategies

Evolution Strategy (ES), see Schwefel (1965) and Rechenberg (1971), employs a self-adapting scheme similar to EP. Like in EP, each individual is a real-valued vector composed of its candidate solution representation x and a set of parameters σ, and the evolution is essentially driven by mutation; on the other hand, recombination is not simply dismissed as in EP, but still plays a minor role. For the generic i-th variable, the general mutation rule is defined by:

$$\begin{cases} \sigma[i] = \sigma[i] e^{\mathcal{N}(0,\tau') + \mathcal{N}_i(0,\tau)} \\ x[i] = N\left(x[i], \sigma[i]\right) \end{cases}$$

where $\mathcal{N}(0,\tau')$ and $\mathcal{N}_i(0,\tau)$ are two different normally distributed random numbers with mean 0 and standard deviation τ' and τ, respectively. The notation $\mathcal{N}_i(0,\tau)$ denotes a different random number for each variable, whereas $\mathcal{N}(0,\tau')$ is a common –solution-wise– random number. τ and τ' are called local and global learning rate, respectively. It must be remarked that several alternative rules have been proposed in literature for the update of σ, see e.g. the schemes proposed in Rechenberg (1971) and Eiben and Smith (2003).

In general, recombination occurs between two randomly chosen individuals, and it can be either discrete or intermediate. In the first case, the offspring takes some of the genes from each of the two parents, in the latter it is generated for example calculating a randomly weighted average of the corresponding genes of the two parents. A multitude of alternative recombination strategies among pairs or small groups of solutions have also been proposed in literature, see e.g. the BLX$-\alpha$ crossover described in Eshelman (1990) and Herrera et al. (2003) or its variant introduced in Lozano et al. (2004). The advantages of one strategy with respect to another depend, in general, on the specific problem.

Finally, survivor selection can be performed either in the genetic algorithm fashion by replacing the whole parent population with the best members of the offspring population ((μ,λ) or "comma" strategy) or by merging parent and offspring populations and selecting the wanted number of individuals on the basis of their fitness values ($(\mu+\lambda)$ or "plus" strategy).

A powerful Evolution Strategy approach that deserves being mentioned is the Covariance Matrix Adaptation Evolution Strategy (CMA-ES) Hansen and Ostermeier (2001). In CMA-ES, solutions are generated via a multivariate normal distribution whose mean and covariance matrix are adaptively updated from a subset of promising solutions. CMA-ES has a solid theoretical background and several desirable properties such as invariance to several transformation of the objective function and a relatively low number of parameters.

2.2 Swarm Intelligence

Unlike EAs, Swarm Intelligence (SI) algorithms consider the solutions of as members of a group of particles or animals (e.g. a flock or a swarm), where each individual has a "limited" intelligence which contributes to create a form of collective, or social, intelligence (e.g. by means of simple behavioral rules, an individual may tend to follow its neighbors or a swarm leader). A peculiar characteristic of SI algorithms is the so called "one-to-one" spawning (or replacement) logic, that is each individual in the swarm generates a new individual, according to some perturbation mechanism, and a replacement occurs only if the new individual outperforms the old one. The pseudo-code of a generic SI algorithm is shown in Fig. 3, where pop_t is the swarm at the t-th iteration and N is the number of individuals in it. Here we briefly introduce two well-known Swarm Intelligence algorithms, namely Particle Swarm and Bacterial Foraging Optimization.

counter $t = 0$
// ** initial population pop_0 **
generate and evaluate N random individuals to create pop_0
while *budget condition* **do**
  for $i = 1, \ldots, N$ **do**
    // ** one-to-one spawning **
    perturb individual $pop_t[i]$ and evaluate it
    compare the fitnesses of $pop_t[i]$ and the perturbed individual
    // ** update population pop_{t+1} **
    select the winning individual and copy it into $pop_{t+1}[i]$
  end
  $t = t + 1$
end

FIGURE 3 SI Algorithm pseudo-code

2.2.1 Particle Swarm Optimization

The metaphor employed in Particle Swarm Optimization (PSO), see Kennedy and Eberhart (1995), is that particles in a swarm make use of their "personal" and "social" experience in order to explore a decision space and detect solutions with a high performance. More specifically, a swarm of candidate solutions is randomly sampled within the decision space. Subsequently, the fitness value of each candidate solution is computed and the solutions are ranked on the basis of their performance. The solution associated to the best fitness value overall detected is named global best x^{gb}. At the first iteration, for each solution x_i a corresponding local best solution x_i^{lb} is initialized with its position in the search space. In the next iterations, for each solution x_i, x_i^{lb} is updated with the most successful position found so far by that solution. Each particle x_i explores the search space according to the following rule:

$$\begin{cases} v_i = \omega v_i + \alpha_1 (x_i^{lb} - x_i) + \alpha_2 (x^{gb} - x_i) \\ x_i = x_i + v_i \end{cases}$$

where v_i represents a velocity (perturbation) vector, ω is the so-called inertia parameter (the higher this parameter, the longer it takes the particle to change direction), and α_1, α_2 are two parameters that control the attraction of each particle towards the best-known local/global solutions. For each particle, and at each step, the values of these two parameters are randomly generated (typically they are extracted from a uniform distribution $\mathcal{U}(0, 1)$, with 0 excluded and 1 included). The metaphoric meaning of these formulas is that each particle performs a move in the search space combining its "personal" experience (i.e. its local best x_i^{lb}) with a form of "social" knowledge given by a partial imitation of the most promising individual in the swarm. These two contributions are randomly weighted in order to keep a high level of diversity and prevent premature convergence.

The fitness value of the newly generated x_i is calculated and a replacement of local and/or global best occurs if needed. At the end of the optimization process, the final global best detected is the estimate of the global optimum returned

by the particle swarm algorithm. A pseudo-code showing the main features of the basic PSO is given in Fig. 4. It is worth to note that many versions and variants of PSO have been proposed in literature in order to improve its performance, see e.g. the linearly variable weight factor proposed in Shi and Eberhart (1998) and the constriction factor introduced in Clerc and Kennedy (2002).

```
generate N_p particles and N_p velocities pseudo-randomly
copy the particle swarm into the set of local bests: ∀i, x_{i-lb} = x_i
while budget condition do
    for i = 1 : N_p do
        compute f(x_i)
    end
    for i = 1 : N_p do
        // ** Velocity Update **
        generate a vector of random numbers U(0,1)
        v_i = ωv_i + U(0,1)(x_i^{lb} − x_i) + U(0,1)(x^{gb} − x_i)
        // ** Position Update **
        x_i = x_i + v_i
        // ** Survivor Selection **
        if f(x_i) ⩽ f(x_{i-lb}) then
            x_i^{lb} = x_i
            if f(x_i) ⩽ f(x^{gb}) then
                x^{gb} = x_i
            end
        end
    end
end
```

FIGURE 4 PSO pseudo-code

2.2.2 Bacterial Foraging Optimization

Bacterial Foraging Optimization (BFO), see Passino (2002) and Das et al. (2009b), is inspired by the foraging behavior of the E. coli bacteria within some environment with a non-uniform distribution of nutrients. The basic idea is to explore the search space performing tentative moves similar to the swim foraging pattern (called "chemotaxis") observed in motile bacteria. Basically, bacterial chemotaxis is a complex combination of two types of moves, namely tumbling (i.e. changes of direction) and swimming (i.e. moves along a successful direction), which respectively enable the bacteria to search for nutrients in random directions and rapidly approach higher concentrations of nutrients. In other words, the alternation between "swims" and "tumbles" guarantees a balance between exploitation and exploration of the search space. The classical BFO consists of three phases, namely: 1) chemotaxis, 2) reproduction, and 3) dispersal. During chemotaxis, the

movement of the i-th bacterium is modeled as:

$$x_i = x_i + C_i \frac{\Delta_i}{\sqrt{\Delta_i^T \Delta}}$$

where Δ_i is the direction vector of the chemotactic step, and C_i is a parameter which controls the step size. In tumbles, Δ_i is a random vector whose elements are uniformly distributed in $[-1, 1]$; in swims instead, Δ_i is the same as the last chemotactic step, thus allowing the bacterium to exploit a promising direction.

To mimic the asexual reproduction of E. coli, at each iteration BFO sorts all the bacteria according to their fitness and selects the best half of the swarm. Each survivor is then splitted into two replicas, thus keeping the swarm size constant. Finally, in order to prevent premature convergence and keep a high diversity rate, after a fixed number of chemotaxis/reproduction steps a few bacteria are chosen, with some probability, for being replaced with new random individuals.

Like other SI algorithms, BFO has been successfully applied to many practical problems. However, it must be remarked that compared to other metaheuristics BFO possesses a poor convergence behavior, especially over high dimensional complex optimization problems. To overcome these issues, some adaptive and self-adaptive variants of the original BFO have been proposed, see e.g Chen et al. (2008, 2011) and Dasgupta et al. (2009, 2010).

2.3 Memetic Algorithms and Memetic Computing

Following the definition given in Hart et al. (2004), a Memetic Algorithm (MA) is a hybrid meta-heuristic composed of an evolutionary framework, which acts as global searcher, and one or more local search components activated within its generation cycle (see, for example, the local search methods described in chapter 1). In a broader sense, MAs can be seen as the founding subset of Memetic Computing (MC), a broad group of techniques that use the notion of *memes* as *"units of information encoded in computational representations for the purpose of problem solving"*, Ong et al. (2010). Generally speaking, MC methods are dynamic computational structures composed of multiple interacting memes coordinated according to some logic. Due to their robustness and versatility, these methods have proven successful in many applications and nowadays MC is one of the most active area of research in Computational Intelligence.

A crucial issue in MAs (and MC in general) is the coordination among memes, that is the way (when and how) each meme is activated. For example, a rather simple strategy to control the activation of local search in MAs, called "partial Lamarckianism", see Houck et al. (1997), consists in randomly applying the local search with some probability. Another problem is the "intensity" of the local search, i.e. the exploitation pressure applied on a given solution to be locally improved. A possible scheme, proposed in Molina (2005), consists in classifying the solutions according to their fitness and associate a different set of local-search pa-

rameters to each of set of solutions. Related to these two issues, the most critical point in MAs is however the balance between exploitation and exploration, see Lozano and García-Martínez (2010). Many strategies have been proposed to keep a proper balance between these two conflicting pressures, for example using multiple subpopulations, Mühlenbein et al. (1991), clustering solutions, Seront and Bersini (2000) (to detect clusters of neighbor solutions upon which it may not be fruitful to apply local search), or even "interweaving" the local search within the population-based engine, see e.g. the so-called hill climbing crossover described in Jones (1995) and applied in MAs in Lozano et al. (2004).

3 DIFFERENTIAL EVOLUTION

Amongst meta-heuristics, Differential Evolution (DE), see Storn and Price (1995), deserves a separate mention, since it shares some properties of Evolutionary Algorithms (EAs) and some others of Swarm Intelligence (SI) Algorithms. Like most popular EAs, DE is a population-based algorithm, but unlike other EAs, it generates offspring by perturbing each solution in the population with a scaled difference of two randomly selected individuals, instead of simply recombining two parents. Still, DE applies a form of crossover similar to some recombination schemes used in EAs. In addition, DE employs the one-to-one logic spawning typical of SI, which allows replacement of an individual only if the offspring outperforms its corresponding parent.

Thanks to, on one hand, its simplicity and ease of implementation, and on the other hand, reliability and high performance, DE has proven to be a robust and versatile function optimizer, thus becoming very popular among computer scientists and engineers. A broad review of applications of DE is presented in Price et al. (2005) and Plagianakos et al. (2008). However, even if standard DE has a great potential, many modifications of the original framework have been proposed in literature in order to improve its performance. Following the taxonomy introduced in Neri and Tirronen (2010), we can subdivide the modified versions of DE into two classes:

1. DE integrating extra components. This class includes those algorithms composed of a DE framework and one or more additional algorithmic components, e.g. local search methods, an alternative mutation scheme, etc.

2. Modified structures of DE. This class includes those algorithms which make substantial modifications within the DE structure, i.e. in the search logic, the selection mechanism, etc.

In this chapter we will briefly review the basic DE framework, and some of the most recent DE-based algorithmic structures belonging to these two classes. Most of the contributions of this work are related to the application of some of the ideas behind these DE structures, within the context of a compact Differential Evolution (cDE) framework (see chapters 4 and 5).

3.1 Standard Differential Evolution

According to the original definition given in Storn and Price (1995), DE consists of the following steps, see Fig. 5. An initial sampling of N_p individuals is performed pseudo-randomly with a uniform distribution function within the decision space D. At each generation, for each individual x_i, three mutually distinct individuals x_r, x_s and x_t are pseudo-randomly extracted from the population. Then, a provisional offspring x'_{off} is generated by mutation as:

$$x'_{off} = x_t + F(x_r - x_s) \tag{5}$$

where $F \in [0, 1+[$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated. With $F \in [0, 1+[$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1, see Price et al. (2005). While there is no theoretical upper limit for F, effective values are rarely greater than 1.0. The mutation scheme shown in eq. (5) is also known as DE/rand/1. Other variants of the mutation rule have been subsequently proposed in literature, see Qin and Suganthan (2005):

- DE/best/1: $x'_{off} = x_{best} + F\,(x_s - x_t)$
- DE/cur-to-best/1: $x'_{off} = x_i + F\,(x_{best} - x_i) + F\,(x_s - x_t)$
- DE/best/2: $x'_{off} = x_{best} + F\,(x_s - x_t) + F\,(x_u - x_v)$
- DE/rand/2: $x'_{off} = x_r + F\,(x_s - x_t) + F\,(x_u - x_v)$
- DE/rand-to-best/2: $x'_{off} = x_r + F\,(x_{best} - x_i) + F\,(x_r - x_s) + F\,(x_u - x_v)$

where x_{best} is the solution with the best performance among individuals of the population, x_u and x_v are two additional pseudo-randomly selected individuals. It is worthwhile to mention the rotation invariant mutation shown in Price (1999):

- DE/current-to-rand/1 $x_{off} = x_i + K\,(x_t - x_i) + F'\,(x_r - x_s)$

where K is is the combination coefficient, which as suggested in Price (1999) should be chosen with a uniform random distribution from $[0, 1]$ and $F' = K \cdot F$. For this special mutation the mutated solution does not undergo the crossover operation (since it already contains the crossover), described below.

Recently, in Price et al. (2005), a new mutation strategy has been defined. This strategy, namely DE/rand/1/either-or, consists of the following:

$$x'_{off} = \begin{cases} x_t + F\,(x_r - x_s) & \text{if } rand\,(0,1) < p_F \\ x_t + K\,(x_r + x_s - 2x_t) & \text{otherwise} \end{cases} \tag{6}$$

where for a given value of F, the parameter K is set equal to $0.5\,(F + 1)$. When

```
generate N_p individuals of the initial population pseudo-randomly
while budget condition do
    for k = 1 : N_p do
        compute f(x_k)
    end
    for k = 1 : N_p do
        // ** Mutation **
        select three individuals x_r, x_s, and x_t
        compute x'_off = x_t + F(x_r − x_s)
        // ** Crossover **
        x_off = x'_off
        for i = 1 : n do
            if rand(0,1) < Cr then
                x_off[i] = x_k[i]
            end
        end
        // ** Survivor Selection **
        if f(x_off) ⩽ f(x_k) then
            save index for replacement x_k = x_off
        end
    end
    perform replacements
end
```

FIGURE 5 DE/rand/1/bin pseudo-code

the provisional offspring has been generated by mutation, each gene of the individual x'_{off} is exchanged with the corresponding gene of x_i with a uniform probability and the final offspring x_{off} is generated:

$$x_{off}[j] = \begin{cases} x_i[j] & \text{if } rand\,(0,1) < CR \\ x_{off}[j]' & \text{otherwise} \end{cases} \tag{7}$$

where $rand\,(0,1)$ is a random number between 0 and 1; j is the index of the gene under examination. This crossover strategy is well-known as binomial crossover and indicated as "bin". For the sake of completeness, we must mention that there exist a few other crossover strategies, for example the "exponential" strategy, see Price et al. (2005). In this case, a design variable of the provisional offspring $x'_{off}[i]$ is randomly selected and copied into the i-th design variable of a copy of the parent x. This guarantees that parent and offspring have different genotypes. Subsequently, a set of random numbers between 0 and 1 are generated. As long as $rand\,(0,1) \leq Cr$, where the crossover rate Cr is a predetermined parameter, the design variables from the provisional offspring (mutant) are copied into the corresponding positions of the final offspring. The first time that $rand\,(0,1) > Cr$, the copy process is interrupted. Thus, all the remaining design variables of the offspring are copied from the parent. For the sake of clarity, the pseudo-code of the exponential crossover is shown in Algorithm 6. The resulting offspring x_{off}

$x_{off} = x$
generate $i = round\,(n \cdot rand\,(0,1))$
$x_{off}[i] = x'_{off}[i]$
while $rand\,(0,1) \leq Cr$ **do**
  $i = i + 1$
  if $i == n$ **then**
    $i = 1$
  end
  $x_{off}[i] = x'_{off}[i]$
end

FIGURE 6 Exponential crossover pseudo-code

is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) \leq f(x_i)$; otherwise no replacement occurs.

3.2 Algorithmic Issues in Differential Evolution

As shown in Subsection 3.1, DE is based on a very simple idea, i.e. a search by means of adding vectors and a one-to-one spawning for the survivor selection. Thus, DE can be very easily implemented and contains a limited amount of parameters to be tuned (only N_p, F, and CR).

From an algorithmic viewpoint, the success of DE is due to an implicit self-adaptation contained within the algorithmic structure, as shown in Feoktistov (2006). More specifically, since, for each candidate solution, the search rule depends on other solutions belonging to the population (e.g. x_t, x_r, and x_s), the capability of detecting new promising offspring solutions depends on the current distribution of solutions within the decision space. During the early stages of the optimization process, the solutions tend to be spread out within the decision space: this implies that, for a given scale factor value, mutation appears to generate new solutions by exploring the space by means of a large step size (if x_r and x_s are distant solutions, $F\,(x_r - x_s)$ is a vector characterized by a large modulus). In later stages, the solutions tend to concentrate in specific parts of the decision space. Therefore, the step size in the mutation is progressively reduced and the search is performed in the neighborhood of the solutions. In other words, due to its structure, a DE scheme is highly explorative at the beginning of the evolution and subsequently becomes more and more exploitative during the optimization.

Although this mechanism seems, at first glance very efficient, it hides a limitation. If for some reason the algorithm does not succeed at generating offspring solutions which outperform the corresponding parent, the search is repeated again with similar step size values and will likely fail by falling into an undesired stagnation condition (see Lampinen and Zelinka (2000)). Stagnation is that undesired effect which occurs when a population-based algorithm does not converge to a solution (even suboptimal) and the population diversity is still

high. In the case of DE, stagnation occurs when the algorithm, due to the limited amount of exploratory moves, does not manage to improve upon any solution of its population for a prolonged amount of generations.

It is clear that successful functioning of a DE depends on the parameter setting of the three control parameters mentioned above. The population size N_p is related to the amount of possible moving vectors. Over all the possible moves given by a population, some moves are beneficial in the search for the optimum while some others are ineffective and result in a waste of computational effort. Therefore, too small a population size can contain too limited an amount of moves, while too large a population size may contain a high number of ineffective moves which can likely mislead the search. A guideline for sizing population in DE is given in Storn and Price (1997), where a setting of N_p equal to ten times the dimensionality of the problem is proposed. However, a recent study in Neri and Tirronen (2008) shows that a population size lower than the dimensionality of the problem can be optimal in many cases.

Regarding the scale factor F and the crossover rate CR, their setting is neither an intuitive nor a straightforward task but is unfortunately crucial for guaranteeing the algorithmic functioning. Several studies have thus been proposed in literature: for example, in Storn and Price (1997) and Liu and Lampinen (2002b) the settings $F \in [0.5, 1]$ and $CR \in [0.8, 1]$ are recommended; in Liu and Lampinen (2002a) and Rönkkönen et al. (2005) the setting $F = CR = 0.9$ is chosen on the basis of discussion in Price and Storn (1997); finally the empirical analysis reported in Zielinski et al. (2006) shows that in many cases the setting of $F \geq 0.6$ and $CR \geq 0.6$ leads to results having better performance. On the other hand, different studies, e.g. Gämperle et al. (2002) and Mallipeddi and Suganthan (2008), highlight that an efficient parameter setting is very dependent on problems. This result can be seen as a confirmation of the validity of the No Free Lunch Theorem defined by Wolpert and Macready (1997), with reference to the DE framework.

The problem of the parameter setting is emphasized when DE is employed for handling difficulties complex optimization problems, e.g. characterized by a high dimensional or noisy landscape. For example, a large decision space requires a wide range of possible moves to enhance its capability of detecting new promising solutions. However, since, as mentioned before, an enlargement in population size causes an increase in the set of potential ineffective moves, a proper choice of F and CR becomes a crucial aspect in the success of DE in large scale applications. High dimensionality is not the only curse of DE. As highlighted in Krink et al. (2004), DE seems to be inefficient for noisy optimization problems as well. This study experimentally shows that the reason for that is related to an ineffective parameter setting, since a deterministic choice of the scale factor can be inadequate in fitness landscape plagued by noise. Thus, some specific countermeasures must be applied. A simple modification to handle noisy problems has been proposed in Das and Konar (2005) and Das et al. (2005a). A randomized scale factor according to the following formula has been used:

$$F = 0.5\,(1 + rand\,(0, 1)) \tag{8}$$

where $rand(0,1)$ is a uniformly distributed random number between 0 and 1. As shown in Das et al. (2005b), the employment of the scale factor randomization turns out to be beneficial not only for noisy problems but also for stationary problems. Although this topic is not yet clear to computer scientists, a randomization in the scale factor seems to compensate the excessively deterministic search structure of a standard DE and offers new potential search moves. However, while many studies (e.g. Zhenyu et al. (2006), Ali and Fatti (2006) and Nearchou and Omirou (2006)) confirm that the introduction of some "stochasticity" into the DE framework appears to be promising and propose similar approaches, other studies (e.g. Qing (2008), Ali and Törn (2004)) suggest that the employment of excessive randomization in DE is not always considered to be beneficial.

Regarding the population size, some adaptive schemes have been proposed e.g .in Teo (2005, 2006), Teng et al. (2009) and Sing et al. (2007). In Tirronen and Neri (2009) a fitness diversity adaptation for the population size and the other parameters has been proposed (for the fitness diversity adaptation see Caponio et al. (2007) and Neri et al. (2007a,b,c)). In Caponio et al. (2009) the concept of super-fit adaptation is introduced, which consists in controlling the performance of the best individual with respect to the average performance of other individuals of the population. In Zaharie (2003), a multi-population DE approach with a parameter adaptation strategy and population diversity control has instead been proposed. A comparative analysis of some adaptive schemes employed in DE is presented in Zielinski et al. (2008).

Finally, some papers made some attempts in hybridizing DE with other algorithmic structures, like PSO (Hendtlass (2001) and Xu et al. (2008)) and SA (Das et al. (2007) and Hu et al. (2008)). A DE-based Memetic Algorithm employing three local search algorithms coordinated by means of fitness diversity adaptation and a probabilistic scheme is instead proposed in Tirronen et al. (2008).

3.3 Additional Components in Differential Evolution

This section gives a description of four additional components recently introduced in literature, and attempts to justify the algorithmic philosophy which suggests these additions to a standard DE framework. Although these components are very diverse, the common idea is that they allow extra moves that can enrich the original set of DE moves, tending to increase the exploitative pressure within the explorative DE structure.

3.3.1 DE with Trigonometric Mutation

In Trigonometric Differential Evolution (TDE) the mutation operation in eq. (5) is replaced, with a prefixed probability M_t, by an alternative expression, namely trigonometric mutation. This mutation scheme is a greedy operator that, for three given points, generates an offspring by exploiting the most promising search di-

rection. More specifically, following the definition given in Fan and Lampinen (2003), it is formulated as:

$$x'_{off} = \frac{(x_r+x_s+x_t)}{3} + (p_s - p_r)(x_r - x_s) + (p_t - p_s)(x_s - x_t) + (p_r - p_t)(x_t - x_r) \quad (9)$$

where for $k = r, s, t$,

$$p_k = \frac{|f(x_k)|}{|f(x_r)| + |f(x_s)| + |f(x_t)|}. \quad (10)$$

Since this mutation promotes the generation of the offspring along (locally) optimal directions, the employment of this operator within TDE is supposed to increase the exploitative pressure and balance the standard exploration rule of DE. In this sense, it can be seen as a single step local search (see Hart et al. (2004)).

3.3.2 DE with Adaptive Hill Climbing Simplex Crossover

In order to enhance performance of DE, in Noman and Iba (2008) a memetic approach, called Differential Evolution with Adaptive Hill Climbing Simplex Crossover (DEahcSPX), has been proposed. The main idea is that a proper balance of the exploration abilities of DE and the exploitation abilities of a local search can lead to an algorithm with higher performance. The proposed algo-

while *budget condition* **or** $f(C) \geq f(x_b)$ **do**
  select pseudo-randomly $n_p - 1$ individuals from the DE population
  compute the center of mass (including x_b): $O = \frac{1}{n_p}\sum_{i=1}^{n_p} x_i$
  for $i = 1 : n_p - 1$ **do**
    $r_i = rand(0,1)^{\frac{1}{i+1}}$
  end
  for $i = 1 : n_p - 1$ **do**
    $y_i = O + \epsilon(x_i - O)$
  end
  $C_1 = 0$
  for $i = 2 : n_p - 1$ **do**
    $C_i = r_{i-1}(y_{i-1} - y_i + C_{i-1})$
  end
  $C = C_{n_p} + y_{n_p}$
end

FIGURE 7 SPX pseudo-code

rithm uses the DE/rand/1/bin described in section 3.1 as an evolutionary framework within which a local search method, namely the Simplex Crossover (SPX) Tsutsui et al. (1999), is deterministically applied to the best individual of the population. More specifically, at each generation, the individual having the best fitness value, indicated here with x_b, is extracted and the LS described in Fig. 7

is applied (ϵ is a control parameter of the SPX which has been set equal to 1 in Noman and Iba (2008)). If the SPX succeeds in improving upon the starting solution, a replacement occurs according to a meta-Lamarckian logic (Ong and Keane (2004)).

3.3.3 DE with Population Size Reduction

The Differential Evolution with Population Size Reduction (DEPSR) employs, within a DE framework, a variable population size which is progressively reduced during the optimization process, see Brest and Maučec (2008). This population size reduction requires that initial population size N_p^1, total budget T_b (i.e. total number of fitness evaluations) and number of stages N_s (i.e. the number of population sizes employed during the algorithm's run) are prearranged.

The total budget of the algorithm is divided into N_s periods, each period being characterized by a population size value N_p^k (N_p^1 is the initial population size). The population reduction is simply carried out by halving the population size at the beginning of the new stage. In other words, $N_p^{k+1} = N_p^k/2$, for $k = 1, 2, \ldots, N_s - 1$. At the end of each stage, i.e. at each N_g^k generation for $k = 2, 3, \ldots, N_s$, the population is divided, on the basis of the position index i of the individuals, into two unsorted equally-sized sub-populations. Then, a one-to-one selection occurs, so that the corresponding i-th individuals of the two sub-populations are pairwise compared, and the one having the most promising fitness values is retained for the subsequent generation.

The main idea behind this strategy is to focus the search in progressively smaller search spaces in order to prevent a possible stagnation, especially in high-dimensional landscapes. At the beginning of the optimization process, the search requires a highly explorative search rule, i.e. a large population size, in order to explore a large portion of the decision space. During the next stages, the search space is instead progressively narrowed by decreasing the population size, thus exploiting the most promising search directions previously detected.

3.3.4 DE with Scale Factor Local Search

Differential Evolution with Scale Factor Local Search (DESFLS) has been introduced in Tirronen et al. (2009) and extended in Neri et al. (2009) for self-adaptive DE schemes. An improved version that we are referring to here, has been proposed in Neri and Tirronen (2009). The main idea in these algorithms is that a local search is applied, with a certain probability, to the scale factor during the generation of an offspring individual. As shown in Tirronen et al. (2009), local search in the scale factor space can be seen as the minimization over the variable F of fitness function f in the direction given by x_r and x_s and modified by the crossover. More specifically, at first the scale factor local search determines those genes which are undergoing binomial crossover by means of the standard criterion explained in eq. (7), then it attempts to find the scale factor value which guarantees an offspring with the best performance. Thus, for given values of x_t,

x_r, x_s, and the set of design variables to be swapped during the crossover operation, the scale factor local search attempts to solve the following minimization problem:

$$\min_F f(F) \qquad \text{in} \qquad [-1.2, 1.2]\,. \tag{11}$$

For sake of clarity, the procedure describing the fitness function $f(F)$ is shown

```
// ** Fitness function f_F = f(F) **
x'_off = x_t + F (x_r − x_s)
perform crossover and generate x_off
f_F = f(x_off)
return f_F
```

FIGURE 8 Fitness function $f(F)$

in Fig. 8. As described Neri and Tirronen (2009), the meaning of negative values for the scale factor is the inversion of the search direction. In order to perform this minimization, two local search algorithms have been considered and compared in Tirronen et al. (2009) and Neri et al. (2009), namely Golden Section Search (Kiefer (1953)) and Hill-Climb Local Search (see Russell and Norvig (2003)). Experimental results in Neri and Tirronen (2009) showed that cooperative employment of both the algorithms leads to the best performance.

3.4 Modified Structures of Differential Evolution

This section describes DE-based algorithms which modify the standard DE structure, not just introducing additional components. Like the algorithms described in the previous section, all these modifications allow extra moves, thus enriching the search capabilities of the original DE framework.

3.4.1 Self-Adapting Parameter Setting in DE

In order to avoid the manual parameter setting of F and CR, a simple and effective self-adapting strategy has been proposed in Brest et al. (2006), and further extended to large-scale problems in Zamuda et al. (2008), constrained optimization in Brest et al. (2006) and multi-objective problems in Zamuda et al. (2007). This strategy, called Self-Adapting Control Parameters in Differential Evolution, is extensively discussed also in Brest et al. (2007) and Brest et al. (2008). The DE algorithm employing this strategy, namely jDE, consist in a standard DE/rand/1/bin framework (see section 3.1), with the following modifications. With reference to Fig. 5, when the initial population is generated, two extra values between 0 and 1 are also generated per each individual. These values represent F and CR related to the individual under analysis. Each i-th individual is thus composed (in a self-adaptive logic) of its genotype and its control parameters:

$$x_i = (x_i[1], x_i[2], \ldots, x_i[n], F_i, CR_i)\,.$$

When, at each generation, the i-th individual x_i is taken into account and three other individuals are extracted pseudo-randomly, its parameters F_i and CR_i are updated according to the following scheme:

$$F_i = \begin{cases} F_l + F_u rand_1, & \text{if } rand_2 < \tau_1 \\ F_i, & \text{otherwise} \end{cases} \tag{12}$$

$$CR_i = \begin{cases} rand_3, & \text{if } rand_4 < \tau_2 \\ CR_i, & \text{otherwise} \end{cases} \tag{13}$$

where $rand_j$, $j \in \{1, 2, 3, 4\}$, are uniform pseudo-random values between 0 and 1; τ_1 and τ_2 are the probabilities that parameters are updated, and F_l and F_u represent the minimum and the maximum scale factor values, respectively. The newly calculated values of F_i and CR_i are then used for generating the offspring, according to a rand/1/bin strategy.

3.4.2 Opposition Based DE

The Opposition Based Differential Evolution (OBDE), proposed in Rahnamayan et al. (2006b) and Rahnamayan et al. (2008), employs the opposition points logic in order to enhance the search properties of DE and explore a wider portion of the decision space. OBDE has proven successful for solving some of the most difficult problems, e.g. noisy and large scale problems, see Rahnamayan et al. (2006a) and Rahnamayan and Wang (2008). This logic consists in the following. For a given point $x_i = (x_i[1], x_i[2], \ldots, x_i[n])$ belonging to a set $D = [a_1, b_1] \times [a_2, b_2] \times \ldots, \times [a_n, b_n]$, $i = 1, \ldots, N_p$, its opposition point is defined as:

$$\tilde{x}_i = (a_1 + b_1 - x_i[1], a_2 + b_2 - x_i[2], \ldots, a_n + b_n - x_i[n]) \,. \tag{14}$$

The OBDE algorithm consists of a DE framework and two opposition based components: the first after the initial sampling and the second after the survivor selection scheme. After the creation of the initial population, for each point x_i its opposition point $\tilde{x}_i$ is calculated according to the formula (14). The fitness values of both groups of points (original and opposition) are then calculated, and the best N_p individuals are selected for the first generation. At each subsequent generation, with a probability j_r (jump rate), the opposition based component is activated again. In this case, instead of using the predefined boundaries $[a_i, b_i]$, for each point x_i its opposition point is calculated as:

$$\tilde{x}_i = \left(\min_i x_i[1] + \max_i x_i[1] - x_i[1], \ldots, \min_i x_i[n] + \max_i x_i[n] - x_i[n] \right) \tag{15}$$

where $\min_i x_i[j]$ and $\max_i x_i[j]$ are respectively the minimum and maximum values, over the coordinate j, taken by individuals of the population at the present generation. In other words, at each generation, the range of variability of the individuals is taken as a "bounding box" for the calculation of the opposition points. The two sets of points (original and opposition) are then merged and those N_p points having the best performance are selected for the next generation.

3.4.3 DE with Global and Local Neighborhoods

The Differential Evolution with Global and Local Neighborhoods (DEGL), see Chakraborty et al. (2006) and Das et al. (2009a), modifies the mutation operation in DE, explained in Subsection 3.1, by defining a neighborhood as a portion of the population identified by a radius k. More specifically, individuals of the population are pseudo-randomly sorted and each individual is characterized by a position index i. The neighborhood of the i-th individual x_i is given by those individuals $x_{i-k}, \ldots, x_i, \ldots, x_{i+k}$.

The concept of neighborhood is used during the mutation operation since the provisional offspring x'_{off} is generated through a combination of two contributions, with the first one given by the neighborhood individuals and the second by the entire population. Thus, in order to perform the mutation, for an individual x_i, the local contribution is calculated as:

$$L_i = x_i + \alpha \left(x_{n-best} - x_i \right) + \beta \left(x_p - x_q \right) \tag{16}$$

where x_{n-best} is the individual having best performance in the neighborhood; x_p and x_q are two individuals pseudo-randomly selected from the neighborhood. Values α and β are two constants which have a similar role to that of the scale factor F, see eq. (5). Similarly, the global contribution is given by:

$$G_i = x_i + \alpha \left(x_{p-best} - x_i \right) + \beta \left(x_r - x_s \right) \tag{17}$$

where x_{p-best} is that individual having the best performance out of the entire population, x_r and x_s are two individuals pseudo-randomly selected from the population. The two contributions are then combined by means of:

$$x'_{off} = wG_i + (1 - w) L_i \tag{18}$$

where w is a weight factor to be set between 0 and 1.

Regarding the parameter setting, Chakraborty et al. (2006) suggest to set $\alpha = \beta$ equal to a constant value. Also the neighborhood radius k is constant through the iterations. On the contrary, the weight factor w varies during the optimization process. According to the original definition of DEGL, the weight factor varies in the following way:

$$w = w_{\min} + (w_{\max} - w_{\min}) \frac{g}{g_{\max}} \tag{19}$$

where $w_{\min}$ and $w_{\max}$ are the lower and upper bounds of the weight factor, respectively. The indexes g and $g_{\max}$ denote the current generation index and the maximum amount of generations, respectively. In Das et al. (2009a), four alternative weight factor update schemes have been presented and compared, including a self-adaptive scheme that proved to be the most efficient.

3.4.4 Self-Adaptive DE

The first version of Self-Adaptive Differential Evolution (SADE) has been proposed in Qin and Suganthan (2005), while a more sophisticated implementation, here described, has been proposed in Qin et al. (2009). The main feature of these algorithms is the employment of multiple mutation strategies. More specifically, within the genotype of each individual, some probabilities related to the mutation strategy for the subsequent generation are encoded. In other words, for a given candidate solution x_i, the individual is defined as:

$$x_i = \left\langle x_i[1], x_i[2], \ldots, x_i[j], \ldots, x_i[n], F_i, CR_i^1, CR_i^2, CR_i^3, p_i^1, p_i^2, p_i^3, p_i^4 \right\rangle$$

where p_i^k for $k = 1, 2, 3, 4$ is the probability that the mutation strategy 1, 2, 3, or 4, respectively, is employed on the individual x_i ($\sum_{k=1}^{4} p_i^k = 1$). The four mutation strategies considered in Qin et al. (2009) are: DE/rand/1, DE/rand-to-best/2, DE/rand/2, and DE/current-to-rand/1, see Section 3.1, and the binomial crossover is applied in all the cases except the DE/current-to-rand/1.

When the initial sampling is performed for each solution, each probability is set equal to 0.25. During the subsequent LP generations (LP stands for Learning Period), for each individual the number of successful generations n_s^k related to a certain mutation strategy, i.e. the number of offsprings generated by the k^{th} strategy outperforming the generating parent, is saved. In an analogous way, the number of failures n_f^k is also saved.

At the end of the learning period (after LP generations), for each individual x_i, at each generation G, the probabilities p_i^k are updated according to the formula:

$$p_i^k = \frac{S_i^k}{\sum_{k=1}^{4} S_i^k} \tag{20}$$

where

$$S_i^k = \frac{\sum_{g=G-LP}^{G-1} n_s^k}{\sum_{g=G-LP}^{G-1} n_s^k + \sum_{g=G-LP}^{G-1} n_f^k} + \varepsilon \tag{21}$$

where g is the generation index and ε is a small constant value equal to 0.01 whose role is simply to ensure a numerical stability of the algorithm if $S_i^k = 0$ for all the strategies. Thus, S_i^k represents the success rate of the offspring solutions generated by means of the k^{th} strategy. At the end the self-adaptive strategy, given the four probabilities, the mutation strategy is then chosen by means of Stochastic Universal Sampling, see Baker (1987).

As shown above, within each candidate solution, F_i and CR_i are also encoded. The scale factor F_i is generated, for each individual, by sampling a value from a normal distribution having mean value 0.5 and standard deviation 0.3. Regarding the K factor in DE/current-to-rand/1, a randomization is performed by sampling this value from a uniform distribution between 0 and 1.

The setting of CR_i is also self-adaptively performed. When the initial sampling is executed, CR_i^k is set equal to 0.5 for all the individuals and all the strategies (except DE/current-to-rand/1). During the first LP generations, each CR_i^k is updated by sampling a value from a normal distribution having its center in CR_i^k and standard deviation equal to 0.1. During the learning period, for each mutation strategy (except DE/current-to-rand/1), the set of crossover rates which led to a successful offspring generation CR_s^k are saved. After the learning period, CR_i^k is updated by the median of CR_s^k and the new CR_s^k is calculated by sampling a value from a normal distribution having its center in CR_i^k and standard deviation equal to 0.1.

4 COMPACT ALGORITHMS

Despite the rapid development of high performance computational devices, some modern applications still require complex optimization processes to be performed in very limited hardware conditions. This situation is typical, for example, in robotics and control engineering, where costs and volume limitations impose the use of cheap and compact hardware, e.g. a micro-controller or an embedded system, but very complex tasks, such as online training or self-adaptation of some device parameters, must be carried out on board of it.

In these cases, it is crucial to design an efficient optimization algorithm which requires a minimal amount of memory. In the following, we use the term "memory-saving" to indicate those optimization methods requiring a very limited amount of run-time memory and computational resources. Among memory-saving methods, compact algorithms are an efficient alternative to tackle global optimization problems, as they behave like global meta-heuristics while they do not actually process an actual population of candidate solutions. The name "compact algorithms" refers to those algorithms employing the search logic of population-based algorithms without processing and storing an entire population of solutions, but on the contrary using a probabilistic representation of it. This probabilistic representation simulates the population behavior in the sense that it extensively explores the decision space at the beginning of the optimization process, and then progressively focuses the search on the most promising individuals. Most importantly, the probabilistic model allows a much lower algorithmic memory footprint.

Compact algorithms can be considered a subset of a much broader class of modern algorithms called Estimation of Distribution Algorithms (EDAs), which use a probabilistic model but not necessarily replace a population with it. This chapter presents a brief introduction of EDAs, focusing in particular on compact algorithms. The main compact algorithms structures introduced in literature are described in details, namely compact Genetic Algorithm (cGA), the very first implementation of compact algorithms, and compact Differential Evolution (cDE). The next chapter will describe the contribution of this work, which mostly aims at proposing possible algorithmic enhancements in cDE frameworks.

4.1 Estimation of Distribution Algorithms

Estimation of Distribution Algorithms (EDAs), see Larrañaga and Lozano (2001) and Lozano et al. (2006), also called Probabilistic Model Building Genetic Algorithms (PMBGAs), see Pelikan et al. (2002), are a class of evolutionary algorithms which, instead of using classic operators such as crossover and mutation to generate new individuals, rather perform a sampling from a probabilistic model describing the population. The main idea is that the use of a probabilistic model, which could possibly grasp some specific information about the problem structure, guarantees a strong exploitation of the most promising solutions found at each generation. Once the model has been generated, e.g. fitting an existing initial population, it can be dynamically updated, or completely regenerated, using the new solutions sampled during the evolution. The probabilistic model can also substitute entirely the population, thus becoming the representation of a "virtual" population. The pseudo-code of the basic EDA scheme is given in Fig. 9, where N is the number of individuals in the population, m is the number of individuals selected for estimating the distribution, pop_t is the population at the t-th generation, $p_t(x)$ is the probabilistic model describing the probability of finding a generic individual x over the search space D at the t-th generation, and s_t is the set of individuals which are sampled from $p_t(x)$ at the t-th generation.

counter $t = 0$
generate and evaluate N random individuals to create pop_0
while *budget condition* **do**
  select $m \leq N$ (promising) individuals according to a selection method
  use the m individuals to estimate the distribution model $p_t(x)$
  sample and evaluate a set s_t of new individuals from $p_t(x)$
  replace some individuals of pop_t with s_t to create pop_{t+1}
  $t = t + 1$
end

FIGURE 9 EDA pseudo-code

Estimating the distribution is not trivial at all, since there is a trade-off between the accuracy of the model and its computational cost. EDAs are typically classified according to their complexity, i.e. the level of interaction among variables that their probabilistic model includes, see Pelikan et al. (2002). Following this classification, three categories of models have been used in modern EDAs:

- **Univariate**: this is the easiest way to estimate the probability distribution, i.e. considering all the decision variables independent from each other. Under this assumption, the joint probability distribution becomes the product of the marginal probabilities of the n variables:

$$p_t(x) = \prod_{i=1}^{n} p_t(x_i). \tag{22}$$

In other words, instead of using a covariance matrix, which takes into account the correlation among variables, n separate independent distribution models are used. This is the basic principle of Univariate Marginal Distribution Algorithm (UMDA), see Mühlenbein and Paass (1996) Mühlenbein et al. (1996) and Population Based Incremental Learning (PBIL), see Baluja (1994). Compact GA, see Harik et al. (1999) and Mininno et al. (2008), and compact DE, see Mininno et al. (2011), which are described in detail in the following sections, also belong to this class.

- **Bivariate**: these models cover pairwise dependencies among variables, see e.g. Bivariate Marginal Distribution Algorithm (BMDA), Pelikan and Mühlenbein (1999), and MIMIC, De Bonet et al. (1997).

- **Multivariate**: multivariate models take into account correlations among multiple variables. For example, Larrañaga and Lozano (2001) introduce Estimation of Gaussian Networks Algorithm (EGNA) and Estimation of Multivariate Normal Algorithm (EMNA), which make use respectively of a Gaussian network and a multivariate normal density function. A multivariate Gaussian model is also used in Paul and Iba (2003) and Yuan and Gallagher (2005). Pelikan et al. (2000) present instead an EDA in which the probabilistic model is a Bayesian network, called Bayesian Optimization Algorithm (BOA). In Pelikan (2005), BOA is further enhanced in a hierarchical structure (hBOA).

Hybrid and more complex EDA schemes have also been presented. For example, Zhang et al. (2004) introduce an EDA combined with two local search, namely the incomplete simplex method and the unconstrained optimization by diagonal quadratic approximation. Platel et al. (2009) establish instead a connection between EDAs and Quantum-inspired Evolutionary Algorithms (QEAs), which evolve a population of mutually interacting probabilistic models.

As enlightened before, the major advantage of EDAs is that they are inherently able to learn possible dependencies among variables of the problem and use this structural information to efficiently generate new individuals. Moreover, due to the stochasticity of the random sampling, they are less subject to premature convergence. Eventually, they allow a more compact (memory-saving) representation of the population, which is the main interest of this work.

The main drawbacks of EDAs are related to the model complexity. Clearly, more complex models guarantee a better description of complex problem landscapes (for example with highly correlated decision variables), but they introduce a high computational overhead. On the other hand, algorithms employing simple models such as a Gaussian distribution have a fairly reasonable overhead, but in general they cannot handle efficiently multi-modal problems, unless the landscape has a strong basin of attraction. Moreover, this kind of algorithms may even get stuck on unimodal problems, especially in high dimensional spaces: this situation happens when the mean vector of the Gaussian model is far from the global optimum, but the selected individuals used for updating the model are closely

distributed in a small area. In this case, several generations may be needed for the Gaussian to move close to the global optimum (similarly to a hill-climbing logic), but still it is possible that the Gaussian distribution shrinks again before making any significant progress. As suggested in Yuan and Gallagher (2005), this problem is related to the inherent lack of population diversity due to the normal distribution used, which is unimodal (or, in other words, its kurtosis is zero). However, it should be remarked that keeping the diversity high is not always a good solution, because if the Gaussian is very close to the global optimum a diversity increase may even reduce the convergence speed. Yuan and Gallagher (2005) introduce a simple heuristic that, based on a "proximity" threshold between the best individual and the mean vector, "amplifies" the eigenvalues of the covariance matrix so that the distance between the best and the mean, along the corresponding dimensions, is kept at a certain threshold.

Having in mind these issues, it is still possible to design EDAs which show a good compromise between algorithmic complexity and performance. In the next sections we will describe more deeply two major algorithmic structures using an univariate probabilistic model, namely compact Genetic Algorithm (cGA) and compact Differential Evolution (cDE), which despite their simplicity and ”compactness” have proven successful for a broad set of complex problems.

4.2 Binary Compact Genetic Algorithm

The very first implementation of compact algorithms has been the compact Genetic Algorithm (cGA), defined in Harik et al. (1999), which simulates the behaviour of a standard binary encoded Genetic Algorithm (GA). The cGA consists of the following. A binary vector of length n is randomly generated by assigning a 0.5 probability to each gene to take either the value 0 or the value 1. This description of the probabilities, initialized with n values all equal to 0.5, is named as Probability Vector (PV). By means of the PV two individuals are sampled and their fitness values are calculated. The winner solution, i.e. the solution characterized by a higher performance, biases the PV on the basis of a parameter N_p called virtual population. More specifically, if the winner solution in correspondence to its i-th gene displays a 1 while the loser solution displays a 0 the probability value in position i-th of the PV is augmented by a quantity $\frac{1}{N_p}$. On the contrary, if the winner solution in correspondence to its i-th gene displays a 0 while the loser solution displays a 1 the probability value in position i-th of the PV is reduced by a quantity $\frac{1}{N_p}$. If the genes in position i-th display the same value for both the winner and loser solutions, the i-th probability of PV is not modified. For the sake of clarity, the pseudo-code describing the working principles of cGA is displayed in Fig. 10. With the function *compete* we simply mean the fitness-based comparison.

A convergence analysis of cGA is performed in Rastegar and Hariri (2006) by using Markov chains. Paper Ahn and Ramakrishna (2003) analyzes analogies and differences between cGA and $(1+1)$-ES and extends a mathematical model

of ES, Rudolph (2001), to cGA obtaining useful information on the performance. cGA is further extended in Harik (1999) and Harik et al. (2006): the resulting algorithm, called extended compact Genetic Algorithm (ecGA), is based on the idea that the choice of a good probability distribution is equivalent to linkage learning. The measure of a good distribution is based on Minimum Description Length (MDL) models: simpler distributions are better than the complex ones. The probability distribution used in ecGA is a class of probability models known as Marginal Product Models (MPMs). A theoretical analysis of the ecGA behavior is presented in Sastry and Goldberg (2000). A hybrid version of ecGA integrating the Nelder-Mead algorithm is proposed in Sastry and Xiao (2001). A study on the scalability of ecGA is given in Sastry et al. (2007). In Baraglia et al. (2001) a memetic variant of cGA is proposed in order to enhance the convergence performance of the algorithm in the presence of a relatively high number of dimensions. Some applications of cGA are described in Aporntewan and Chongstitvatana (2001), Gallagher et al. (2004) and Jewajinda and Chongstitvatana (2008).

```
counter t = 0
// ** PV initialization **
for i = 1 : n do
    initialize PV [i] = 0.5
end
while budget condition do
    generate 2 individuals a and b by means of PV
    [winner, loser] = compete (a, b)
    // ** PV Update **
    for i = 1 : n do
        if winner [i] ≠ looser [i] then
            if winner [i] == 1 then
                compute PV [i] = PV [i] + 1/N_p
            else
                compute PV [i] = PV [i] − 1/N_p
            end
        end
    end
    t = t + 1
end
```

FIGURE 10 cGA pseudo-code

4.3 Elitism in Compact Algorithms

Two novel versions of cGA have been proposed in Ahn and Ramakrishna (2003). Both of these algorithms still share the same ideas proposed in Harik et al. (1999) but proved to have a significantly better performance compared to their corre-

sponding earlier versions. These two algorithms, namely persistent elitist compact Genetic Algorithm (pe-cGA) and non-persistent elitist compact Genetic Algorithm (ne-cGA), modify the original cGA in the following way. During the initialization, one candidate solution besides the PV, namely *elite*, is also randomly generated. Subsequently, only one (and not two as in cGA) new candidate solution is generated. This solution is compared with the elite. If the elite is the winner solution, the elite biases the PV as shown for the cGA and the elite is confirmed for the following solution generation and consequent comparison. On the contrary, if the newly generated candidate solution outperforms the elite, the PV is updated as shown for the cGA where the new solution is the winner and the elite is the looser. Under these conditions, the elite is replaced by the new solution which becomes the new elite. In the scheme of pe-cGA this replacement occurs only when the elite is outperformed. In the ne-cGA scheme, if an elite is still not replaced after η comparisons, the elite is replaced by a newly generated solution regardless of its fitness value. It must be remarked that whether the persistent or non-persistent scheme is preferable seems to be a problem dependent issue, see Ahn and Ramakrishna (2003). The pseudo-codes highlighting the working principles of pe-cGA and ne-cGA are given in Fig. 11 and Fig. 12, respectively.

```
counter t = 0
// ** PV initialization **
for i = 1 : n do
    initialize PV[i] = 0.5
end
generate elite by means of PV
while budget condition do
    generate 1 individual a by means of PV
    // ** Elite Selection **
    [winner, loser] = compete(a, elite)
    if a == winner then
        elite = a
    end
    // ** PV Update **
    for i = 1 : n do
        if winner[i] ≠ looser[i] then
            if winner[i] = 1 then
                PV[i] = PV[i] + 1/N_p
            else
                PV[i] = PV[i] − 1/N_p
            end
        end
    end
    t = t + 1
end
```

FIGURE 11 pe-cGA pseudo-code

```
counter t = 0 and θ = 0
// ** PV initialization **
for i = 1 : n do
    initialize PV[i] = 0.5
end
generate elite by means of PV
while budget condition do
    generate 1 individual a by means of PV
    // ** Elite Selection **
    [winner, loser] = compete(a, elite)
    θ = θ + 1
    if a == winner or θ ≥ η then
        elite = a
        θ = 0
    end
    // ** PV Update **
    for i = 1 : n do
        if winner[i] ≠ looser[i] then
            if winner[i] == 1 then
                PV[i] = PV[i] + 1/N_p
            else
                PV[i] = PV[i] − 1/N_p
            end
        end
    end
    t = t + 1
end
```

FIGURE 12 ne-cGA pseudo-code

4.4 Real Compact Genetic Algorithm

The real-valued cGA (rcGA) has been introduced in Mininno et al. (2008), and further investigated in Neri et al. (2010). The rcGA is a compact algorithm inspired by the cGA which exports the compact logic to a real-valued domain thus obtaining an optimization algorithm with a high performance despite the limited amount of employed memory resources. In the following, without loss of generality we assume that each design variable is normalized in the interval $[-1, 1]$. In rcGA the PV is not a vector but a $n \times 2$ matrix:

$$PV^t = \left[\mu^t, \sigma^t\right] \tag{23}$$

where μ and σ are, respectively, vectors containing, for each design variable, mean and standard deviation values of a Gaussian Probability Distribution Function (PDF) truncated within the interval $[-1, 1]$. The height of the PDF is normalized in order to keep its area equal to 1. The apex t indicates the generation (number of performed comparison). At the beginning of the optimization process, for

each design variable i, $\mu^1[i] = 0$ and $\sigma^1[i] = \lambda$, where λ is a large positive constant ($\lambda = 10$). This initialization of $\sigma[i]$ values is done in order to simulate a uniform distribution.

Subsequently, one individual is sampled as elite exactly like in the case of pe-cGA or ne-cGA. A new individual is generated and compared with the elite. More specifically, the sampling mechanism of a design variable $x[i]$ associated to a generic candidate solution x from PV consists of the following steps. As mentioned above, for each design variable indexed by i, a truncated Gaussian PDF characterized by a mean value $\mu[i]$ and a standard deviation $\sigma[i]$ is associated. The formula of the PDF is:

$$PDF\left(truncNorm\left(x\right)\right) = \frac{e^{-\frac{(x-\mu[i])^2}{2\sigma[i]^2}}\sqrt{\frac{2}{\pi}}}{\sigma\left[i\right]\left(\text{erf}\left(\frac{\mu[i]+1}{\sqrt{2}\sigma[i]}\right) - \text{erf}\left(\frac{\mu[i]-1}{\sqrt{2}\sigma[i]}\right)\right)} \tag{24}$$

where erf is the error function, see Gautschi (1972). From the PDF, the corresponding Cumulative Distribution Function (CDF) is constructed by means of Chebyshev polynomials according to the procedure described in Cody (1969). It must be observed that the co-domain of CDF is $[0,1]$. In order to sample the design variable $x[i]$ from PV a random number $rand(0,1)$ is sampled from a uniform distribution. The inverse function of CDF, in correspondence of $rand(0,1)$, is then calculated. This latter value is the normalized variable $x[i]$. A graphical representation of the sampling mechanism is given in Fig. 13.

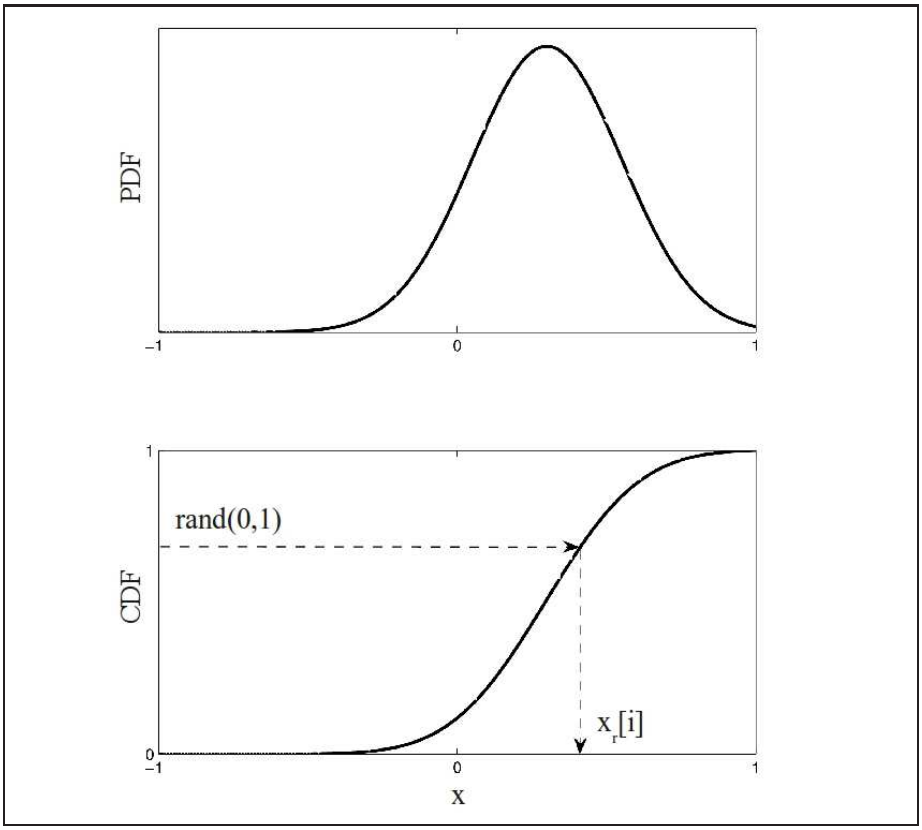

FIGURE 13 Sampling mechanism

counter $t = 0$
// ** PV initialization **
for $i = 1 : n$ **do**
  initialize $\mu\,[i] = 0$
  initialize $\sigma\,[i] = \lambda = 10$
end
generate *elite* by means of PV
while *budget condition* **do**
  // ** Candidate Solution Sampling **
  generate 1 individual x by means of PV
  for $i = 1 : n$ **do**
    generate $rand(0,1)$
    if $rand(0,1) > Cr$ **then**
      $x\,[i] = elite\,[i]$
    end
  end
  // ** Elite Selection **
  $[winner, loser] = compete\,(x, elite)$
  if $x == winner$ **then**
    $elite = x$
  end
  // ** PV Update **
  for $i = 1 : n$ **do**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}\,(winner[i] - loser[i])$
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}\,(winner^2[i] - loser^2[i])}$
  end
  $t = t + 1$
end

FIGURE 14 pe-rcGA pseudo-code

As for the cGA, in rcGA the winner solution biases the PV. The update rule for each element of μ values is given by:

$$\mu^{t+1}\,[i] = \mu^t\,[i] + \frac{1}{N_p}\,(winner\,[i] - loser\,[i])\,, \tag{25}$$

where N_p is virtual population size. The update rule for σ values is given by:

$$\left(\sigma^{t+1}\,[i]\right)^2 = \left(\sigma^t\,[i]\right)^2 + \left(\mu^t\,[i]\right)^2 - \left(\mu^{t+1}\,[i]\right)^2 + \frac{1}{N_p}\left(winner\,[i]^2 - loser\,[i]^2\right). \tag{26}$$

Details for constructing formulas (25) and (26) are given in Mininno et al. (2008). In the same article both persistent and non-persistent structures of rcGA have been tested and it is shown that also in this case the best choice on the elitism seems to be problem dependent. Figure 14 displays the working principle of rcGA with persistent elitism (pe-rcGA).

4.5 Compact Differential Evolution

The compact Differential Evolution (cDE) algorithm, defined in Mininno et al. (2011), modifies the rcGA by sampling from the PDF, not only one solution x, but many solutions according to the mutation rules of Differential Evolution. For

counter $t = 0$
// ** PV initialization **
for $i = 1 : n$ **do**
  initialize $\mu[i] = 0$
  initialize $\sigma[i] = \lambda = 10$
end
generate *elite* by means of PV
while *budget condition* **do**
  // ** Mutation **
  generate 3 individuals x_r, x_s, and x_t by means of PV
  compute $x'_{off} = x_t + F(x_r - x_s)$
  // ** Crossover **
  $x_{off} = x'_{off}$
  for $i = 1 : n$ **do**
    generate $rand(0,1)$
    if $rand(0,1) < Cr$ **then**
      $x_{off}[i] = elite[i]$
    end
  end
  // ** Elite Selection **
  $[winner, loser] = compete(x_{off}, elite)$
  if $x_{off} == winner$ **then**
    $elite = x_{off}$
  end
  // ** PV Update **
  for $i = 1 : n$ **do**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i])$
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])}$
  end
  $t = t + 1$
end

FIGURE 15 pe-cDE/rand/1/bin pseudo-code

example, considering the standard DE/rand/bin, three solutions x_r, x_s, and x_t, are sampled from PV. A provisional offspring x'_{off} is then generated according to the mutation rule of DE/rand/bin (see section 3.1). When the provisional offspring has been generated by mutation, it undergoes binomial crossover with the *elite*, so that the final offspring x_{off} is generated; its fitness value is then computed and is compared with that of the elite solution.

counter $t = 0$
// ** PV initialization **
for $i = 1 : n$ **do**
  initialize $\mu\,[i] = 0$
  initialize $\sigma\,[i] = \lambda = 10$
end
generate *elite* by means of PV
$\theta = 0$
while *budget condition* **do**
  // ** Mutation **
  generate 3 individuals x_r, x_s, and x_t by means of PV
  compute $x'_{off} = x_t + F(x_r - x_s)$
  // ** Crossover **
  $x_{off} = x'_{off}$
  for $i = 1 : n$ **do**
    generate $rand(0,1)$
    if $rand(0,1) < Cr$ **then**
      $x_{off}\,[i] = elite\,[i]$
    end
  end
  // ** Elite Selection **
  $[winner, loser] = compete\,(x_{off}, elite)$
  $\theta = \theta + 1$
  if $x_{off} == winner$ **or** $\theta \geq \eta$ **then**
    $elite = x_{off}$
    $\theta = 0$
  end
  // ** PV Update **
  for $i = 1 : n$ **do**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}\,(winner[i] - loser[i])$
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}\,(winner^2[i] - loser^2[i])}$
  end
  $t = t + 1$
end

FIGURE 16 ne-cDE/rand/1/bin pseudo-code

In the same way explained for the rcGA, *winner* and *loser* solutions are detected and the PV is updated according to formulas (25) and (26). Figure 15 and 16 show, respectively, the pseudo-code of cDE/rand/1/bin with persistent and non-persistent scheme.

The compact Differential Evolution has proven extremely successful for a broad set of problems, especially compared to its population-based DE counterpart. The reasons of the success of cDE are enlightened Mininno et al. (2011). Similarly to some of the approaches described in 3, cDE relies on a "randomized" (due to the sampling mechanism) DE structure: the main difference with those

methods is that instead of imposing a randomization on the control parameters (like jDE and SADE do, see section 3.4), cDE imposes a randomization on the solutions which contribute to the offspring generation. In other words, cDE can be seen as a DE which introduces a randomization within the solution generation and therefore introduces extra search moves which assist the DE structure and attempt to improve upon its performance. Moreover, with respect of other compact algorithms (e.g. cGA), cDE has the advantage of being a straightforward implementation of its population-based equivalent, thus keeping all the basic principles of DE. However good cDE is, it can be further improved in many different ways, e.g. integrating into its basic structure some extra components, as we will see in the next chapter.

5 CONTRIBUTION OF THIS WORK

This chapter briefly describes the contributions of each article presented in this work. Except article **PVIII**, this work focuses on compact algorithms, and mainly on the cDE framework described in the previous chapter, introducing several algorithmic enhancements aimed at improving its performance. Article **PVIII** instead addresses the memory-saving optimization from a different perspective, introducing an extremely simple novel Memetic Computing algorithm specifically designed for applications with limited hardware.

Except the algorithm presented in **PIII**, each cDE-based algorithm described in this work employs persistent elitism. It must be also remarked that, while the original implementation of cDE uses binomial crossover, some of the algorithms presented in this work use exponential crossover, because it turned out to improve the original performance of standard cDE. For further details about different crossover schemes, please refer to chapter 3. Finally, it is worth noticing that all the algorithms use the typical toroidal transformation of DE for bounded problems (see Price et al. (2005)). More specifically, whenever a new point x is generated, if the generic component $x[i]$ falls outside the corresponding interval $[d_1, d_2]$ of the decision space D, it is reassigned within the interval according to the following formula:

$$\begin{cases} x[i] = d_1 + x[i] - d_2 & \text{if } x[i] > d_2 \\ x[i] = d_2 - x[i] + d_1 & \text{if } x[i] < d_1 \end{cases} \tag{27}$$

so that the every new individual is guaranteed to belong to the search space D.

5.1 Memetic Implementations of compact Differential Evolution

Strictly speaking, since cDE (and in general every compact algorithm) does not process a population of solutions but only its probabilistic representation, it cannot be considered as the evolutionary framework of a Memetic Algorithm, such as it has been defined in chapter 2. Nonetheless, a MC approach which includes

a compact optimization as a module of a more complex structure is obviously possible. This section describes two novel MC approaches based on cDE.

5.1.1 Disturbed Exploitation compact Differential Evolution

Article **PI** introduces an unconventional memory-saving MC approach, namely the Disturbed Exploitation cDE (DEcDE). The DEcDE algorithm is composed of a cDE/rand/1/exp framework which, with a prearranged probability M_t, generates the offspring individual by means of the trigonometric mutation proposed described in the chapter 3 (see expressions (9) and (10)), instead of applying DE/rand/1 mutation and exponential crossover. Thus, the trigonometric mutation is a greedy operator that for three given points generates an offspring by exploiting the most promising search directions. The employment of this operator within DEcDE is supposed to offer an exploitative alternative to the standard exploration rule of DE. The trigonometric mutation thus has the role of promoting the generation of the offspring along (locally) optimal directions.

In addition, DEcDE is based on the consideration that the shrinking of the virtual population makes a compact algorithm an exploitative component, more similar to a stochastic local search than to an evolutionary framework. For this reason, a standard combination of a compact algorithm and a local search algorithm can result not an efficient solution for balancing global and local search, see Ishibuchi et al. (2003), Ishibuchi et al. (2007), and Tan et al. (2009), since the resulting algorithm would fail at exploring the decision space from complementary perspectives (Krasnogor (2004)). Thus, the balance between global and local search in the proposed DEcDE algorithm is obtained on the basis of a different idea. The search logic is based on a fairly explorative DE based mutation structure, DE/rand/1. Due to the structure of the exponential crossover, the offspring generation is equivalent to a search in the neighborhood of the parent solution, as only a few design variables are on average involved in the crossover operation. In this sense the variation operators can be considered fairly exploitative. In addition, the trigonometric mutation can be seen as a shallow depth local search algorithm which makes a gradient estimation and attempts to detect a promising solution by following the gradient direction. The trigonometric mutation can also be considered as an exploitative operator. This multiple shallow local search is counterbalanced by an unconventional global search. This global search is performed indirectly by perturbing the PV. More specifically, with a probability M_p the PV is perturbed. Each component $\mu\,[i]$ of the mean value vector μ is perturbed according to the following formula:

$$\mu^{t+1}[i] = \mu^{t+1}[i] + 2\tau \cdot rand\,(0,1) - \tau \tag{28}$$

where τ is a weight representing the maximum amplitude of perturbation. Similar to typical DE schemes, a toroidal mechanism ensures that μ is bounded by 0 and 1 (for example $1 + 0.1 = 0.1$). The perturbation rule for the σ is given by:

$$\left(\sigma^{t+1}[i]\right)^2 = \left(\sigma^{t+1}[i]\right)^2 + \tau \cdot rand\,(0,1)\,. \tag{29}$$

In other words, DEcDE does not contain an explicit global search operator, but it rather employs a local search which is periodically "disturbed". The moving

counter $t = 0$
// ** *PV* initialization **
for $i = 1 : n$ **do**
  initialize $\mu[i] = 0$
  initialize $\sigma[i] = \lambda$
end
generate *elite* by means of *PV*
while *budget condition* **do**
  if $rand(0,1) < M_t$ **then**
    // ** Trigonometric Mutation **
    generate 3 individuals x_r, x_s, and x_t by means of *PV*
    $$x_{off} = \frac{(x_r+x_s+x_t)}{3} + (p_s - p_r)(x_r - x_s) + (p_t - p_s)(x_s - x_t) + (p_r - p_t)(x_t - x_r)$$
  else
    // ** Mutation **
    generate 3 individuals x_r, x_s, and x_t by means of *PV*
    $x'_{off} = x_t + F(x_r - x_s)$
    // ** Crossover **
    apply exponential crossover and generate x_{off}
  end
  // ** Elite Selection **
  $[winner, loser] = compete(x_{off}, elite)$
  if $x_{off} == winner$ **then**
    $elite = x_{off}$
  end
  // ** *PV* Update **
  for $i = 1 : n$ **do**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i])$
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])}$
  end
  // ** *PV* Perturbation **
  if $rand(0,1) < M_p$ **then**
    for $i = 1 : n$ **do**
      $\mu^{t+1}[i] = \mu^{t+1}[i] + 2\tau \cdot rand(0,1) - \tau$
      $\sigma^{t+1}[i] = \sqrt{(\sigma^{t+1}[i])^2 + \tau \cdot rand(0,1)}$
    end
  end
  $t = t + 1$
end

FIGURE 17 DEcDE pseudo-code

operators of mutations and crossover are supposed to detect promising search

directions and quickly exploit them. This fact corresponds to the convergence of the virtual population towards the elite.

This convergence is likely to be premature. The perturbation mechanism then inhibits the algorithmic convergence and forces the algorithm to search elsewhere in the decision space, possibly detecting new promising solutions. In other words, DEcDE can be seen as a multi-start local search algorithm which performs a highly exploitative mini-search between each pair of PV perturbations. A pseudo-code displaying the working principles of DEcDE is given in Fig. 17.

5.1.2 Super-Fit compact Differential Evolution with PSR

Article **PV** proposes a different cDE-based MC approach, namely the Super-Fit cDE with Population Size Reduction (SFcDE-PSR). This algorithm employs a super-fit scheme, which has proven successful in population-based EAs (see e.g. Caponio et al. (2009)), and the Population Size Reduction (PSR) mechanism described in chapter 3. The super-fit mechanism consists in the injection of an elite individual ("super-fit"), generated by means of an exploitative algorithm at the beginning of the optimization process, into the initial virtual population. The algorithm used for finding the super-fit individual is the Rosenbrock Algorithm (see section 1.1.2). It must be remarked that, since the Rosenbrock Algorithm makes use of an $n \times n$ rotation matrix, it cannot be considered memory-saving *per se*. Anyway, since it can be applied as a pre-processing step to feed the super-fit individual to the cDE framework, SFcDE-PSR can still be considered memory-saving. The exploitative pressure given by the super-fit mechanism is partially counterbalanced by the employment of a PSR scheme, with a large population size at the first stages of the optimization process performed by cDE (thus allowing a broader exploration), which is halved progressively as the optimization goes on. For the sake of clarity, the pseudo-code of SFcDE-PSR is given in Fig. 18.

5.2 Structured Population in Compact Algorithms

Analogous to structured population algorithms, in which individuals are distributed over several sub-populations, compact algorithms can be arranged in multiple virtual populations. To give a graphical idea of this concept, each compact unit performs the search of a different region of decision space while pieces of information regarding the achieved improvements are somehow exchanged amongst compact units and used for finding the global optimum. This section describes some possible structures employing multiple cDE units. It is important to notice that these structures cannot be considered strictly memory-saving. Considering that each cDE unit, with an implementation performing in-place operations, needs four n-dimensional vectors (where n is the problem dimension), an algorithm employing N cDE units requires of course $4N$ vectors. Therefore, the total algorithmic memory footprint is obviously larger than a single cDE, but

it is still lower than population-based algorithms employing tens of individuals. Nevertheless, due to their robustness and versatility, the proposed algorithms can be considered powerful general-purpose global optimization tools.

counter $t = 0$
// ** *PV* initialization **
for $i = 1 : n$ **do**
  initialize $\mu[i] = 0$
  initialize $\sigma[i] = \lambda$
end
generate *elite* by means of *PV*
// ** Super-fit generation **
while *budget condition* **or** *tolerance condition* **do**
  apply Rosenbrock Algorithm to *elite*
end
replace the original *elite* with the super-fit individual
calculate remaining budget and Population Size Reduction conditions
while *budget condition* **do**
  // ** Mutation **
  generate 3 individuals x_r, x_s, and x_t by means of *PV*
  $x'_{off} = x_t + F(x_r - x_s)$
  // ** Crossover **
  apply binomial crossover and generate x_{off}
  // ** Elite Selection **
  $[winner, loser] = compete\left(x_{off}, elite\right)$
  if $x_{off} == winner$ **then**
    $elite = x_{off}$
  end
  // ** *PV* Update **
  for $i = 1 : n$ **do**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}\left(winner[i] - loser[i]\right)$
    $\sigma^{t+1}[i] = \sqrt{\left(\sigma^t[i]\right)^2 + \left(\mu^t[i]\right)^2 - \left(\mu^{t+1}[i]\right)^2 + \frac{1}{N_p}\left(winner^2[i] - loser^2[i]\right)}$
  end
  $t = t + 1$
  // ** Virtual Population Size Reduction **
  if $t ==$ *Population Size Reduction condition* **then**
    $N_p = \frac{N_p}{2}$
  end
end

FIGURE 18 SFcDE-PSR pseudo-code

5.2.1 Composed compact Differential Evolution

Article **PII** proposes a possible coordination scheme for multiple cDE units. The resulting algorithm, called Composed cDE (CcDE), employs (N_c) distributed cDE

units, each one applying a rand/1/exp scheme and the perturbation logic shown in formulas (28) and (29).

While the N_c compact units evolve independently, two mechanisms promote the communication amongst the units. In order to understand both these mechanisms let us consider the compact units to be arranged according to a ring topology. In other words, each m^{th} unit has two neighbor units: unit $(m-1)^{th}$ and unit $(m+1)^{th}$. For the sake of clarity, we remark that the neighbors of the N_c^{th} unit are $(N_c-1)^{th}$ and 1^{st} units, respectively. The first mechanism is the unidirectional migration (see Tasoulis et al. (2004)) of the elite individual. More specifically, at each step (comparison between offspring and elite) of each compact unit, with a probability M_e, the elite solution $elite^m$ is duplicated and replaces the solution in $elite^{m+1}$ if the sender outperforms the receiver. In other words, $elite^m$ is overwritten in $elite^{m+1}$ if $f\left(elite^m\right) < f\left(elite^{m+1}\right)$ (minimization problem).

The second mechanism is the scale factor inheritance proposed in Weber et al. (2010). The scale factor inheritance mechanism occurs contextually with the elite migration. More specifically, when the migration occurs the $(m+1)^{th}$ unit inherits the scale factor F^m after a perturbation. More specifically, the scale factor F^{m+1} related to the $(m+1)^{th}$ unit is updated according to the following formula:

$$F^{m+1} = F^m + \alpha\mathcal{N}(0,1) \tag{30}$$

where $\mathcal{N}(0,1)$ is a pseudo-random value sampled from a normal distribution characterized by a zero mean and variance equal to 1. The constant value α has the role of controlling the range of perturbation values $\alpha\mathcal{N}(0,1)$. It must be observed that there are no bounds for the variation of F. On the contrary, an unbounded variation of the control parameter has been allowed by relying on the self-adaptation mechanism. A graphical representation of the proposed CcDE is given in Fig. 19. Each compact unit is schematically represented as truncated

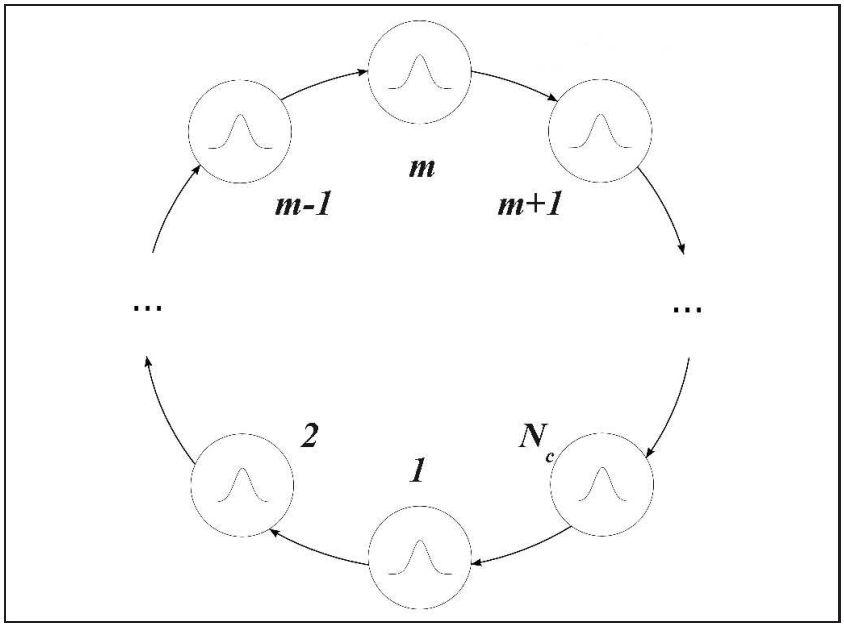

FIGURE 19 Graphical representation of CcDE

Gaussian distribution function. The arrows indicate elite migration and scale factor inheritance mechanism. For the sake of clarity, the pseudo-code of CcDE is also given in Fig. 20.

```
for m = 1 : N_c do
    // ** PV initialization **
    for i = 1 : n do
        initialize μ[i] = 0
        initialize σ[i] = λ
    end
end
generate N_c elite solutions x_e by means of the corresponding PV
while budget condition do
    for m = 1 : N_c do
        // ** cDE/rand/1/exp **
        generate individuals for rand/1 mutation
        apply exponential crossover and generate offspring
        compare offspring and elite
        update PV
        // ** PV perturbation **
        if rand(0,1) < M_p then
            apply perturbation of the PV
        end
        if rand(0,1) < M_e then
            if f(elite^m) < f(elite^{mod(m+1,N_c)}) then
                // ** elite migration **
                send a copy of the elite individual to the neighbor unit
                replace the elite individual: elite^{mod(m+1,N_c)} = elite^m
                // ** scale factor inheritance **
                replace the scale factor: F^{mod(m+1,N_c)} = F^m + αN(0,1)
            end
        end
    end
end
```

FIGURE 20 CcDE pseudo-code

5.2.2 Ensemble compact Differential Evolution

In article **PVII** the idea of ensemble of parameters and strategies, which has proven successful in DE, see Mallipeddi et al. (2011), is introduced into the cDE framework to improve its performance. Following the same approach as CcDE, the resulting algorithm, called ENcDE, employs a set of (N_c) cDE units arranged according to a ring topology (see Fig. 19), each one having its own probability vector PV and evolving independently. In order to promote the communication amongst the units, the migration scheme used in CcDE is employed. In this case

only the elite undergoes migration, while each cDE unit uses its own parameters.

```
for m = 1 : N_c do
    // ** PV initialization **
    for i = 1 : n do
        initialize μ[i] = 0
        initialize σ[i] = λ
    end
    // ** initialize strategies and parameters **
    initialize mutation strategy
    initialize crossover strategy
    initialize N_p, Cr, F (and F)
end
generate N_c elite solutions x_e by means of the corresponding PV
while budget condition do
    for m = 1 : N_c do
        // ** cDE **
        generate individuals for mutation
        apply crossover and generate offspring
        compare offspring and elite
        update PV
    end
    if rand(0,1) < M_e then
        if f(elite^m) < f(elite^{mod(m+1,N_c)}) then
            // ** elite migration **
            send a copy of the elite individual to the neighbor unit
            replace the elite individual: elite^{mod(m+1,N_c)} = elite^m
        end
    end
end
```

FIGURE 21 ENcDE pseudo-code

The novelty of ENcDE consists in the introduction of a pool of mutation strategies, crossover strategies, N_p, F and CR values. More specifically, the pool consists of the following strategies and parameters:

- mutation strategies: DE/rand/1 and DE/current-to-rand/1
- crossover strategies: binomial and exponential
- $N_p \in \{10, 100\}$
- $F, K \in \{0.5, 0.7, 0.9\}$
- $Cr \in \{0.1, 0.5, 0.9\}$

At the beginning of the optimization process, each of the (N_c) cDE units randomly picks from the pool a virtual population size, a mutation and crossover strategy, as well as the associated parameters. Each unit then undergoes mutation, crossover and selection, as described in 4.5. After selection, each PV vector is updated accordingly to formulas (28) and (29), and elite migration is performed, based on a predetermined migration probability M_e. Elite replacement is carried out in the same way as CcDE (see 5.2.1). For the sake of clarity the pseudo-code of ENcDE is given in Fig. 21.

5.2.3 Supervised compact Differential Evolution

An alternative way of performing the coordination is the supervised system introduced in article **PIV**, where the Supervised cDE (ScDE) has been proposed. In ScDE the coordination is performed by a central unit which attempts to enhance all the elite solutions and returns them to each compact unit. More specifically, ScDE is composed of cDE/rand/1/exp units employing the perturbation rules shown in formulas (28) and (29) as well as a supervision component, namely the jDE algorithm (see section 3.4). Obviously, other schemes can be used as a supervisor components but jDE was selected after an empirical testing of various algorithms.

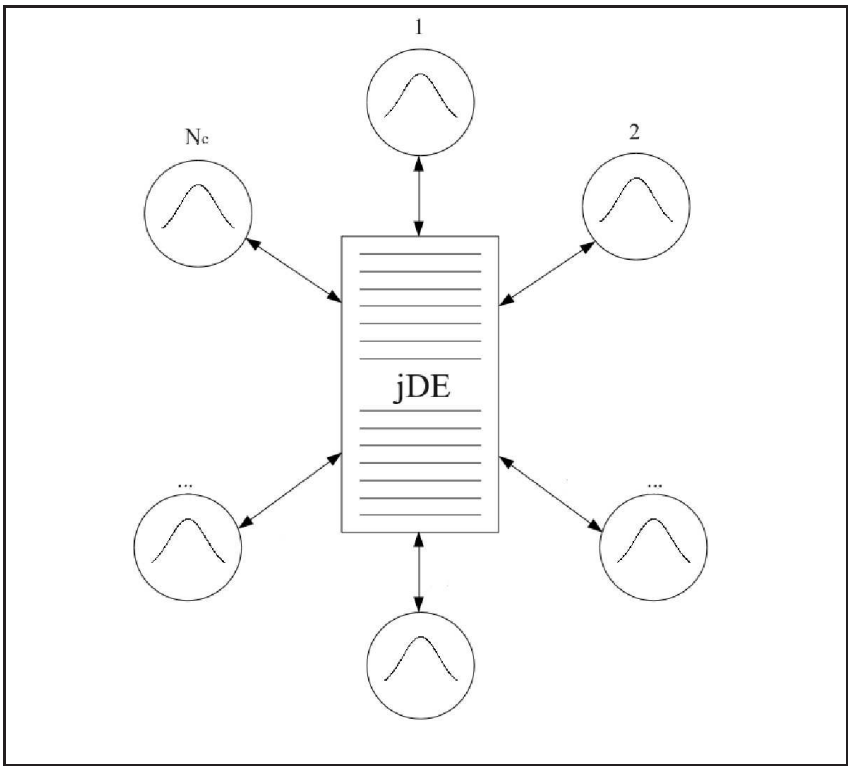

FIGURE 22 Graphical representation of ScDE

Each unit performs one offspring generation and possible elite replacement. When all the compact units performed one step, all the elite solutions are inserted into an auxiliary population. Within this auxiliary population, the candidate so-

lutions (the elites) are processed by means of one generation of global optimizer. After one generation, a new population of elite solutions is produced.

counter $t = 0$
for $m = 1 : N_c$ **do**
  `// ** PV initialization **`
  for $i = 1 : n$ **do**
    initialize $\mu[i] = 0$
    initialize $\sigma[i] = \lambda$
  end
end
generate N_c elite solutions x_e by means of the corresponding PV
while *budget condition* **do**
  `// ** cDE units **`
  for $m = 1 : N_c$ **do**
    `// ** cDE rand/1/exp **`
    generate individuals for mutation
    apply exponential crossover and generate offspring
    compare offspring and elite
    update PV
    `// ** PV perturbation **`
    if $rand(0,1) < M_p$ **then**
      apply perturbation of the PV
    end
    inject elite into the supervision unit
  end
  `// ** supervision unit (jDE) **`
  for $m = 1 : N_c$ **do**
    `// ** F_m update **`
    generate $rand_1$ and $rand_2$

$$F_m = \begin{cases} F_l + F_u rand_1, & \text{if } rand_2 < \tau_1 \\ F_m, & \text{otherwise} \end{cases}$$

    `// ** mutation **`
    perform standard DE/rand/1 mutation
    `// ** Cr_m update **`
    generate $rand_3$ and $rand_4$

$$CR_m = \begin{cases} rand_3, & \text{if } rand_4 < \tau_2 \\ CR_m, & \text{otherwise} \end{cases}$$

    `// ** crossover **`
    perform standard binomial crossover with the Cr_m calculated
    `// ** selection **`
    perform the standard DE one-to-one spawning selection
  end
  `// ** update elites **`
  inject each elite into its corresponding cDE unit
end

FIGURE 23 ScDE pseudo-code

The elite solutions are then injected into the corresponding compact units and replace the old elite solutions. It is obvious that a global optimizer employing a non-sorting selection mechanism (e.g. DE based algorithms) is preferable as it allows a natural reinsertion of the solutions into the corresponding units. The supervision unit has the role of recombining the local achievements carried out by each compact unit and promote their exploitation in order to perform an efficient global search. The newly improved elite solutions after the application of the supervision unit should locally promote the search of unexplored areas of the decision space by affecting the values characterizing the virtual populations. A graphical representation of the supervised model is given in Fig. 22. The double arrows indicate the migration of the elite, the compact units are indicated with a circle and the probabilistic representation of the population, while the rectangular in the middle represents the supervisor unit. For the sake of clarity, the pseudo-code of ScDE is given in Fig. 23.

5.3 Additional Components in compact Differential Evolution

In this section the integration of two components which have proven successful for DE, namely Opposition Based Learning and Noise Analysis, into the cDE framework is described. The first one provides extra moves to the standard DE search logic (see section 3.4), allowing the algorithm to explore different regions of the landscape. The latter is a simple yet powerful way to tackle problems characterized by noise on fitness, since it introduces an intelligent re-sampling procedure in the fitness evaluation process.

5.3.1 Opposition-based compact Differential Evolution

Opposition Based Learning (OBL) is a technique which has been applied, in several circumstances, to enhance the performance of Differential Evolution, see e.g. the OBDE algorithm described in section 3.4. OBL consists of the generation of additional points generated by making use of a central symmetry within a hyper-rectangle in the search space. The OBL scheme has been further improved in the Generalized Opposition-based DE (GODE), see Wang et al. (2011), by integrating some randomness and a progressive narrowing of the search. Combining randomization and opposition learning, GODE provides an alternative logic (or equivalently, an alternative move) that allows to check, under certain probability conditions, unexplored areas of the decision space by means of a projection with respect to a focus (opposition points). On the other hand, dynamically updating the interval boundaries for the generation of opposition points, GODE progressively narrows the search and exploits the available genotype and detects high quality solutions, thus counterbalancing the natural explorative tendency of the DE logic. This dynamic update depends on the population's spread. Since the population is supposed to focus on the most promising areas of the decision

space, this update, corresponds, *de facto*, to a progressive narrowing of the hyper-rectangle of the opposition points generation.

counter $t = 0$
// ** *PV* initialization **
for $i = 1 : n$ **do**
  initialize $\mu[i] = 0$
  initialize $\sigma[i] = \lambda$
end
generate *elite* by means of *PV*
while *budget condition* **do**
  // ** Mutation **
  generate 3 individuals x_r, x_s, and x_t by means of *PV*
  $x'_{off} = x_t + F(x_r - x_s)$
  // ** Crossover **
  apply exponential/binomial crossover and generate x_{off}
  // ** Generalized Opposition-Based Learning **
  if $rand(0,1) < j_r$ **then**
    $k = rand(0,1)$
    $\tilde{x}_{off} = 2k\mu - x_{off}$
    if $f(\tilde{x}_{off}) \leq f(x_{off})$ **then**
      $x_{off} = \tilde{x}_{off}$
    end
  end
  // ** Elite Selection **
  $[winner, loser] = compete(x_{off}, elite)$
  if $x_{off} == winner$ **then**
    $elite = x_{off}$
  end
  // ** *PV* Update **
  for $i = 1 : n$ **do**
    $\mu^{t+1}[i] = \mu^t[i] + \frac{1}{N_p}(winner[i] - loser[i])$
    $\sigma^{t+1}[i] = \sqrt{(\sigma^t[i])^2 + (\mu^t[i])^2 - (\mu^{t+1}[i])^2 + \frac{1}{N_p}(winner^2[i] - loser^2[i])}$
  end
  $t = t + 1$
end

FIGURE 24 cODE pseudo-code

Article **PVI** proposes the integration of the generalized opposition based learning into the compact Differential Evolution framework and tests its impact on the algorithmic performance. The proposed algorithm, called cODE, consists of a standard pe-cDE/rand/1 scheme (either employing binomial or exponential crossover), in which when the offspring is generated, its opposition point is calculated with a probability j_r, called jump rate. More specifically, the opposition point $\tilde{x}_{off}$ of the offspring x_{off} is computed according to the following expres-

sion:

$$\tilde{x}_{off} = k(a + b) - x_{off} \tag{31}$$

where k is a random number uniformly sampled between 0 and 1, and a and b are the vectors representing the bounds of the hyper-rectangle containing the population. It must be observed that for $k = 1$, the generalized scheme coincides with the OBDE scheme described in chapter 3.4. Since in compact optimization a population of solutions is not available, the bounds of the population hyper-rectangle are identified by $a = \mu - \alpha \cdot \delta$ and $b = \mu + \alpha \cdot \delta$, where α is an arbitrary constant. By substituting the values of compact bounds within equation (31), the following formula is obtained:

$$\tilde{x}_{off} = 2k\mu - x_{off}. \tag{32}$$

It must be observed that the value of σ does not appear in the formula since the population is assumed to be symmetrical with respect to the mean value μ. If the opposition point is generated, its fitness is calculated and compared with the fitness value of the original offspring. The most promising solution is then retained and its fitness value is compared with that of the elite individual x_e. Following that, the usual PV update of cDE is performed. For the sake of clarity, the pseudo-code of cODE is given in Fig. 24.

5.3.2 Noise Analysis compact Differential Evolution

Article **PIII** focuses on the class of optimization problems with a noisy fitness function. This situation is typical, for instance, of problems where the fitness is computed by means of measurement devices. For a complete classification of uncertainties in optimization, see Jin and Branke (2005). As summarized in Di Pietro et al. (2004), the noise in the objective function causes two types of undesirable behavior: 1) a candidate solution may be underestimated and thus eliminated, 2) a candidate solution may be overestimated, thus saved and allowed to lead towards incorrect search directions. In other words, a noise fitness landscape can be seen as characterized by false optima which consequently mislead the algorithm search, see Neri and Mäkinen (2007). Under these conditions, optimization algorithms can be easily misled by noise and thus detect unsatisfactory solutions. Although Evolutionary Algorithms, thanks to their stochastic nature, appear to behave robustly in noisy environments, see Beyer and Sendhoff (2006), and Arnold and Beyer (2006), they still can perform poorly when the fitness landscape is affected by noise. As highlighted in Branke and Schmidt (2003), the most critical operation is the selection since it requires a fitness-based comparison that in the presence of noise can jeopardize the entire selection and search process.

To tackle these issues from the perspective of memory-saving optimization, article **PIII** proposes a novel cDE-based scheme, namely Noise Analysis cDE (NAcDE). NAcDE consists of a ne-cDE/rand/1 bin framework, see section 4.5, employing a noise analysis survivor selection scheme. The choice of a non persistent elitism scheme is crucial in noisy environments, in order to avoid that overestimated solutions mislead the search, see Mininno et al. (2011): indeed, since a

compact algorithm stores in memory only one solution, the elite, if the elite selection has been performed over-estimating the quality of the solution, the entire search can be jeopardized if a persistent elitism is used. If on one hand the non-persistent elitism guarantees a periodic refreshment of the genotypes leading the search, on the other the noise analysis survivor selection performs a fine-tuning in the noise filtering.

```
// ** Compete function with Noise Analysis **
// ** [winner, loser] = compete(x_off, x_e) **
winner = x_e
loser = x_off
if |f̄(x_e) − f̄(x_off)| > 2σ then
    if f̄(x_off) ≤ f̄(x_e) then
        winner = x_off
        loser = x_e
    end
else
    α = min{f̄(x_e), f̄(x_off)}
    β = max{f̄(x_e), f̄(x_off)}
    ν = (α+2σ−(β−2σ)) / (β+2σ−(α−2σ))
    n_s = min( ⌈(1.96 / (2·(1−ν)))^2⌉ , n_smax )
    perform re-sampling
    update f̄(x_e) and f̄(x_off)
    update fitness counter
    if f̄(x_off) ≤ f̄(x_e) then
        winner = x_off
        loser = x_e
    end
end
```

FIGURE 25 Noise Analysis survivor selection pseudo-code

More specifically, the noise analysis component performs an analysis of the noise and automatically re-samples a proper number n_s of solutions in order to ensure both reliable pairwise comparisons and a minimal cost in terms of fitness evaluations. In other words, n_s represents the minimum amount of samples which ensure a reliable characterization of the noise distribution, i.e. the amount of samples which allows that the average fitness value can be considered as the mean value of a distribution. In order to avoid infinite sampling, n_s is saturated to an upper limit n_{smax}, which is the only extra parameter to be set with respect to the standard cDE. The setting of this parameter can be intuitively carried out e.g. on the basis of the global computational budget available and the precision requirement in the specific application. When additional samples are performed, the average fitness values $\bar{f}$ are updated and the solution characterized by the most promising average fitness is selected for subsequent generation. It must be highlighted that the noise analysis assumes that the noise is Gaussian and that the

standard deviation of the noise can be estimated. This situation obviously does not cover all the optimization problems in noisy environments, but it is anyway typical of most industrial applications. The pseudo-code of NAcDE is the same as standard ne-cDE/rand/1/bin, see Fig. 16. The function *compete*() used for comparing the offspring and the elite is modified as depicted in Fig. 25.

5.4 A Different Memory-Saving Approach: Single-Solution MC

Article **PVIII** addresses the problem of memory-saving optimization following a different approach. More specifically, it introduces an extremely simple single-solution Memetic Computing method, called Three Stage Optimal Memetic Exploration (3SOME). This algorithm is composed of three memes: the first two are stochastic, respectively with a "long" and a "moderate" search radius, while the third one is deterministic and with a short search radius. The bottom-up combination of the three operators is coordinated by means of a natural, simple sequential trial and error logic. In addition to the extreme compactness of the resulting algorithm, the design choices behind it lead to the conclusion that complexity in algorithmic structures can be unnecessary, if not detrimental, and that simple bottom-up approaches can be competitive as well.

A philosophical interpretation of this conclusion can be given invoking the Ockham's Razor, which can be expressed in the following way: *entia non sunt multiplicanda praeter necessitatem* (entities must not be multiplied beyond necessity). In Memetic Computing, entities are memes, in the sense of search operators, and their quantity refers to their multiple coordination to construct complex algorithmic structures. In other words, in order to obtain an algorithm with a good performance, as well as limited memory footprint and computational overhead, a correct algorithmic design approach should be to start from a *tabula rasa* (blank slate, white paper) and build up the algorithm with a few simple memes, rather than starting from fully working algorithms and complicate them further. In any case, the complexity of the algorithm should never be excessive with respect to the problem(s) to be solved.

Looking at the logic of 3SOME in detail, during the long distance exploration, similar to a stochastic global search, a new solution is sampled within the entire decision space by using a crossover (see Fig. 26). In other words, this exploration stage performs a global stochastic search, attempting to detect unexplored promising basins of attraction. On the other hand, while this search mechanism extensively explores the decision space, it also promotes retention of a small section of the elite within the trial solution, which appears to be extremely beneficial in terms of performance with respect to a stochastic blind search (which would generate a completely new solution). This mechanism is repeated until it does not detect a solution that outperforms the original elite. When a new promising solution is detected, and thus the elite is updated, the middle distance exploration is activated, in order to allow a more focused search around the new solution.

```
generate a random solution x_t within D
generate i = round (n · rand (0, 1))
x_t[i] = x_e[i]
while rand (0, 1) ≤ Cr do
    x_t[i] = x_e[i]
    i = i + 1
    if i == n then
        i = 1
    end
end
if f (x_t) ≤ f (x_e) then
    x_e = x_t
end
```

FIGURE 26 Long distance exploration

```
construct a hypercube with side width δ centered in x_e
for j = 1 : k × n do
    generate a random solution x_t within the hypercube
    generate i = round (n · rand (0, 1))
    x_t[i] = x_e[i]
    while rand (0, 1) ≤ Cr′ do
        x_t[i] = x_e[i]
        i = i + 1
        if i == n then
            i = 1
        end
    end
    if f (x_t) ≤ f (x_e) then
        x_e = x_t
    end
end
```

FIGURE 27 Middle distance exploration

During the middle distance exploration, a hyper-cube is generated around the candidate solution and some points are stochastically generated within it, in order to explore a limited bounded region of the decision space (see Fig. 27). In other words, this stage attempts to focus the search around promising solutions in order to determine whether the current elite deserves further computational budget or other unexplored areas of the decision space must be explored. More specifically, a hyper-cube whose edge has side width equal to δ is constructed around the elite solution x_e. Within this region, a fixed number of trial points is generated by random perturbing the elite along a limited number of dimensions, thus making a randomized exploitation of the current elite solution. At the end of this stage, if the elite has been updated a new hypercube is constructed around the new elite and the search is repeated. On the contrary, if the middle

distance exploration does not lead to an improvement, an alternative search logic is applied, that is the deterministic logic of the short distance exploration.

```
while local budget condition do
    x_t = x_e
    x_s = x_e
    for i = 1 : n do
        x_s[i] = x_e[i] − ρ
        if f(x_s) ≤ f(x_t) then
            x_t = x_s
        else
            x_s[i] = x_e[i] + ρ/2
            if f(x_s) ≤ f(x_t) then
                x_t = x_s
            end
        end
    end
    if f(x_t) ≤ f(x_e) then
        x_e = x_t
    else
        ρ = ρ/2
    end
end
```

FIGURE 28 Short distance exploration

```
generate the solution x_e
while global budget condition do
    while x_e is not updated do
        apply to x_e the long distance exploration as in Fig. 26
    end
    while x_e is updated do
        apply to x_e the middle distance exploration as in Fig. 27
    end
    apply to x_e the short distance exploration as in Fig. 28
    if x_e has been updated then
        apply middle distance exploration as in Fig. 27
    else
        apply long distance exploration as in Fig. 26
    end
end
```

FIGURE 29 Coordination of the exploration stages

During the short distance exploration, a simple deterministic local search is applied to the solution, in order to quickly exploit the most promising search directions and refine the search, see Fig. 28. The meaning of this exploration stage

is to perform the descent of promising basins of attraction and possibly finalize the search if the basin of attraction is globally optimal. De facto, the short distance exploration is a simple steepest descent deterministic local search algorithm, with an exploratory logic similar to that of Hooke-Jeeves algorithm, see section 1.1.2. This exploration is repeated until a prefixed budget is exceeded. After that, if there is an improvement in the quality of the solution, the focused search of middle distance exploration is repeated subsequently. Otherwise, if no improvement in solution quality is found, the long distance search is activated again to attempt to find new basins of attractions. For the sake of clarity, the pseudo-code displaying the working principle of 3SOME and highlighting the coordination amongst the three levels of exploration is given in Fig. 29.

5.5 Comparative Analysis of the Proposed Algorithms

In order to compare the complexity of all the algorithms presented in this chapter, their algorithmic components and memory footprint are reported in Table 1. The memory footprint is expressed in terms of minimum amount of memory slots required by the algorithm, where a memory slot is the memory space occupied by a vector having n components, n being the dimensionality of the problem. For the sake of simplicity, the memory requirement for scalar variables (e.g. fitness values, temporary variables, etc.) has been neglected for all the algorithms.

As shown in Table 1, all the algorithms employing a single cDE-based unit require four memory slots. This fact can be explained as follows. One slot is needed to store the elite. It must be remarked that the elite is kept in memory in all the cases, also in non persistent structures, see for example NAcDE. Two other slots are needed to store the PV, respectively one for μ and one for σ. The fourth slot is needed to store the trial solution, i.e. the offspring. It is worthwhile noticing that, in order to use only one slot for offspring generation, an in-place implementation is needed. Considering for example rand/1 mutation, an extra scalar variable is used to store the i-th component of x_r, x_s and x_t, while the i-th component of x'_{off} is used as accumulator to perform in-place the mutation operation $x'_{off} = x_r + F(x_s - x_t)$. Similarly, when crossover (either binary or exponential) is applied, the two solutions undergoing it, namely the elite and x'_{off}, are already stored in memory. Thus x_{off} can be generated simply replacing some components of x'_{off}.

Using a similar implementation, all the algorithms employing multiple cDE units, namely CcDE, ENcDE and ScDE, obviously require a number of slots multiple of four, according to the number of cDE units. In other words, these algorithms are memory-wise more expensive (at least compared to single cDE-based structures), but still they can be used as general purpose optimizers. Analogously, the structure employing the Rosenbrock Algorithm as super-fit, namely SFcDE-PSR, is affected by an additional memory overhead due to the use of a $n \times n$ rotation matrix, equivalent to the employment of n additional memory slots.

TABLE 1 Algorithmic complexity of the proposed algorithms

Algorithm	Components	Memory slots
DEcDE	cDE based structure trigonometric or rand/1 mutation exponential crossover PV perturbation	4
SFcDE-PSR	cDE based structure rand/1/bin strategy Super-Fit (Rosenbrock algorithm) Population Size Reduction	$4+n$
CcDE	N_c cDE units rand/1/exp strategy PV perturbation elite migration & scale factor inheritance	$4N_c$
ENcDE	N_c cDE units multiple mutation/crossover strategies elite migration	$4N_c$
ScDE	N_c cDE units rand/1/exp strategy PV perturbation supervision unit (jDE)	$4N_c$
cODE	cDE based structure rand/1 mutation, bin/exp crossover Opposition Based Learning	4
NAcDE	cDE based structure (non persistent) rand/1/bin strategy Noise Analysis	4
3SOME	single-solution structure 3 sequential operators trial and error coordination	3

On the other hand, the 3SOME framework is characterized by a slightly lower memory footprint (only three slots), compared to the single cDE-based structures. In this case, one slot is needed for the elite, one for the trial solution (in both long and middle distance operators), and one to store the initial elite which is used for replacements in the short distance operator.

As for the computational overhead, i.e. the computing time spent to perform the algorithmic operations (excluding the computing time of fitness evaluations), an in-depth analysis is reported in paper **PVIII**. To summarize this analysis, on a predetermined landscape the overhead of 3SOME is almost seven times lower than the overhead of DEcDE (considering a Matlab R2009b implementation executed on a PC Intel Core 2 Duo 2.4 GHz with 4 GB RAM employing GNU/Linux Ubuntu 10.04). Intuitively, this ratio is valid also with respect to the other cDE-based algorithms, including those employing multiple cDE units, which are anyway executed sequentially. The additional overhead of cDE-based structures is related to the sampling operation, which involves the computation of the CDF and its inverse. In this regard, the 3SOME framework appears more appropriate when the application is plagued by hard real-time constraints.

6 CONCLUSION

Despite the rapid development of high performance computational devices, some applications impose the solution of complex optimization problems in very limited hardware conditions. This situation is typical for example in robotics and control engineering where, in order to have modest cost and volume of the device, normally a general-purpose computer with large computational resources and memory capacity cannot be used and the optimization must be instead performed within a micro-controller or an embedded system, which is often endowed with minimal hardware features.

In these cases, it is crucial to design an efficient optimization algorithm which requires a minimal amount of memory. Among memory-saving methods, compact algorithms are an efficient alternative to tackle global optimization problems, as they behave like global meta-heuristics while they do not actually process an actual population of candidate solutions. With the exception of the last article presented in this thesis, which introduces a single-solution Memetic Computing approach, this work focuses on a specific compact algorithm, namely the compact Differential Evolution (cDE). Some possible algorithmic enhancements have been proposed, for example introducing additional components to the original framework (i.e. Population Size Reduction, Trigonometric Mutation and Perturbation Exploitation, Opposition Based Learning and Noise Analysis) which greatly improve the performance of standard cDE, still making a limited use of memory. On the other hand, structures employing multiple cDE units have also been investigated: although having a slightly higher memory footprint, the resulting algorithms introduce an interesting concept, that is using multiple compact algorithms as distributed local search methods. In these structures, each cDE unit explores the search space from a different perspective and shares, in a memetic fashion, part of its knowledge with other memes.

Related to that, a possible alternative to compact algorithms is finally introduced in the last article included. A promising single-solution Memetic Computing method is presented, consisting in three embarrassingly simple memes coordinated by means of a natural trial-and-error logic. While the first two memes explore the search space using very simple stochastic rules, the last one refines

the search with a classic deterministic exploitation.

The main finding of this work is that extremely simple algorithms, if carefully designed, may have the same performance as most of the modern state-of-the-art sophisticated optimization methods. In other words, keeping the algorithmic design simple, and having clearly in mind the contribution of each algorithmic component, it is still possible to obtain good results on complex problems.

Future work in the field of memory-saving optimization will go towards multiple directions. Related to compact algorithms, possible directions are the investigation of different probabilistic models of the population (e.g. composed multi-modal distribution functions), and the extension of the compact logic to other computational paradigms (e.g. Particle Swarm and Bacterial Foraging Optimization). As to what memory-saving MC regards, the most interesting research area appears to be the implementation of some sort of "self-organizing" memetic structure, that is an intelligent system which can automatically design a memetic framework, and adapt it to the specific problem. Just like we humans do, such systems may analyze the problem and solve it, in a very efficient way, resorting to an "algorithmic knowledge database", i.e. a collection of memes with different properties and solving abilities. Keeping those memes simple is, in the view of the author, the very first step towards this exciting idea.

YHTEENVETO (FINNISH SUMMARY)

Viimeisen kymmenen vuoden aikana "laskennallinen älykkyys" on lisääntynyt merkittävästi teollisuuden ja kulutuselektroniikan sulautetuissa järjestelmissä. Tällaisia sulautettuja järjestelmiä löytyy esimerkiksi kaukoantureista ja robottiruohonleikkureista. Laskennallinen älykkyys voi perustua siihen että järjestelmä ratkaisee numeerisesti tietyn optimointiongelman, toisin sanoen laite "oppii" lisää käytön aikana. Optimointiongelman numeeriseen ratkaisemiseen tuo lisähaasteen se, että sulautetussa järjestelmässä on yleensä hyvin rajalliset laskenta- ja muistiresurssit.

Tämä työ, jonka suomenkielinen otsikko on "Muistia säästäviä optimointimenetelmiä rajallisilla laskentaresursseilla varustetuille järjestelmille", esittelee niin sanottuja kompakteja evoluutioalgoritmeja edellä mainittujen optimointitehtävien tehokkaaseen ratkaisemiseen.

Evoluutioalgoritmit ovat matemaattisia optimointialgoritmeja, jotka matkivat luonnossa havaittavaa evoluutiota (sopeutuminen ja vahvimman selviytyminen). Ratkaisukandidaatit muodostavat populaation jonka sisällä tapahtuu mutaatiota, risteytymistä ja vahvimpien yksilöiden "perimän" siirtymistä seuraaville sukupolville. Memeettiset algoritmit ovat evoluutioalgoritmien luokka, jossa perinteisiä evoluutiopohjaisia algoritmeja tehostetaan paikallisilla hakuheuristiikoilla. Tässä työssä tutkitut ja edelleen kehitetyt kompaktit evoluutioalgoritmit poikkeavat perinteisistä menetelmistä siinä, että suurta ratkaisukandidaattipopulaatiota ei talleteta laskentayksikön muistiin. Sen sijaan talletetaan vain populaation statistiikkaa kuvaavat parametrit. Työssä on tarkasteltu pääasiassa kompaktin differentiaalievoluution nimellä tunnettujen optimointialgoritmien edelleen kehittämistä. Lisäksi tekijä esittelee uuden, "yhden ratkaisun memeettiseksi algoritmiksi" nimetyn menetelmän.

Tämän työn päätulos on havainto, että äärimmäisen yksinkertaisilla, mutta huolellisesti suunnitelluilla optimointialgoritmeillakin voidaan päästä tyydyttäviin tuloksiin monimutkaisia optimointitehtäviä ratkaistaessa.

APPENDIX 1 BENCHMARK FUNCTIONS

This appendix contains the definitions of the test function used in the included articles. Two major benchmarks have been used, namely the 25 Benchmark Functions for the Congress on Evolutionary Computation 2005 (Special Session on Real-Parameter Optimization) and the 7 Benchmark Functions for the Congress on Evolutionary Computation 2008 (Special Session and Competition on Large Scale Global Optimization). In order to have a heterogeneous set of test functions, with different features and different dimensions, and perform fair comparisons among the proposed methods and the state-of-the-art algorithms (which, in some cases, may be tuned on a specific benchmark), in most of the papers these two benchmarks have been mixed together with functions defined on other sources, see appendix 1.3. For each test function, the relative parameters (dimension, decision space, etc.) and properties (modes, scalability, separability, shift, rotation, noise, etc.) are outlined.

In the next, the following notation is used. D is the problem dimension; $\mathbf{x}$ is a row vector $(1 \times D)$, whose i-th component is indicated with x_i. All other vectors and matrices and matrices are expressed using the same notation. Unless specified differently, all other numbers in the following equations are scalars.

APPENDIX 1.1 Benchmark Functions - CEC 2005

Shifted Sphere Function: f_1 from Suganthan et al. (2005) with $D = 30$.

$$f_1(\mathbf{x}) = \sum_{i=1}^{D} z_i^2 + f_{bias_1}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: unimodal, shifted, separable, scalable.

Shifted Schwefel's Problem 1.2: f_2 from Suganthan et al. (2005) with $D = 30$.

$$f_2(\mathbf{x}) = \sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_j \right)^2 + f_{bias_2}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: unimodal, shifted, non-separable, scalable.

Shifted Rotated High Conditioned Elliptic Function: f_3 from Suganthan et al. (2005) with $D = 30$.

$$f_3(\mathbf{x}) = \sum_{i=1}^{D} \left(10^6\right)^{\frac{i-1}{D-1}} z_i^2 + f_{bias_3}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: unimodal, shifted, rotated, non-separable, scalable.

Shifted Schwefel's Problem 1.2 with Noise in Fitness: f_4 from Suganthan et al. (2005) with $D = 30$.

$$f_4(\mathbf{x}) = \left(\sum_{i=1}^{D} \left(\sum_{j=1}^{i} z_j \right)^2 \right) \cdot (1 + 0.4\,|N(0,1)|) + f_{bias_4}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: unimodal, shifted, non-separable, scalable, noise in fitness.

Schwefel's Problem 2.6 with Global Optimum on Bounds: f_5 from Suganthan et al. (2005) with $D = 30$.

$$f_5(\mathbf{x}) = max_i\,(|\mathbf{A}_i\mathbf{x} - \mathbf{B}_i|) + f_{bias_5}$$

where $\mathbf{A}$ is a $D \times D$ matrix, a_{ij} are integer random numbers in the range $[-500, 500]$, $det(\mathbf{A}) \neq 0$, $\mathbf{A}_i$ is the i-th row of $\mathbf{A}$, $\mathbf{B}_i = \mathbf{A}_i \times \mathbf{o}$, $\mathbf{o}$ is a $D \times 1$ vector, with o_i are random numbers in the range $[-100, 100]$, corresponding to the optimum. Decision space $[-100, 100]^D$. Properties: unimodal, non-separable, scalable.

Shifted Rosenbrock's Function: f_6 from Suganthan et al. (2005) with $D = 30$.

$$f_6(\mathbf{x}) = \sum_{i=1}^{D} \left(100 \left(z_i^2 - z_{i+1} \right)^2 + (z_i - 1)^2 \right) + f_{bias_6}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: multi-modal, shifted, non-separable, scalable, having a very narrow valley from local optimum to global optimum.

Shifted Rotated Griewank's Function without Bounds: f_7 from Suganthan et al. (2005) with $D = 30$.

$$f_7(\mathbf{x}) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} cos\left(\frac{z_i}{\sqrt{i}} \right) + 1 + f_{bias_7}$$

where $\mathbf{z} = (\mathbf{x} - \mathbf{o}) \times \mathbf{M}$, $\mathbf{M} = \mathbf{M'}(1 + 0.3\,|N(0,1)|)$, $\mathbf{M'}$ a linear transformation matrix, and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Initial population in $[0, 600]^D$. Properties: multi-modal, rotated, shifted, non-separable, scalable, no bounds for $\mathbf{x}$.

Shifted Rotated Ackley's Function with Global Optimum on Bounds: f_8 from Suganthan et al. (2005) with $D = 30$.

$$f_8(\mathbf{x}) = -20exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} z_i^2} \right) - exp\left(\frac{1}{D}\sum_{i=1}^{D} cos\,(2\pi z_i) \right) + 20 + e + f_{bias_8}$$

where $\mathbf{z} = (\mathbf{x} - \mathbf{o}) \times \mathbf{M}$, $\mathbf{M}$ a linear transformation matrix, and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-32, 32]^D$. Properties: multi-modal, rotated, shifted, non-separable, scalable, global optimum on the bounds.

Shifted Rastrigin's Function: f_9 from Suganthan et al. (2005) with $D = 30$.

$$f_9(\mathbf{x}) = \sum_{i=1}^{D} \left(z_i^2 - 10\cos 2\pi z_i + 10 \right) + f_{bias_9}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-5, 5]^D$. Properties: multi-modal, shifted, separable, scalable, huge number of local optima.

Shifted Rotated Rastrigin's Function: f_{10} from Suganthan et al. (2005) with $D = 30$.

$$f_{10}(\mathbf{x}) = \sum_{i=1}^{D} \left(z_i^2 - 10\cos 2\pi z_i + 10 \right) + f_{bias_{10}}$$

where $\mathbf{z} = (\mathbf{x} - \mathbf{o}) \times \mathbf{M}$, $\mathbf{M}$ a linear transformation matrix, and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-5, 5]^D$. Properties: multi-modal, shifted, rotated, non-separable, scalable, huge number of local optima.

Shifted Rotated Weierstrass Function: f_{11} from Suganthan et al. (2005) with $D = 30$.

$$f_{11}(\mathbf{x}) = \sum_{i=1}^{D} \sum_{k=0}^{k_{max}} \left(a^k cos 2\pi b^k (z_i + 0.5) \right) - D \sum_{k=0}^{k_{max}} \left(a^k cos 2\pi b^k \cdot 0.5 \right) + f_{bias_{11}}$$

where $a = 0.5$, $b = 0.3$, $k_{max} = 20$, $\mathbf{z} = (\mathbf{x} - \mathbf{o}) \times \mathbf{M}$, $\mathbf{M}$ a linear transformation matrix, and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-0.5, 0.5]^D$. Properties: multi-modal, shifted, rotated, non-separable, scalable, continuous but differentiable only on a set of points.

Schwefel's Problem 2.13: f_{12} from Suganthan et al. (2005) with $D = 30$.

$$f_{12}(\mathbf{x}) = \sum_{i=1}^{D} (\mathbf{A}_i - \mathbf{B}_i(x))^2 + f_{bias_{12}}$$

where $\mathbf{A}_i = \sum_{j=1}^{D} (a_{ij} sin\alpha_j + b_{ij} cos\alpha_j)$, $\mathbf{B}_i(x) = \sum_{j=1}^{D} (a_{ij} sin x_j + b_{ij} cos x_j)$, for $i = 1, \ldots, D$, $\mathbf{A}$ and $\mathbf{B}$ are two $D \times D$ matrices, a_{ij} and b_{ij} are integer random numbers in the range $[-100, 100]$, and the shifted optimum $\alpha = [\alpha_1, \alpha_2, \ldots, \alpha_D]$, with α_j random numbers in the range $[-\pi, \pi]$. Decision space $[-\pi, \pi]^D$. Properties: multi-modal, shifted, non-separable, scalable.

Shifted Expanded Griewank's plus Rosenbrock's Function: f_{13} from Suganthan et al. (2005) with $D = 30$, f_2 and f_8 defined as above.

$$\begin{aligned} f_{13}(\mathbf{x}) = \; & f_8(f_2(z_1, z_2)) + f_8(f_2(z_2, z_3)) + \ldots \\ & f_8(f_2(z_{D-1}, z_D)) + f_8(f_2(z_D, z_1)) + f_{bias_{13}} \end{aligned}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o} + \mathbf{1}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-5, 5]^D$. Properties: multi-modal, shifted, non-separable, scalable.

Shifted Rotated Expanded Scaffer's F6 Function: f_{14} from Suganthan et al. (2005) with $D = 30$.

$$f_{14}(\mathbf{x}) = F(z_1, z_2) + F(z_2, z_3) + \ldots + F(z_{D-1}, z_D) + F(z_D, z_1) + f_{bias_{14}}$$
$$F(x, y) = 0.5 + \frac{sin^2\sqrt{x^2+y^2} - 0.5}{(1+0.001(x^2+y^2))^2}$$

where $\mathbf{z} = (\mathbf{x} - \mathbf{o}) \times \mathbf{M}$, $\mathbf{M}$ a linear transformation matrix, and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: multi-modal, shifted, non-separable, scalable.

Hybrid Composition Function: f_{15} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, separable near the global optimum (Rastrigin), scalable, a huge number of local optima, different functions properties are mixed together, Sphere functions give two flat areas for the function.

Rotated Version of Hybrid Composition Function f_{15}: f_{16} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, Sphere functions give two flat areas for the function.

Rotated Version of Hybrid Composition Function f_{15} with Noise in Fitness: f_{17} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, Sphere functions give two flat areas for the function, with Gaussian noise in fitness.

Rotated Hybrid Composition Function: f_{18} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, Sphere functions give two flat areas for the function, a local optimum is set on the origin.

Rotated Hybrid Composition Function with narrow basin global optimum: f_{19} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, Sphere functions give two flat areas for the function, a local optimum is set on the origin, a narrow basin for the global optimum.

Rotated Hybrid Composition Function with Global Optimum on the Bounds: f_{20} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties:

multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, Sphere functions give two flat areas for the function, a local optimum is set on the origin, a narrow basin for the global optimum.

Rotated Hybrid Composition Function: f_{21} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together.

Rotated Hybrid Composition Function with High Condition Number Matrix: f_{22} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, global optimum is on the bound.

Non-Continuous Rotated Hybrid Composition Function: f_{23} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, non-continuous, global optimum is on the bound.

Rotated Hybrid Composition Function: f_{24} from Suganthan et al. (2005) with $D = 30$. Decision space $[-5, 5]^D$. Properties: multi-modal, rotated, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, unimodal functions give flat areas for the function.

Rotated Hybrid Composition Function without bounds: f_{25} from Suganthan et al. (2005) with $D = 30$. Initial population in $[2, 5]^D$. Properties: multi-modal, non-separable, scalable, a huge number of local optima, different functions properties are mixed together, unimodal functions give flat areas for the function, no bounds for $\mathbf{x}$.

APPENDIX 1.2 Benchmark Functions - CEC 2008

Shifted Sphere Function: f_1 from Tang et al. (2007) with $D = 100$.

$$f_1(\mathbf{x}) = \sum_{i=1}^{D} z_i^2 + f_{bias_1}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: unimodal, shifted, separable, scalable.

Shifted Schwefel Problem 2.21: f_2 from Tang et al. (2007) with $D = 100$.

$$f_2(\mathbf{x}) = \max_{i=1}^{D} |z_i| + f_{bias_2}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: unimodal, shifted, non-separable, scalable.

Shifted Rosenbrock's Function: f_3 from Tang et al. (2007) with $D = 100$.

$$f_3(\mathbf{x}) = \sum_{i=1}^{D-1} \left(100\left(z_i^2 - z_{i+1}\right)^2 + (z_i - 1)^2\right) + f_{bias_3}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o} + \mathbf{1}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-100, 100]^D$. Properties: multi-modal, shifted, non-separable, scalable.

Shifted Rastrigin's Function: f_4 from Tang et al. (2007) with $D = 100$.

$$f_4(\mathbf{x}) = \sum_{i=1}^{D} \left(z_i^2 - 10\cos(2\pi z_i) + 10\right) + f_{bias_4}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-5, 5]^D$. Properties: multi-modal, shifted, separable, scalable, huge number of local optima.

Shifted Griewank's Function: f_5 from Tang et al. (2007) with $D = 100$.

$$f_5(\mathbf{x}) = \sum_{i=1}^{D} \frac{z_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{z_i}{\sqrt{i}}\right) + 1 + f_{bias_5}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-600, 600]^D$. Properties: multi-modal, shifted, non-separable, scalable.

Shifted Ackley's Function: f_6 from Tang et al. (2007) with $D = 100$.

$$f_6(\mathbf{x}) = -20e^{-0.2\sqrt{1/D\sum_{i=1}^{D} z_i^2}} - e^{(1/D)\sum_{i=1}^{D}\cos(2\pi z_i)} + 20 + e + f_{bias_6}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-32, 32]^D$. Properties: multi-modal, shifted, non-separable, scalable.

FastFractal DoubleDip Function: f_7 from Tang et al. (2007) with $D = 100$.

$$f_7(\mathbf{x}) = \sum_{i=1}^{D} fractal1D\left(x_i + twist\left(x_{(i \bmod D)+1}\right)\right)$$

$$twist(y) = 4\left(y^4 - 2y^3 + y^2\right)$$

$$fractal1D(x) \approx \sum_{k=1}^{3} \sum_{1}^{2^{k-1}} \sum_{1}^{ran2(o)} doubledip\left(x, ran1(o), \frac{1}{2^{k-1}(2 - ran1(o))}\right)$$

$$doubledip\,(x,c,s) =$$
$$\begin{cases} \left(-6144\,(x-c)^6 - 3088\,(x-c)^4 - 392\,(x-c)^2 + 1\right) s & -0.5 < x < 0.5 \\ 0 & \text{otherwise} \end{cases}$$

where $ran1(o)$ and $ran2(o)$ are, respectively, a double and an integer, pseudo-randomly chosen, with seed o, with equal probability from the interval $[0,1]$ and the set $\{0,1,2\}$. Decision space $[-1,1]^D$. Properties: multi-modal, non-separable, scalable.

APPENDIX 1.3 Benchmark Functions - Extra

Shifted non-continuous Rastrigin's Function: f_{11} from Qin et al. (2009) with variable D.

$$f\,(\mathbf{x}) = 10 + D\sum_{i=1}^{D}\left(y_i^2 - 10\cos 2\pi y_i\right)$$

$$y_i = \begin{cases} z_i & \text{if } |z_i| < 1/2 \\ round(2z_i)/2 & \text{if } |z_i| \geq 1/2 \end{cases}$$

where $\mathbf{z} = \mathbf{x} - \mathbf{o}$ and the shifted optimum $\mathbf{o} = [o_1, o_2, \ldots, o_D]$. Decision space $[-500,500]^D$. Properties: multi-modal, shifted, rotated, separable, scalable, huge number of local optima.

Schwefel's Function: f_{12} from Qin et al. (2009) with variable D.

$$f\,(\mathbf{x}) = 418.9829D + \sum_{i=1}^{D}\left(-x_i \sin\sqrt{|x_i|}\right).$$

Decision space $[-500,500]^D$. Properties: multi-modal, separable, scalable.

Schwefel Problem 2.22: f_2 from Vesterstrøm and Thomsen (2004) with variable D.

$$f\,(\mathbf{x}) = \sum_{i=1}^{D}|x_i| + \prod_{i=1}^{D}|x_i|.$$

Decision space $[-10,10]^D$. Properties: unimodal, separable, scalable.

Generalized penalized Function 1: f_{12} from Vesterstrøm and Thomsen (2004) with variable D.

$$\begin{aligned} f\,(\mathbf{x}) \;=\; & \tfrac{\pi}{D}\left\{10\sin^2\pi y_1 + \textstyle\sum_{i=1}^{D}\left((y_i-1)^2\left(1+10\sin^2\pi y_i\right)\right) + (y_D-1)^2\right\} \\ & + \textstyle\sum_{i=1}^{D} u(x_i,10,100,4) \end{aligned}$$

where

$$y_i = 1 + \frac{1}{4}(x_i+1)$$

and

$$u(x,a,k,m) = \begin{cases} k(x_i-a)^m & \text{if } x_i > a \\ 0 \quad \text{if } |x_i| \leq a & \\ k(-x_i-a)^m & \text{if } x_i < -a \end{cases}$$

Decision space $[-50,50]^D$. Properties: multi-modal, separable, scalable.

Generalized penalized Function 2: f_{13} from Vesterstrøm and Thomsen (2004) with variable D.

$$\begin{aligned} f(\mathbf{x}) &= \tfrac{1}{10}\left\{\sin^2 3\pi x_1 + \textstyle\sum_{i=1}^{D-1}\left((x_i-1)^2\left(1+\sin^2 3\pi x_{i+1}\right)\right)\right\} + \\ &\quad \tfrac{1}{10}\left\{(x_D-1)\left(1+\sin 2\pi x_n\right)^2\right\} + \textstyle\sum_{i=1}^{D} u(x_i,5,100,4) \end{aligned}$$

where $u(x,a,k,m)$ is defined as above. Decision space $[-50,50]^D$. Properties: multi-modal, separable, scalable.

Michalewicz's Function: from Michalewicz (1996) with variable D.

$$f(\mathbf{x}) = -\sum_{i=1}^{D} \sin(x_i)\left[\sin\left(\frac{ix_i^2}{\pi}\right)\right]^{2m}$$

where $m = 10$. Decision space $[0,\pi]^D$. Properties: multi-modal, separable, scalable.

Kowalik's Function: f_{15} from Yao et al. (1999) with $D = 4$.

$$f(\mathbf{x}) = \sum_{i=1}^{11}\left[a_i - \frac{x_1\left(b_i^2+b_i x_2\right)}{b_i^2+b_i x_3+x_4}\right]^2$$

where a_i and b_i, $i = 1,\ldots,11$ are fixed coefficients (see Yao et al. (1999)). Decision space $[-5,5]^D$. Properties: multi-modal, non-separable.

Six-hump camel-back Function: f_{20} from Qin et al. (2009) with $D = 2$.

$$f(\mathbf{x}) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4$$

Decision space $[-5,5]^D$. Properties: multi-modal (2 global optima and 4 local optima), non-separable.

Branin Function: f_{17} from Vesterstrøm and Thomsen (2004) with $D = 2$.

$$f(\mathbf{x}) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right)^2 + 10\left(1-\frac{1}{8\pi}\right)\cos x_1 + 10$$

Decision space $[-5,10]\times[0,15]$. Properties: multi-modal (3 global optima), non-separable.

Hartman's Family: f_{19} and f_{20} from Vesterstrøm and Thomsen (2004), respectively with $D = 3$ and $D = 6$.

$$f(\mathbf{x}) = -\sum_{i=1}^{4} c_i \exp\left[-\sum_{j=1}^{D} a_{ij}(x_j - p_{ij})^2\right]$$

where a_{ij}, c_i and p_{ij} are fixed coefficients (see Vesterstrøm and Thomsen (2004)). Decision space $[0,1]^D$. Properties: multi-modal (one global optimum and 4 and 6 local optima, respectively), separable.

Shekel's Family: $f_{21} - f_{23}$ from Vesterstrøm and Thomsen (2004), with $D = 4$.

$$f(\mathbf{x}) = -\sum_{i=1}^{m} \left[(\mathbf{x} - \mathbf{A}_i)(\mathbf{x} - \mathbf{A}_i)^T + c_i\right]^{-1}$$

where $m = 5, 7, 10$ for f_{21}, f_{22}, and f_{23}, respectively, $\mathbf{A}$ is a $m \times D$ matrix with fixed coefficients, $\mathbf{A}_i$ is the i-th row of $\mathbf{A}$, and c_i are fixed coefficients (see Vesterstrøm and Thomsen (2004)). Decision space $[0,10]^D$. Properties: multi-modal (five, seven and ten local optima, respectively), non-separable.

APPENDIX 2 STATISTIC METHODS FOR COMPARING ALGORITHMS

In this work two non-parametric statistic tests have been used to perform an *a posteriori* analysis of the performance of the various algorithms considered, namely the Wilcoxon Rank-Sum Test, which performs a single hypothesis test, and the Holm-Bonferroni procedure, which performs multiple hypotheses tests simultaneously. In this appendix these two methods are briefly described and their application to the analysis of the algorithm results is explained.

To clarify the concept of null-hypothesis, suppose that we have two different independent samples A and B composed, respectively, of n_A and n_B observations. If the data in the two sets A and B can be considered independent samples from identical continuous distributions with equal means, the null-hypothesis is accepted ($H_0 : A = B$). Otherwise, the null-hypothesis is rejected and either $H_1 : A < B$ or $H_1 : A > B$ (alternative hypothesis), meaning that the two distributions are clearly separated and it is possible to determine which one has the lowest mean. A graphical representation of the null-hypothesis accepted or rejected is shown in Fig. 30.

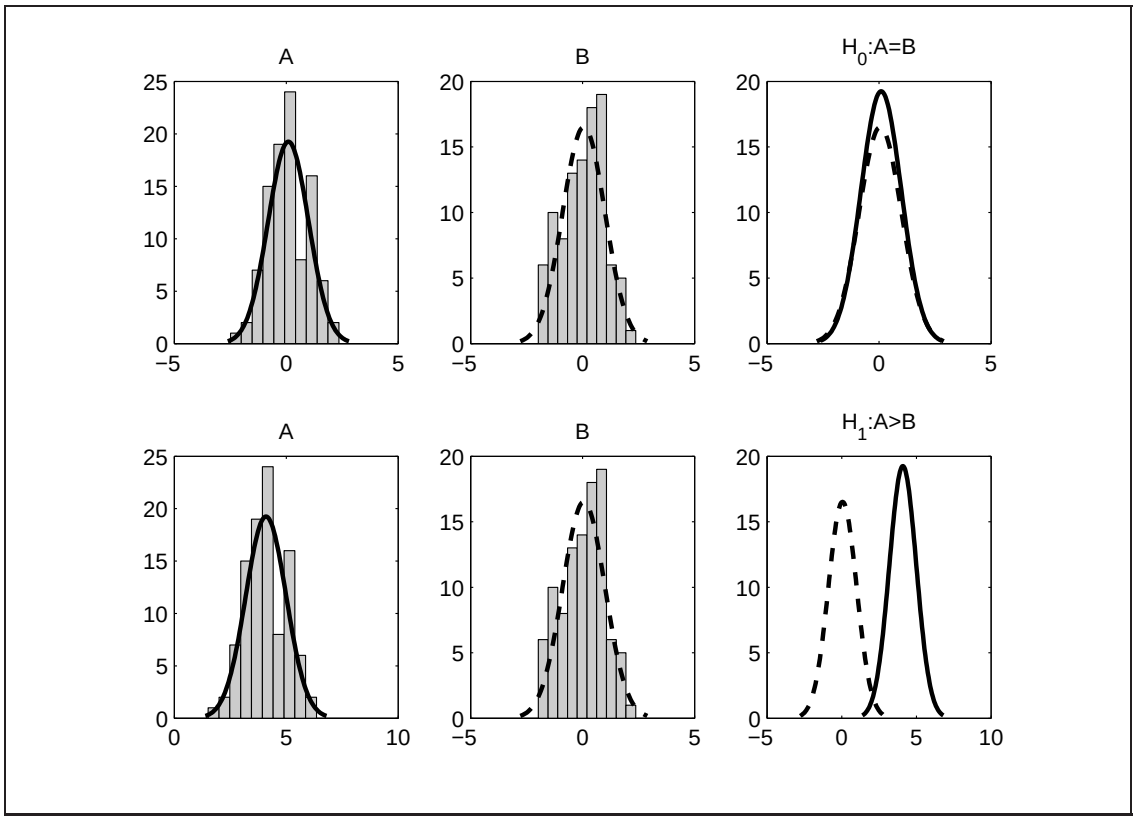

FIGURE 30 Graphical representation of the null-hypothesis accepted (top) or rejected (bottom)

In case of algorithmic performance, the two samples represent the sets of the final values (i.e. the best fitness values found at the end of the allotted budget) obtained by multiple runs of two algorithms under tests, over a fixed problem.

In order to test if one of the two algorithms statistically outperforms the other, or rather they have the same performance, we need to perform a statistic analysis of the final results distribution, i.e. check if the null-hypothesis is accepted or not.

APPENDIX 2.1 Wilcoxon Rank-Sum Test

The Wilcoxon Rank-Sum test, also called Mann-Whitney U test, see Wilcoxon (1945) and Mann and Whitney (1947), is a non-parametric (distribution-free) alternative to the two-sample unpaired t-test, used to check the null-hypothesis on two samples under analysis. Here non-parametric means that no assumption about the distribution of the data is needed, unlike the t-test which assumes that the data are normally distributed. The two samples can be of different length, since only the order of the observations is taken into account. The Rank-Sum test belongs indeed to the family of statistic ranking methods, that is methods in which scores (ranks) are used instead of the actual numerical data in order to perform the analysis of their distribution.

The (one-sided) Wilcoxon Rank-Sum test can be used to check the null-hypothesis $H_0 : A = B$ or the alternative hypothesis $H_1 : A \leqslant B$. the Wilcoxon Rank-Sum test can also be used as a two-sided test, resulting in one of the two conditions $H_0 : A = B$ (null-hypothesis) or $H_1 : A \neq B$ (alternative hypothesis), i.e. it is not clear which sample has the lowest mean. It is important to notice that although the original definition by Wilcoxon (1945) uses means as a measure of location of the two distributions, any measure of "central tendency" of the two groups can be used. For example, Mann and Whitney (1947) use the medians.

The Wilcoxon Rank-Sum test consists of the following. The $n_A + n_B$ observations are combined and ranked, so that rank 1 is assigned to the smallest observation, rank 2 to the 2nd smallest, and so on. In case of two or more observations having the same value (ties), they are assigned the corresponding mean rank value (e.g. if three observations have ordinal ranks $5, 6, 7$ but they have equal values, they are assigned the same rank $(5 + 6 + 7)/3 = 6$). For each sample, the rank-sum r_{sum} is then calculated, and the p-value is calculated as the probability that the total rank-sum of a distribution of the same size is lower than r_{sum}. For small samples size (up to 20), this probability distribution is tabulated, otherwise it can be approximated using the normal distribution. Small values of the p-value cast doubt on the validity of the null-hypothesis, i.e. suggest, under a certain significance level (usually set to 0.05), that the two distributions are separated and the null-hypothesis should be rejected.

For the sake of completeness, we must mention that, analogous to Wilcoxon Rank-Sum test, also the Wilcoxon Signed-Rank test, see Wilcoxon (1945), has been proposed as a statistic analysis technique for comparing performance of algorithms, see García et al. (2008a,b). This method is the non-parametric version of the paired t-test. The main difference between the Wilcoxon Signed-Rank test and the previous one is that the Wilcoxon Rank-Sum assumes the two samples

to be independent, while the Wilcoxon Signed-Rank test is applied to two related samples having the same length, or two sets of repeated observations of a single sample. As such, Wilcoxon Signed-Rank test performs a paired, two-sided signed rank test on the null-hypothesis that data in the vector $A - B$ come from a continuous, symmetric distribution with zero mean, against the alternative that the distribution does not have zero mean. Since this method is paired, meaning that it is based on the pairwise differences between the two samples A and B, in this case the condition $n_a = n_b$ is necessary.

It is worthwhile to note that, although both the Wilcoxon Rank-Sum and Signed-Rank tests can be used to assess if the means of two data-sets A and B differ, i.e if there are significant differences between them, testing the hypothesis of zero mean for $A - B$ is not strictly equivalent to a hypothesis of equal mean for A and B. The choice of which one of the two methods should be used to compare the behavior of two algorithms heavily relies on the assumption of independence between the two sets of final values. It the two data-sets can be considered reasonably independent, the Wilcoxon Rank-Sum is to be preferred since it is more reliable. On the other hand, Wilcoxon Signed-Rank is rather sensible to the precision of the differences, and it cannot be applied if the two data-sets are of different length. As for the t-tests instead, it should be noticed that when the assumptions of the (either paired or unpaired) t-test hold, the Wilcoxon tests are somewhat less reliable and powerful than their equivalent t-test. However, the loss of accuracy is in general quite small. On the other hand, when those assumptions are violated, the Wilcoxon tests can be even more reliable than their equivalent t-test. Moreover, the Wilcoxon tests are much less sensitive to outliers than their equivalent t-test. For these reasons, the Wilcoxon Rank-Sum test seems to be a reasonable choice for comparing the results of two algorithms and analyzing their behavior. All the papers presented in this work make use of the Wilcoxon Rank-Sum test, since no assumption was made on the distributions of the final values of each run and all the runs of each algorithm were considered independent.

APPENDIX 2.2 Holm Procedure

Unlike the Wilcoxon Rank-Sum test, the Holm-Bonferroni procedure, see Holm (1979), is a multiple comparison procedure that can be used to draw some statistically significant conclusions regarding the "global" performance of an algorithm under study compared to other algorithms, as suggested in García et al. (2008a,b). In this context, "global performance" means the performance calculated over a set of test functions, instead of the single problem-specific performance (which is instead considered in the Wilcoxon test).

The Holm procedure consists of the following. Considering the final values of each run of a set of algorithms over a set of test functions, the algorithms are ranked on the basis of their average performance calculated over each problem. More specifically, a score R_i for $i = 1, \cdots, N_A$ (where N_A is the number of algo-

rithms under analysis) is assigned. The score is computed in the following way: for each k-th problem, $k = 1, \cdots, N_{TP}$ (where N_{TP} is the number of test problems in consideration), a score $R_{i,k}$ of N_A is assigned to the algorithm displaying the best performance (i.e. the lowest mean of the final values, for a minimization problem), $N_A - 1$ is assigned to the second best, $N_A - 2$ to the third and so on. The algorithm displaying the worst performance scores 1. For each i-th algorithm, a mean score is calculated averaging the sum of the scores of each problem, according to the following formula:

$$R_i = \frac{\sum_{k=1}^{N_{TP}} R_{i,k}}{N_P}.$$

On the basis of these scores the algorithms are sorted (ranked) in the following way. Within the calculated R_i values, the algorithm under study is taken as a reference algorithm and its mean score is renamed as R_0. The ranks of the remaining $N_A - 1$ algorithms, are sorted from best to worst. Indicating with R_j the sorted ranks of the remaining algorithms, the values z_j, for $j = 1, \cdots, N_A - 1$, are calculated as:

$$z_j = \frac{R_j - R_0}{\sqrt{\frac{N_A(N_A+1)}{6N_{TP}}}}.$$

By means of the z_j values, the corresponding cumulative normal distribution values p_j are calculated. These p_j values are then compared with the corresponding $\delta/(N_A - j)$ where δ is the significance level (usually set to 0.05) of null-hypothesis rejection. As long as the condition $p_j < \delta/(N_A - j)$ holds, the null-hypothesis (that the two algorithms have indistinguishable performances) is rejected, i.e. the reference algorithm outperforms the selected j-th algorithm. This repeated check stops as soon as the above condition fails, meaning that the j-th null-hypothesis is accepted, i.e. the distribution of values of the two algorithms can be considered the same (there is no clear out-performance). At this point, all the remaining hypotheses are also accepted.

ACRONYMS

3SOME	Three Stage Optimal Memetic Exploration
BFO	Bacterial Foraging Optimization
BOA	Bayesian Optimization Algorithm
BMDA	Bivariate Marginal Distribution Algorithm
CcDE	Composed compact Differential Evolution
cDE	compact Differential Evolution
CDF	Cumulative Distribution Function
CEC	Congress on Evolutionary Computation
cGA	compact Genetic Algorithm
CI	Computational Intelligence
CMA-ES	Covariance Matrix Adaptation Evolution Strategies
cODE	compact Opposition-based Differential Evolution
DE	Differential Evolution
DEahcSPX	DE with Adaptive Hill Climbing Simplex Crossover
DEcDE	Disturbed Exploitation compact Differential Evolution
DEGL	DE with Global and Local Neighborhoods
DEPSR	DE with Population Size Reduction
DESFLS	DE with Scale Factor Local Search
EA	Evolutionary Algorithm
ecGA	extended compact Genetic Algorithm
EDA	Estimation of Distribution Algorithm
EGNA	Estimation of Gaussian Networks Algorithm
EMNA	Estimation of Multivariate Normal Algorithm
ENcDE	Ensemble compact Differential Evolution
ES	Evolution Strategy
GA	Genetic Algorithm
GODE	Generalized Opposition-based DE
hBOA	hierarchical Bayesian Optimization Algorithm
jDE	Self Adaptive Control Parameters in Differential Evolution
LS	Local Search
MA	Memetic Algorithm
MC	Memetic Computing

MDL	Minimum Description Length
MPM	Marginal Product Model
NAcDE	Noise Analysis compact Differential Evolution
ne	non-persistent elitism
OBDE	Opposition Based Differential Evolution
OBL	Opposition Based Learning
PBIL	Population Based Incremental Learning
PDF	Probability Distribution Function
pe	persistent elitism
PMBGA	Probabilistic Model Building Genetic Algorithm
PSO	Particle Swarm Optimization
PSR	Population Size Reduction
PV	Probability (or Perturbation) Vector
QEA	Quantum-inspired Evolutionary Algorithms
rcGA	real-valued compact Genetic Algorithm
SADE	Self-Adaptive Differential Evolution
ScDE	Supervised compact Differential Evolution
SFcDE-PSR	Super-Fit cDE with Population Size Reduction
SI	Swarm Intelligence
SPX	Simplex Crossover
TDE	Trigonometric Differential Evolution
UMDA	Univariate Marginal Distribution Algorithm

REFERENCES

Ahn, C. W. & Ramakrishna, R. S. 2003. Elitism based compact genetic algorithms. IEEE Transactions on Evolutionary Computation 7 (4), 367–385.

Ali, M. M. & Fatti, L. P. 2006. A differential free point generation scheme in the differential evolution algorithm. Journal of Global Optimization 35 (4), 551–572.

Ali, M. M. & Törn, A. 2004. Population set-based global optimization algorithms: Some modifications and numerical studies. Computer Operational Research 31 (10), 1703–1725.

Aporntewan, C. & Chongstitvatana, P. 2001. A hardware implementation of the compact genetic algorithm. In Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 1, 624–629.

Arnold, D. V. & Beyer, H.-G. 2006. A general noise model and its effects on evolution strategy performance. IEEE Trans. Evolutionary Computation 10 (4), 380–391.

Baker, J. E. 1987. Reducing bias and inefficiency in the selection algorithm. In Proceedings of the International Conference on Genetic Algorithms. Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, 14–21.

Baluja, S. 1994. Population-Based Incremental Learning: A Method for Integrating Genetic Search Based Function Optimization and Competitive Learning.

Baraglia, R., Hidalgo, J. & Perego, R. 2001. A hybrid heuristic for the traveling salesman problem. IEEE Transactions on Evolutionary Computation 5 (6), 613–622.

Bazaraa, M. S., Sherali, H. D. & Shetty, C. M. 2006. Nonlinear Programming: Theory And Algorithms. Wiley-Interscience.

Beyer, H.-G. & Sendhoff, B. 2006. Functions with noise-induced multimodality: a test for evolutionary robust optimization-properties and performance analysis. IEEE Trans. Evolutionary Computation 10 (5), 507–526.

Branke, J. & Schmidt, C. 2003. Selection in the presence of noise. In GECCO, 766–777.

Brest, J., Bošković, B., Greiner, S., Žumer, V. & Maučec, M. S. 2007. Performance comparison of self-adaptive and adaptive differential evolution algorithms. Soft Computing 11 (7), 617–629.

Brest, J., Bošković, B., Zamuda, A. & Žumer, V. 2008. An analysis of the control parameters'adaptation in DE. In U. K. Chakraborty (Ed.) Advances in Differential Evolution, Vol. 143. Springer. Studies in Computational Intelligence, 89–110.

Brest, J., Greiner, S., Bošković, B., Mernik, M. & Žumer, V. 2006. Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation 10 (6), 646–657.

Brest, J. & Maučec, M. S. 2008. Population size reduction for the differential evolution algorithm. Applied Intelligence 29 (3), 228–247.

Brest, J., Žumer, V. & Maucec, M. 2006. Self-adaptive differential evolution algorithm in constrained real-parameter optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, 215–222.

Caponio, A., Cascella, G. L., Neri, F., Salvatore, N. & Sumner, M. 2007. A fast adaptive memetic algorithm for on-line and off-line control design of PMSM drives. IEEE Transactions on System Man and Cybernetics-part B 37 (1), 28–41.

Caponio, A., Neri, F. & Tirronen, V. 2009. Super-fit control adaptation in memetic differential evolution frameworks. Soft Computing-A Fusion of Foundations, Methodologies and Applications 13 (8), 811–831.

Chakraborty, U. K., Das, S. & Konar, A. 2006. Differential evolution with local neighborhood. In Proceedings of the IEEE Congress on Evolutionary Computation, 2042–2049.

Chen, H., Zhu, Y. & Hu, K. 2008. Self-adaptation in Bacterial Foraging Optimization algorithm. In Proc. 3rd International Conference on Intelligent System and Knowledge Engineering ISKE 2008, Vol. 1, 1026–1031.

Chen, H., Zhu, Y. & Hu, K. 2011. Adaptive bacterial foraging optimization. Abstract and Applied Analysis 2011.

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

Cody, W. J. 1969. Rational chebyshev approximations for the error function. Mathematics of Computation 23 (107), 631–637.

Das, S., Abraham, A., Chakraborty, U. K. & Konar, A. 2009a. Differential evolution with a neighborhood-based mutation operator. IEEE Transactions on Evolutionary Computation 13, 526–553.

Das, S., Biswas, A., Dasgupta, S. & Abraham, A. 2009b. Bacterial foraging optimization algorithm: Theoretical foundations, analysis, and applications. In Foundations of Computational Intelligence (3), 23–55.

Das, S., Konar, A. & Chakraborty, U. K. 2005a. Improved differential evolution algorithms for handling noisy optimization problems. In Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 2, 1691–1698.

Das, S., Konar, A. & Chakraborty, U. K. 2005b. Two improved differential evolution schemes for faster global search. In Proceedings of the 2005 conference on Genetic and evolutionary computation. ACM, 991–998.

Das, S., Konar, A. & Chakraborty, U. K. 2007. Annealed differential evolution. In Proceedings of the IEEE Congress on Evolutionary Computation, 1926–1933.

Das, S. & Konar, A. 2005. An improved differential evolution scheme for noisy optimization problems. In Pattern recognition and machine intelligence, Vol. 3776. Springer. Lecture Notes in Computer Science, 417–421.

Dasgupta, S., Das, S., Abraham, A. & Biswas, A. 2009. Adaptive computational chemotaxis in bacterial foraging optimization: an analysis. Trans. Evol. Comp 13, 919–941.

Dasgupta, S., Das, S., Biswas, A. & Abraham, A. 2010. Automatic circle detection on digital images with an adaptive bacterial foraging algorithm. Soft Comput. 14 (11), 1151–1164.

De Bonet, J. S., Isbell, C. L. & Viola, P. 1997. Mimic: Finding optima by estimating probability densities. In Advances in Neural Information Processing Systems, Vol. 9.

Di Pietro, A., While, L. & Barone, L. 2004. Applying evolutionary algorithms to problems with noisy, time-consuming fitness functions. In Proceedings of the 2004 IEEE Congress on Evolutionary Computation. IEEE Press, 1254–1261.

Eiben, A. E. & Smith, J. E. 2003. Introduction to Evolutionary Computation. Berlin: Springer Verlag, 175–188.

Eshelman, L. J. 1990. The chc adaptive search algorithm: How to have safe search when engaging in nontraditional genetic recombination. In G. J. E. Rawlins (Ed.) Foundations of Genetic Algorithms I. San Mateo CA: Morgan Kaufmann, 265–283.

Fan, H.-Y. & Lampinen, J. 2003. A trigonometric mutation operation to differential evolution. Journal of Global Optimization 27 (1), 105–129.

Feoktistov, V. 2006. Differential Evolution in Search of Solutions. Springer.

Fogel, L. J., Owens, A. J. & Walsh, M. J. 1966. Artificial Intelligence through Simulated Evolution. New York, USA: John Wiley.

Gallagher, J. C., Vigraham, S. & Kramer, G. 2004. A family of compact genetic algorithms for intrinsic evolvable hardware. IEEE Transactions Evolutionary Computation 8 (2), 111–126.

García, S., Fernández, A., Luengo, J. & Herrera, F. 2008a. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability. Soft Computing 13 (10), 959–977.

García, S., Molina, D., Lozano, M. & Herrera, F. 2008b. A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization. Journal of Heuristics 15 (6), 617–644.

Gautschi, W. 1972. Error function and fresnel integrals. In M. Abramowitz & I. A. Stegun (Eds.) Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables, 297–309.

Glover, F. 1989a. Tabu search - part II. ORSA Journal on Computing 2, 4–32.

Glover, F. 1989b. Tabu search - part I. ORSA Journal on Computing 1, 190–206.

Goldberg, D. E. 1989. Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA, USA: Addison-Wesley Publishing Co.

Griewank, A. & Walther, A. 2008. Evaluating Derivatives: Principles and Techniques of Algorithmic Differentiation (2nd edition). Philadelphia: SIAM.

Gross, D. & Harris, C. M. 1985. Fundamentals of Queueing Theory. Wiley, NY.

Gämperle, R., Müller, S. D. & Koumoutsakos, P. 2002. A parameter study for differential evolution. In Proceedings of the Conference in Neural Networks and Applications (NNA), Fuzzy Sets and Fuzzy Systems (FSFS) and Evolutionary Computation (EC). WSEAS, 293–298.

Hansen, N. & Ostermeier, A. 2001. Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9 (2), 159–195.

Harik, G. 1999. Linkage Learning via Probabilistic Modeling in the ECGA.

Harik, G. R., Lobo, F. G. & Goldberg, D. E. 1999. The compact genetic algorithm. IEEE Transactions on Evolutionary Computation 3 (4), 287–297.

Harik, G. R., Lobo, F. G. & Sastry, K. 2006. Linkage learning via probabilistic modeling in the extended compact genetic algorithm (ECGA). In M. Pelikan, K. Sastry & E. Cantú-Paz (Eds.) Scalable Optimization via Probabilistic Modeling, Vol. 33. Springer. Studies in Computational intelligence, 39–61.

Hart, W. E., Krasnogor, N. & Smith, J. E. 2004. Memetic evolutionary algorithms. In W. E. Hart, N. Krasnogor & J. E. Smith (Eds.) Recent Advances in Memetic Algorithms. Berlin, Germany: Springer, 3–27.

Hendtlass, T. 2001. A combined swarm differential evolution algorithm for optimization problems. In Lecture Notes in Computer Science, Springer-Verlag, Vol. 2070, 11–18.

Herrera, F., Lozano, M. & Sánchez, A. M. 2003. A taxonomy for the crossover operator for real-coded genetic algorithms: An experimental study. International Journal of Intelligent Systems 18, 309–338.

Holland, J. 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press.

Holm, S. 1979. A simple sequentially rejective multiple test procedure. Scandinavian Journal of Statistics 6 (2), 65–70.

Hooke, R. & Jeeves, T. A. 1961. Direct search solution of numerical and statistical problems. Journal of the ACM 8, 212–229.

Houck, C., Joines, J., Kay, M. & Wilson, J. 1997. Empirical investigation of the benefits of partial lamarckianism. Evolutionary Computation 5 (1), 31–60.

Hu, Z.-B., Su, Q.-H., Xiong, S.-W. & Hu, F.-G. 2008. Self-adaptive hybrid differential evolution with simulated annealing algorithm for numerical optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, 1189–1194.

Ishibuchi, H., Hitotsuyanagi, Y. & Nojima, Y. 2007. An empirical study on the specification of the local search application probability in multiobjective memetic algorithms. In Proc. of the IEEE Congress on Evolutionary Computation, 2788–2795.

Ishibuchi, H., Yoshida, T. & Murata, T. 2003. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flow shop scheduling. IEEE Transactions on Evolutionary Computation 7, 204–223.

Jewajinda, Y. & Chongstitvatana, P. 2008. Cellular compact genetic algorithm for evolvable hardware. In Proceedings of the International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Vol. 1, 1–4.

Jin, Y. & Branke, J. 2005. Evolutionary optimization in uncertain environments-a survey. Evolutionary Computation, IEEE Transactions on 9 (3), 303–317.

Jones, T. 1995. Crossover, macromutation, and population-based search. In L. Eshelman (Ed.) Sixth International Conference on Genetic Algorithms. San Mateo CA: Morgan Kaufmann, 73–80.

Kennedy, J. & Eberhart, R. 1995. Particle swarm optimization. In Proceedings of IEEE International Conference on Neural Networks, 1942–1948.

Kiefer, J. 1953. Sequential minimax search for a maximum. Proceedings of the American Mathematical Society 4, 502–506.

Kirkpatrick, S., Gelatt, C. D. J. & Vecchi, M. P. 1983. Optimization by simulated annealing. Science 220, 671–680.

Krasnogor, N. 2004. Toward robust memetic algorithms. In W. E. Hart, N. Krasnogor & J. E. Smith (Eds.) Recent Advances in Memetic Algorithms. Berlin, Germany: Springer. Studies in Fuzzines and Soft Computing, 185–207.

Krink, T., Filipič, B. & Fogel, G. B. 2004. Noisy optimization problems - a particular challenge for differential evolution? In Proceedings of the IEEE Congress on Evolutionary Computation, 332–339.

Lampinen, J. & Zelinka, I. 2000. On stagnation of the differential evolution algorithm. In P. Oŝmera (Ed.) Proceedings of 6th International Mendel Conference on Soft Computing, 76–83.

Larrañaga, P. & Lozano, J. A. 2001. Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Kluwer.

Liu, J. & Lampinen, J. 2002a. A fuzzy adaptive differential evolution algorithm. In Proceedings of the 17th IEEE region 10 international conference on computer, communications, control and power engineering, Vol. I, 606–611.

Liu, J. & Lampinen, J. 2002b. On setting the control parameter of the differential evolution algorithm. In Proceedings of the 8th international Mendel conference on soft computing, 11–18.

Lozano, J. A., Larrañaga, P., Inza, I. & Bengoetxea, E. 2006. Towards a New Evolutionary Computation. Advances on Estimation of Distribution Algorithms, Vol. 192. Springer-Verlag. Studies in Fuzziness and Soft Computing.

Lozano, M. & García-Martínez, C. 2010. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report. Computers & Operations Research 37 (3), 481–497.

Lozano, M., Herrera, F., Krasnogor, N. & Molina, D. 2004. Real-coded memetic algorithms with crossover hill-climbing. Evolutionary Computation, Special Issue on Memetic Algorithms 12 (3), 273–302.

Mallipeddi, R., Suganthan, P., Pan, Q. & Tasgetiren, M. 2011. Differential evolution algorithm with ensemble of parameters and mutation strategies. Applied Soft Computing 11 (2), 1679–1696. The Impact of Soft Computing for the Progress of Artificial Intelligence.

Mallipeddi, R. & Suganthan, P. 2008. Empirical study on the effect of population size on differential evolution algorithm. In Proceedings of the IEEE Congress on Evolutionary Computation, 3663–3670.

Mann, H. B. & Whitney, D. R. 1947. On a Test of Whether one of Two Random Variables is Stochastically Larger than the Other. The Annals of Mathematical Statistics 18 (1), 50–60.

Michalewicz, Z. 1996. Genetic algorithms + data structures = evolution programs (3rd ed.). London, UK: Springer-Verlag.

Mininno, E., Cupertino, F. & Naso, D. 2008. Real-valued compact genetic algorithms for embedded microcontroller optimization. IEEE Transactions on Evolutionary Computation 12 (2), 203–219.

Mininno, E., Neri, F., Cupertino, F. & Naso, D. 2011. Compact differential evolution. IEEE Transactions on Evolutionary Computation 15 (1), 32–54.

Molina, D. 2005. Adaptive local search parameters for real-coded memetic algorithm. In Proceedings of the IEEE Congress on Evolutionary Computation, 888–895.

Mühlenbein, H., Schomisch, M. & Born, J. 1991. The parallel genetic algorithm as function optimizer. Parallel Computing 17 (6–7), 619–632.

Mühlenbein, H., Bendisch, J. & Voigt, H.-M. 1996. From recombination of genes to the estimation of distributions ii. continuous parameters. In PPSN, 188–197.

Mühlenbein, H. & Paass, G. 1996. From recombination of genes to the estimation of distributions i. binary parameters. In PPSN, 178–187.

Nearchou, A. C. & Omirou, S. L. 2006. Differential evolution for sequencing and scheduling optimization. Journal of Heuristics 12 (6), 395–411.

Nelder, A. & Mead, R. 1965. A simplex method for function optimization. Computation Journal Vol 7, 308–313.

Neri, F., Cotta, C. & Moscato, P. 2011. Handbook of Memetic Algorithms, Vol. 379. Springer. Studies in Computational Intelligence.

Neri, F., Mininno, E. & Kärkkäinen, T. 2010. Noise analysis compact genetic algorithm. In Applications of Evolutionary Computation, Vol. 6024. Springer. Lecture Notes in Computer Science, 602–611.

Neri, F. & Mäkinen, R. A. E. 2007. Hierarchical evolutionary algorithms and noise compensation via adaptation. In S. Yang, Y. S. Ong & Y. Jin (Eds.) Evolutionary Computation in Dynamic and Uncertain Environments. Springer. Studies in Computational Intelligence, 345–369.

Neri, F. & Tirronen, V. 2008. On memetic differential evolution frameworks: a study of advantages and limitations in hybridization. In Proceedings of the IEEE World Congress on Computational Intelligence, 2135–2142.

Neri, F. & Tirronen, V. 2009. Scale factor local search in differential evolution. Memetic Computing 1 (2), 153–171.

Neri, F. & Tirronen, V. 2010. Recent advances in differential evolution: A review and experimental analysis. Artificial Intelligence Review 33 (1–2), 61–106.

Neri, F., Toivanen, J. I., Cascella, G. L. & Ong, Y. S. 2007a. An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE/ACM Transactions on Computational Biology and Bioinformatics 4 (2), 264–278.

Neri, F., Toivanen, J. I. & Mäkinen, R. A. E. 2007b. An adaptive evolutionary algorithm with intelligent mutation local searchers for designing multidrug therapies for HIV. Applied Intelligence 27 (3), 219–235.

Neri, F., Tirronen, V., Kärkkäinen, T. & Rossi, T. 2007c. Fitness diversity based adaptation in multimeme algorithms: A comparative study. In Proceedings of the IEEE Congress on Evolutionary Computation, 2374–2381.

Neri, F., Tirronen, V. & Kärkkäinen, T. 2009. Enhancing differential evolution frameworks by scale factor local search - part II. In Proceedings of the IEEE Congress on Evolutionary Computation. Piscataway, NJ, USA: IEEE Press. CEC'09, 118–125.

Noman, N. & Iba, H. 2008. Accelerating differential evolution using an adaptive local search. IEEE Transactions on Evolutionary Computation 12 (1), 107–125.

Ong, Y. S. & Keane, A. J. 2004. Meta-lamarkian learning in memetic algorithms. IEEE Transactions on Evolutionary Computation 8 (2), 99–110.

Ong, Y.-S., Lim, M.-H. & Chen, X. 2010. Memetic computation-past, present and future. IEEE Computational Intelligence Magazine 5 (2), 24–31.

Passino, K. M. 2002. Biomimicry of bacterial foraging for distributed optimization and control. IEEE Control Systems Magazine 22 (3), 52–67.

Paul, T. K. & Iba, H. 2003. Real-coded estimation of distribution algorithm. In Proceedings of the Fifth Metaheuristics International Conference (MIC2003), 60–1–60–6.

Pelikan, M. & Mühlenbein, H. 1999. The bivariate marginal distribution algorithm. In R. Roy, T. Furuhashi & P. K. Chawdhry (Eds.) Advances in Soft Computing - Engineering Design and Manufacturing. London: Springer-Verlag, 521–535.

Pelikan, M., Goldberg, D. E. & Cantú-Paz, E. 2000. Linkage problem, distribution estimation, and bayesian networks. Evol. Comput. 8, 311–340.

Pelikan, M., Goldberg, D. E. & Lobo, F. G. 2002. A survey of optimization by building and using probabilistic models. Comput. Optim. Appl. 21, 5–20.

Pelikan, M. 2005. Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms (1st edition). Springer. Studies in Fuzziness and Soft Computing.

Plagianakos, V. P., Tasoulis, D. K. & Vrahatis, M. N. 2008. A review of major application areas of differential evolution. In U. K. Chakraborty (Ed.) Advances in Differential Evolution, Vol. 143. Springer. Studies in Computational Intelligence, 197–238.

Platel, M. D., Schliebs, S. & Kasabov, N. 2009. Quantum-inspired evolutionary algorithm: a multimodel eda. Trans. Evol. Comp 13, 1218–1232.

Price, K. & Storn, R. 1997. Differential evolution: A simple evolution strategy for fast optimization. Dr. Dobb's J. Software Tools 22 (4), 18–24.

Price, K. V. 1999. Mechanical engineering design optimization by differential evolution. In D. Corne, M. Dorigo & F. Glover (Eds.) New Ideas in Optimization. McGraw-Hill, 293–298.

Price, K. V., Storn, R. & Lampinen, J. 2005. Differential Evolution: A Practical Approach to Global Optimization. Springer.

Qin, A. K., Huang, V. L. & Suganthan, P. 2009. Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Transactions on Evolutionary Computation 13, 398–417.

Qin, A. K. & Suganthan, P. 2005. Self-adaptive differential evolution algorithm for numerical optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 2, 1785–1791.

Qing, A. 2008. A study on base vector for differential evolution. In Proceedings of the IEEE Congress on Evolutionary Computation, 550–556.

Rahnamayan, S., Tizhoosh, H. & Salama, M. M. A. 2006a. Opposition-based differential evolution for optimization of noisy problems. In Proceedings of the IEEE Congress on Evolutionary Computation, 1865–1872.

Rahnamayan, S., Tizhoosh, H. R. & Salama, M. M. 2006b. Opposition-based differential evolution algorithms. In Proceedings of the IEEE Congress on Evolutionary Computation, 2010–2017.

Rahnamayan, S., Tizhoosh, H. R. & Salama, M. M. 2008. Opposition-based differential evolution. IEEE Transactions on Evolutionary Computation 12 (1), 64–79.

Rahnamayan, S. & Wang, G. G. 2008. Solving large scale optimization problems by opposition-based differential evolution (ode). WSEAS Transactions on Computers 7 (10), 1792–1804.

Rastegar, R. & Hariri, A. 2006. A step forward in studying the compact genetic algorithm. Evolutionary Computation 14 (3), 277–289.

Rechenberg, I. 1971. Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Technical University of Berlin. Ph. D. Thesis.

Rosenbrock, H. H. 1960. An automatic method for finding the greatest or least value of a function. The Computer Journal 3 (3), 175–184.

Rudolph, G. 2001. Self-adaptive mutations may lead to premature convergence. IEEE Transactions on Evolutionary Computation 5 (4), 410–414.

Russell, S. J. & Norvig, P. 2003. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall.

Rönkkönen, J., Kukkonen, S. & Price, K. V. 2005. Real-parameter optimization with differential evolution. In Proceedings of IEEE International Conference on Evolutionary Computation, Vol. 1, 506–513.

Sastry, K. & Xiao, G. 2001. Cluster optimization using extended compact genetic algorithm.

Sastry, K., Goldberg, D. E. & Johnson, D. D. 2007. Scalability of a hybrid extended compact genetic algorithm for ground state optimization of clusters. Materials and Manufacturing Processes 22 (5), 570–576.

Sastry, K. & Goldberg, D. E. 2000. On Extended Compact Gentic Algorithm.

Schwefel, H.-P. 1965. Kybernetische Evolution als Strategie der experimentellen Forschung in der Strömungstechnik. Technical University of Berlin, Hermann Föttinger–Institute for Fluid Dynamics. Ph. D. Thesis.

Seront, G. & Bersini, H. 2000. A new ga-local search hybrid for continuous optimization based on multi-level single linkage clustering. In GECCO, 90–95.

Shi, Y. & Eberhart, R. 1998. A modified particle swarm optimizer. In IEEE World Congress on Computational Intelligence, 69–73.

Sing, T. N., Teo, J. & Hijazi, M. H. A. 2007. Empirical testing on 3-parents differential evolution (3PDE) for unconstrained function optimization. In Proceedings of the IEEE Congress on Evolutionary Computation, 2259–2266.

Spall, J. C. 2003. Introduction to Stochastic Search and Optimization (1st edition). New York, NY, USA: John Wiley & Sons, Inc.

Storn, R. & Price, K. 1997. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11, 341–359.

Storn, R. & Price, K. 1995. Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces. ICSI.

Suganthan, P., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A. & Tiwari, S. 2005. Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization. Nanyang Technological University and KanGAL.

Tan, K., Chiam, S., Mamun, A. & Goh, C. 2009. Balancing exploration and exploitation with adaptive variation for evolutionary multi-objective optimization. European Journal of Operational Research 197, 701–713.

Tang, K., Yao, X., Suganthan, P., MacNish, C., Chen, Y. P., Chen, C. M. & Yang, Z. 2007. Benchmark Functions for the CEC 2008 Special Session and Competition on Large Scale Global Optimization. Nature Inspired Computation and Applications Laboratory, USTC, China.

Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P. & Vrahatis, M. N. 2004. Parallel differential evolution. In Proceedings of the IEEE Congress on Evolutionary Computation, 2023–2029.

Teng, N. S., Teo, J. & Hijazi, M. H. A. 2009. Self-adaptive population sizing for a tune-free differential evolution. Soft Computing – A Fusion of Foundations, Methodologies and Applications 13 (7), 709–724.

Teo, J. 2006. Exploring dynamic self-adaptive populations in differential evolution. Soft Computing – A Fusion of Foundations, Methodologies and Applications 10 (8), 673–686.

Teo, J. 2005. Differential evolution with self-adaptive populations. In Knowledge-Based Intelligent Information and Engineering Systems, Vol. 3681. Springer. Lecture Notes in Computer Science, 1284–1290.

Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K. & Rossi, T. 2008. An enhanced memetic differential evolution in filter design for defect detection in paper production. Evolutionary Computation 16, 529–555.

Tirronen, V. & Neri, F. 2009. Differential evolution with fitness diversity self-adaptation. In R. Chiong (Ed.) Nature-Inspired Algorithms for Optimisation, Vol. 193. Springer. Studies in Computational Intelligence, 199–234.

Tirronen, V., Neri, F. & Rossi, T. 2009. Enhancing differential evolution frameworks by scale factor local search - part I. In IEEE Congress on Evolutionary Computation, 94–101.

Toivanen, J. I., Mäkinen, R. A. E., Järvenpää, S., Ylä-Oijala, P. & Rahola, J. 2009. Electromagnetic sensitivity analysis and shape optimization using method of moments and automatic differentiation. IEEE Transactions on Antennas & Propagation 57 (1), 168–175.

Toivanen, J. I. & Mäkinen, R. A. E. 2011. Implementation of sparse forward mode automatic differentiation with application to electromagnetic shape optimization. Optimization Methods and Software 26 (4-5), 601–616.

Tsutsui, S., Yamamura, M. & Higuchi, T. 1999. Multi-parent recombination with simplex crossover in real coded genetic algorithms. In Proceedings of the Genetic Evol. Comput. Conf. (GECCO), 657–664.

Vesterstrøm, J. & Thomsen, R. 2004. A comparative study of differential evolution particle swarm optimization and evolutionary algorithms on numerical benchmark problems. In Proceedings of the IEEE Congress on Evolutionary Computation, Vol. 3, 1980–1987.

Wang, H., Wu, Z. & Rahnamayan, S. 2011. Enhanced opposition-based differential evolution for solving high-dimensional continuous optimization problems. Soft Computing-A Fusion of Foundations, Methodologies and Applications N/A (N/A). to appear.

Weber, M., Tirronen, V. & Neri, F. 2010. Scale factor inheritance mechanism in distributed differential evolution. Soft Computing - A Fusion of Foundations, Methodologies and Applications 14 (11), 1187–1207.

Wilcoxon, F. 1945. Individual comparisons by ranking methods. Biometrics Bulletin 1 (6), 80–83.

Wolpert, D. & Macready, W. 1997. No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation 1 (1), 67–82.

Xu, X., Li, Y., Fang, S., Wu, Y. & Wang, F. 2008. A novel differential evolution scheme combined with particle swarm intelligence. In Proceedings of the IEEE Congress on Evolutionary Computation, 1057–1062.

Yao, X., Liu, Y. & Lin, G. 1999. Evolutionary programming made faster. IEEE Transactions on Evolutionary Computation 3, 82–102.

Yuan, B. & Gallagher, M. 2005. Experimental results for the special session on real-parameter optimization at CEC 2005: A simple, continuous EDA. In Proceedings of the IEEE Conference of Evolutionary Computation, 1792–1799.

Zaharie, D. 2003. Control of population diversity and adaptation in differential evolution algorithms. In D.Matousek & P. Osmera (Eds.) Proceedings of MENDEL International Conference on Soft Computing, 41–46.

Zamuda, A., Brest, J., Bošković, B. & Žumer, V. 2007. Differential evolution for multiobjective optimization with self adaptation. In Proceedings of the IEEE Congress on Evolutionary Computation, 3617–3624.

Zamuda, A., Brest, J., Bošković, B. & Žumer, V. 2008. High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction. In Proceedings of the IEEE World Congress on Computational Intelligence, 2032–2039.

Zhang, Q., Sun, J., Tsang, E., Ford, J. & others 2004. Hybrid estimation of distribution algorithm for global optimization. Engineering Computations 21 (1), 91–107.

Zhenyu, G., Bo, C., Min, Y. & Binggang, C. 2006. Self-adaptive chaos differential evolution. In Advances in Natural Computation, Vol. 4221. Springer. Lecture Notes in Computer Science, 972–975.

Zielinski, K., Wang, X. & Laur, R. 2008. Comparison of adaptive approaches for differential evolution. In Proceedings of the 10th international conference on Parallel Problem Solving from Nature: PPSN X. Berlin, Heidelberg: Springer-Verlag, 641–650.

Zielinski, K., Weitkemper, P., Laur, R. & Kammeyer, K.-D. 2006. Parameter study for differential evolution using a power allocation problem including interference cancellation. In Proceedings of the IEEE Congress on Evolutionary Computation, 1857–1864.

ORIGINAL PAPERS

PI

DISTURBED EXPLOITATION COMPACT DIFFERENTIAL EVOLUTION FOR LIMITED MEMORY OPTIMIZATION PROBLEMS

by

Ferrante Neri, Giovanni Iacca and Ernesto Mininno 2011

In Information Sciences, volume 181 (2011), issue 12, pages 2469-2487

PII

COMPOSED COMPACT DIFFERENTIAL EVOLUTION

by

Giovanni Iacca, Ernesto Mininno and Ferrante Neri 2011

In Evolutionary Intelligence, volume 4, number 1, pages 17-29

PIII

NOISE ANALYSIS COMPACT DIFFERENTIAL EVOLUTION

by

Giovanni Iacca, Ferrante Neri and Ernesto Mininno 2011

In International Journal of Systems Science, to appear

PIV

GLOBAL SUPERVISION FOR COMPACT DIFFERENTIAL EVOLUTION

by

Giovanni Iacca, Rammohan Mallipeddi, Ernesto Mininno, Ferrante Neri and Ponnuthurai Nagaratnam Suganthan 2011

In 2011 IEEE Symposium on Differential Evolution Proceedings, pages 1-8

PV

SUPER-FIT AND POPULATION SIZE REDUCTION IN COMPACT DIFFERENTIAL EVOLUTION

by

Giovanni Iacca, Rammohan Mallipeddi, Ernesto Mininno, Ferrante Neri and Ponnuthurai Nagaratnam Suganthan 2011

In 2011 IEEE Workshop on Memetic Computing Proceedings, pages 1-8

PVI

OPPOSITION-BASED LEARNING IN COMPACT DIFFERENTIAL EVOLUTION

by

Giovanni Iacca, Ferrante Neri and Ernesto Mininno 2011

In Application of Evolutionary Computation, volume 6624/2011 of Lecture Notes in Computer Science, pages 264-273

PVII

ENSEMBLE STRATEGIES IN COMPACT DIFFERENTIAL EVOLUTION

by

Rammohan Mallipeddi, Giovanni Iacca, Ponnuthurai Nagaratnam Suganthan, Ferrante Neri and Ernesto Mininno 2011

In 2011 IEEE Congress on Evolutionary Computation Proceedings, pages 1972 - 1977

PVIII

OCKHAM'S RAZOR IN MEMETIC COMPUTING: THREE STAGE OPTIMAL MEMETIC EXPLORATION

by

Giovanni Iacca, Ferrante Neri, Ernesto Mininno, Yew-Soon Ong and Meng-Hiot Lim 2011

In Information Sciences, accepted