JYVÄSKYLÄ STUDIES IN COMPUTING
243

Fabio Caraffini

Algorithmic Issues in Computational Intelligence Optimization

From Design to Implementation, from Implementation to Design

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 243

Fabio Caraffini

Algorithmic Issues in Computational Intelligence Optimization

From Design to Implementation, from Implementation to Design

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen Alfa-salissa
syyskuun 17. päivänä 2016 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Agora, Alfa hall, on September 17, 2016 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2016

Algorithmic Issues in Computational Intelligence Optimization

From Design to Implementation, from Implementation to Design

JYVÄSKYLÄ STUDIES IN COMPUTING 243

Fabio Caraffini

Algorithmic Issues in Computational Intelligence Optimization

From Design to Implementation, from Implementation to Design

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2016

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

URN:ISBN:978-951-39-6742-0
ISBN 978-951-39-6742-0 (PDF)

ISBN 978-951-39-6741-3 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2016

ABSTRACT

Caraffini, Fabio
Algorithmic issues in Computational Intelligence Optimization: from design to implementation, from implementation to design
Jyväskylä: University of Jyväskylä, 2016, 94 p.(+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 243)
ISBN 978-951-39-6741-3 (nid.)
ISBN 978-951-39-6742-0 (PDF)
Finnish summary
Diss.

The vertiginous technological growth of the last decades has generated a variety of powerful and complex systems. By embedding within modern hardware devices sophisticated software, they allow the solution of complicated tasks. As side effect, the availability of these heterogeneous technologies results into new difficult optimization problems to be faced by researchers in the field. In order to overcome the most common algorithmic issues, occurring in such a variety of possible scenarios, this research has gone through cherry-picked case-studies.

A first research study moved *from implementation to design* considerations. Implementation limitations, such as memory constraints and real-time requirements, inevitably plague the algorithmic design. Such limitations are typical of embedded systems. In this light, a fast and memory-saving "compact" algorithm was designed to be used within microcontrollers. Three robotic applications were subsequently addressed by means of selected single-solution approaches and the proposed compact algorithm. A new memetic computing approach using a micro-population was also designed to tackle large scale problems.

In a second moment, the opposite approach, *from design to implementation*, was employed. As the benefit of metaheuristic optimization is the capability of tackling black-box systems, 6 novel general-purpose optimizers were designed according to different working principles. Their validity was thoroughly tested by means of popular benchmark suites.

Finally, a theoretical study concludes this piece of research. The dynamic behaviour of population-based optimization algorithms, such as Genetic Algorithm and Particle Swarm Optimization, was observed. Their general-purpose nature questioned. The presence of an intrinsic structural bias was graphically displayed and rigorously formalized. It was shown that the bias prevent them from equally exploring all the areas of the search space, with a particularly deleterious strength in presence of a large population size.

Keywords: hyper-heuristics, memetic computing, differential evolution, compact algorithms, single-solution algorithms, local search, structural bias.

Author

Dr. Fabio Caraffini
Department of Mathematical Information Technology
Faculty of Information Technology
University of Jyväskylä
Finland

Supervisors

Prof. Ferrante Neri

Dr. Ernesto Mininno

Prof. Tuomo Rossi

Prof. Raino Mäkinen

Department of Mathematical Information Technology
Faculty of Information Technology
University of Jyväskylä
Finland

Reviewers

Prof. Janez Brest
Faculty of Electrical Engineering and Computer Science
University of Maribor
Slovenia

Dr. Raymond Chiong
School of Design Communication and IT
University of Newcastle
Australia

Opponent

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

PREFACE

Computational Intelligence Optimization is a relatively young discipline, originated with the coding of mathematical algorithms within the first computers in the late '60s and early '70s. It has quickly evolved, and is now studied under several heterogeneous aspects and sub-fields. Nowadays, regardless of the specific nature of the optimization problem, it is a quite common and consolidated practice to employ Artificial Intelligence (AI) techniques (from which the name) in order to codify robust, versatile, and above all smart optimizers. This evolution has also been allowed by the vertiginous technological progress in producing a variety of silicon-based devices. The digitalization and execution of complex routines, not only in the most sophisticated engineering applications, but especially in addressing everyday life tasks, made us able to perform ours activities faster and better. On the other hand, the complexity in designing more ad more efficient systems has been increasing during the last decades. As a side effect of the technological growth, computer scientists have experienced the problem of having to deal with multiple scenarios. Algorithms are often supposed to handle black-box problems, but also very specific cases through a tailored design. Even more complications arose by the need to consider the implementation of similar techniques, but in completely different devices: e.g. we can think of programmable wireless sensors with modest memory and computational power, see (Iacca, 2013), against other distributed systems, see (Weise and Chiong, 2010), or fast multi-core personal computers, FPGA and GPU units (Tan and Zhou, 2011; Ramírez-Chavez et al., 2011), etc. Novel optimization problems derived from new technologies, and new challenges continue to arise for computer scientists, who are often asked to satisfy conflicting needs as much as possible. It is quite clear that is difficult, if not impossible, to make coexist too many properties such as versatility and efficacy, robustness and specificity. A trade-off is inevitable.

The scientific community of computational intelligence had to react promptly to keep up with the technological growth. Innovative "evolutionary" approaches, already pioneered by Alan Turing in the late '40s, have been thoroughly proposed and studied. In the last four decades, they have been significantly improved , and the trend of using nature-inspired approaches has become established (Chiong, 2009; Chiong et al., 2012). Those more flexible algorithms borrowing some key mechanisms from physical processes, such as evolution and animal or social collective behaviours, see e.g. (Chiong, 2009), ended up replacing the former gradient-based methods, e.g. Newton and Quasi-Newton strategies (Fletcher and Powell, 1963; Press et al., 2007). Amongst them, few approaches emerged from a plethora of similar techniques based on different metaphors: Evolutionary Algorithms (EA) such as the popular Genetic Algorithm (GA) and the Evolution Strategy (ES), see (Holland, 1975a) and (Beyer and Schwefel, 2002) respectively, and Swarm Intelligence (SI) approaches such as the Particle Swarm Optimization (PSO) algorithm (Kennedy and Eberhart, 1995), are now consolidated frameworks. Also Differential Evolution (DE), by Storn and Price (1995), plays a major

role for real-valued optimization and is largely applied on engineering applications. It is finally worth mentioning the Simulated Annealing (SA) algorithm (Kirkpatrick et al., 1983), which is employed for both discrete and real-valued applications.

For years, it was a common believe that the aforementioned techniques were general-purpose, and that was possible to improve them to the point of finding some sort of "universal optimizer". Unfortunately, experience over the years has discouraged this belief, and the point made by the No Free Lunch Theorems (NFLTs) in (Wolpert and Macready, 1997) has shown the need of tailoring an optimizer to the specific problem, when possible, in order to increase the performance. Hence, mechanisms have been proposed to specialize a metaheuristic to the problem at hand. As example, algorithm's parameters can be self-adapted on-the-fly as in (Brest et al., 2006), (Qin and Suganthan, 2005) and (Zhang and Sanderson, 2009). Alternatively, some sort of more specialized local search routines can be embedded within the evolutionary cycle of en EA, as proposed in (Moscato, 1989) under the name of Memetic Algorithm (MA). The latter approach is undoubtedly interesting as it leaves room to the designer for applying different algorithmic solutions. Therefore, the MA approach has been largely used in the last two decades, and further extended to the generation of hybrid structures. Such algorithms are the results of the coordination, or fusion, of multiple metaheuristics and (or) other operators. The most successful works on such hybridization processes have produced similar approaches, supported from two different school of thoughts: Memetic Computing (MC), see (Ong et al., 2010), and Hyper-heuristics, see (Burke et al., 2003b; Özcan et al., 2008; Burke et al., 2010). Regardless of the philosophy behind them, both the approaches generated valid algorithmic solutions, and still seems to have a great potential. Interesting MC algorithms, e.g. (Krasnogor and Smith, 2005), (Nguyen and Yao, 2008) and (Smith, 2007), have shown to be able to adapt to the problem at hand. Also hyper-heuristics, see (Burke et al., 2003a), (Özcan et al., 2008) and (Burke et al., 2010), proved to work well on both discrete and real-valued problems.

Very complex MC structures have been designed by integrating several optimizers, see e.g. (Montes de Oca et al., 2009), (Molina et al., 2010a) and (Peng et al., 2010). In contrast to this trend, recent studies have shown that also simple algorithms can perform as well as complex ones, see (Iacca et al., 2012a; Caraffini et al., 2012b, 2013), if their operators are properly coordinated within the algorithmic structure (Caraffini et al., 2012a). Thus, specialized MC algorithms can be obtained by coordinating very simple operators rather then complex algorithms. General-purpose algorithms tend to be complex and heavy, and adding local searchers inevitably results into a more complex structure. In this thesis, besides a number of general purpose algorithms, lighter optimizers were proposed in order to tackle hardware limitations. Simple MC algorithms were implemented on board of microcontrollers, and other optimizers were simplified to run faster and use a significant inferior amount memory. This was possible by considering the bottom-up approach suggested in (Iacca et al., 2012a), the "compact" optimization approach in (Harik et al., 1999; Mininno et al., 2011), and other solutions.

As a number of population-based algorithms are also presented in this work, an interesting study on the side effect of having a large population of solutions was also included.

It has to be remarked that this piece of work focusses on a subset of CIO dealing with *real-valued, statistic, singe-objective, box-constrained* problems. The multi-objective case, as well as the discrete one, the dynamic one, etc., are not considered. Without a loss of generality, it is assumed that the optimization problem to be solved is a minimization process. All the algorithms involved in this study are equipped with toroidal correction, see (Caraffini, 2014), to handle solutions generated out of the bounds of the search space.

ACRONYMS

3SOME	3 stage Optimal Memetic Exploration
AGV	Automated Guided Vehicle
AOS	Adaptive Operator Selection
BGA	Breeder Genetic Algorithm
BIOS	Basic Input-Output System
BLX	BLend X-over
CCPSO2	Cooperatively Coevolving Particle Swarms 2
CDF	Cumulative Distribution Function
CEC	Congress on Evolutionary Computation
CLPSO	Comprehensive Learning Particle Swarm Optimization
cGA	compact Genetic Algorithm
CoDE	Composite Differential Evolution
DE	Differential Evolution
EBP	Error Back Propagation
EDA	Estimation of Distribution Algorithm
EP	Evolutionary Programming
EPSDE	Ensemble of Parameters and Mutation Strategies DE
EPSDE-LS	EPSDE empowered by Local Search
ES	Evolution strategy
FE	Fitness evaluation
FPGA	Field Programmable Gate Array
GA	Genetic Algorithm
GPU	Graphics Processing Unit
GRQ	Global Research Question
GS	Global Search
ISPO	Intelligent Single Particle Optimizer
LS	Local Search
LSOP	Large Scale Optimization Problem
MA	Memetic Algorith
MADE	Multicriteria Adaptive Differential Evolution
MC	Memetic Computing
MDE-pBX	Modified Differential Evolution with p-best crossover

MN	Memetic Node
MS-CAP	Multi-Strategy Coevolving Aging Particle
MSE	Mean Square Error
NDBI	Normalized Distance to the Best Individual
NFLT	No Free Lunch Theorem
NRFI	Normalized Relative Fitness Improvement
PDF	Probability Density Function
PID	Proportional Integral Derivative
PLC	Programmable Logic Controller
PM	Probability Matching
PMS	Parallel Memetic Structure
RAM	Random Access Memory
rcGA	real-valued compact Genetic Algorithm
RIS	Re-sampled Inheritance Search
RP	Resilient Propagation
RQ	Research Question
RS	Re-sampled Search
RTOS	Real-Time Operating System
RWS	Roulette Wheel Selection
SADE	Self-adaptive Differential Evolution
SDE	Self-adaptive Differential Evolution
SMADE	Super-fit Multicriteria Adaptive Differential Evolution
SPAM	Separability Prototype for Automatic Memes
SPSA	Simultaneous Perturbation Stochastic Approximation
SUS	Stochastic Universal Sampling
VISPO	Very Intelligent Single Particle Optimization

LIST OF FIGURES

LIST OF TABLES

CONTENTS

LIST OF INCLUDED ARTICLES

PI Giovanni Iacca, Fabio Caraffini and Ferrante Neri. Compact differential evolution light: high performance despite limited memory requirement and modest computational overhead. *Journal of Computer Science and Technology*, volume 27, number 5, pages 1056-1076, 2012.

PII Giovanni Iacca, Fabio Caraffini, Ferrante Neri and Ernesto Mininno. Robot Base Disturbance Optimization with Compact Differential Evolution Light. *Applications of Evolutionary Computation (EvoApplications)*, Lecture Notes in Computer Science Volume 7248, pages 285-294, 2012.

PIII Giovanni Iacca, Fabio Caraffini and Ferrante Neri. Memory-saving memetic computing for path-following mobile robots. *Applied Soft Computing*, volume 13, number 4, pages 2003-2016, 2013.

PIV Fabio Caraffini, Ferrante Neri and Ilpo Poikolainen. Micro-Differential Evolution with Extra Moves Along the Axes. *IEEE Symposium on Differential Evolution (SDE)*, pages 46-53, 2013.

PV Jixiang Cheng, Gexiang Zhang, Fabio Caraffini and Ferrante Neri. Multicriteria Adaptive Differential Evolution for Global Numerical Optimization. *Integrated Computer-Aided Engineering*, pre-press, DOI 10.3233/ICA-150481, 2015.

PVI Fabio Caraffini, Ferrante Neri, Jixiang Cheng, Gexiang Zhang, Lorenzo Picinali, Giovanni Iacca and Ernesto Mininno. Super-fit Multicriteria Adaptive Differential Evolution. *IEEE Congress on Evolutionary Computation (CEC)*, pages 1678-1685, 2013.

PVII Caraffini, F. and Iacca, G. and Neri, F. and Picinali, L. and Mininno, E.. A CMA-ES super-fit scheme for the re-sampled inheritance search. *IEEE Congress on Evolutionary Computation (CEC)*, pages 1123-1130, 2013.

PVIII Iacca, Giovanni and Neri, Ferrante and Caraffini, Fabio and Suganthan, Ponnuthurai Nagaratnam. A Differential Evolution Framework with Ensemble of Parameters and Strategies and Pool of Local Search Algorithms. *Applications of Evolutionary Computation (EvoApplications)*, Lecture Notes in Computer Science, Volume 8602, pages 615-626, 2014.

PIX Giovanni Iacca, Fabio Caraffini and Ferrante Neri. Multi-strategy coevolving aging particle optimization. *International journal of neural systems*, volume 24, number 1, pages 1450008(1)-1450008(19), 2014.

PX Michael G. Epitropakis, Fabio Caraffini, Ferrante Neri and Edmund K. Burke. A Separability Prototype for Automatic Memes with Adaptive Operator Selection. *Applications of Evolutionary Computation (EvoApplications)*, Lecture Notes in Computer Science, Volume 8602, pages 615-626, 2014.

PXI Anna V. Kononova, David W. Corne, Philippe De Wilde, Vsevolod Shneer and Fabio Caraffini. Structural bias in population-based algorithms. *Information Sciences*, volume 298, number 0, pages 468-490, 2015.

The author of this thesis made a relevant contribution on each included paper. He coded the entire software related to the production of the 11 papers attached to this piece of work. In particular, he autonomously produced the statistical comparison platform and the software for automatic results visualization, that were used for all the research performed during his doctoral studies. Furthermore, the author actively participated to the algorithmic design of all the optimization algorithms presented in this thesis, taking care of the majority of the design in most of the contributions. He took part to the writing of all the attached papers.

Regarding the work in PI, which introduced the author to CIO, he made the implementation of the proposed algorithm specific to the engineering application under study. Moreover, he contributed along with the other authors to the conceptual design of the algorithmic structure. Also, he performed a Matlab Stateflow implementation for the comparison algorithms involved in the robotic application. The author also simulated the path-following task by creating a Matlab Simulink model for the robot and for the simulation environment.

As for PII, he designed the simulation platform, the experimental set-up of the robot application, and led the experiments. He produced data and the statistical analysis in the paper.

In PIII, the author made a major contribution. He assembled all the hardware and coded all the software for the proposed real-world application. Optimization algorithms were also implemented on board of the robot by the author of this thesis. He independently played a major role by equipping the robot with routines for sensors calibration, and with a bluetooth system. The latter, was used for collecting data and monitoring the control system in real-time. As for the tuning of the employed PID regulator, he individually compared several computational intelligence approaches against standard solid methods, such as the Ziegler-Nichols and the areas methods.

In PIV, he entirely performed the algorithmic design, developed all the software, and statistically interpreted numerical results.

As for the study in PV, he made a contribution to the paper by designing the algorithm jointly with the international colleagues. Moreover, he performed the experiments, and independently added a real-world application besides the benchmarks problems.

The algorithm proposed in the previous paper was subsequently enhanced in PVI, a study entirely led by the author of this thesis. He autonomously took care of the algorithmic design, software implementation, data production, images, tables and statistic tests.

The same was done for PVII, which is major contribution of the author of this thesis.

A contribution was given to the design of the algorithm in PVIII. Implementation of the algorithm, as well as production, visualization and interpretation of numerical results are main contribution of the author of this thesis.

All the software related to PIX was designed, implemented, and run by the author of this thesis. He took care of interpreting and displaying data, and produced a further document (available on-line) with additional numerical results.

A major role was played in terms of algorithmic design, software implementation, data production and visualization, in order to produce the research published in PX.

Finally, he took a major part in the research leading to the production of PXI. In particular, he looked after all the software design, and produced the results as indicated in the theoretical part of the paper.

1 RESEARCH QUESTIONS

This thesis presents an extensive work including 11 publications. A first research output, focussed on memory-saving optimization for robotic applications, was independently performed at University of Jyväskylä, within the CIO research group. Then, the continuous involvement in the life of the research group, attending conferences, and the growing interest in this discipline, generated several side projects and international collaborations. Thus, the original research focus has slightly changed and evolved during the years. For these reasons, this work progressively addresses multiple research questions.

As discussed in the preface, a first interest was given to robotic applications. Several cases are included in this thesis. As for the applications in PI and PIX, an off-line approach was appropriate for addressing simulations and training of neural networks. Conversely, the decision of optimizing the control system on-the-fly, rather than off-line, looked more suitable, interesting and innovative for other studies. However, the idea of performing such optimization on board of robots, was in contrast with the use of too heavy and complex algorithms. Implementation details about the nature of the devices running the optimization process had to be considered during the algorithmic design. The need of embedding the optimizer together with the control system in a microcontroller forces the use of simpler algorithmic structures. The design had to play a major role in order to come out with algorithms that, despite being simple and memory-saving, display a good performance. Moreover, a too high computational overhead (typical of complex metaheuristics) was undesirable in order to guarantee a real-time response. For these reasons, the following research question (**RQ**) had arisen: *is it possible to design simple, but effective, optimization heuristics that can be easily embedded in systems plagued by memory limitations? Is it possible to modify successful approaches and make them faster in order to tackle real-time problems?* (**RQ** I)

In a second moment, as dealing with general purpose algorithms, a special attention was given to DE, MC and hyper-heuristic variants. Six novel algorithms were proposed in order to address the following research question *is it possible to design a robust and reliable general purpose optimizer, capable of self-adapting to the problem at hand? What alternative methods can be used, on top of those already present*

in the literature, for a better adaptation and thus for better performances? **(RQ** II)

Advantages of using population based algorithms have been largely commended in the past (Prügel-Bennett, 2010), while their limitations have not been addressed yet. **RQ** II arose the need of understanding what mechanisms take place within population based algorithms, and which one is beneficial rather then counterproductive. For such algorithms to be successful, is known that an effective informed (possibly not revisiting) sampling strategy is needed, as well as the absence of a structural bias, which, if present, would predispose the algorithm towards limiting its search to specific regions of the solution space. Only a little is known about the latter, thus, a last **RQ** was formulated as follow: *is the search, in popular general-purpose heuristics such as GA and PSO, unbiased? Are they able to equally explore different landscapes, or necessarily present a preferential bias towards a certain region of the search space?* **(RQ** III)

Despite the heterogeneous nature of this thesis, the 3 the research questions raised by far have a common point, and can be summarized in a single "global research question" (**GRQ**) motivating this piece of research: *What are the successful design strategies for efficiently tackling real-world scenarios, and in particular engineering optimization problems?*

1.1 Thesis structure

The remainder of this thesis is structured as follows:

Chapter 2 contains introduction and background of this piece of research. A formalization of the addressed optimization problem is first given in section 2.1. Subsequently, Section 2.2 comments on the metaheuristic approach. A particular attention is given to differences and use of global and local searchers, see Section 2.2.1. The exploration-exploitation conundrum is also discussed. A technical description of the most popular local searchers in the literature is provided as well. It follows a wider literature review of nature inspired metaheuristics for general purposed optimization (Section 2.3). A special mention is given to state-of-the-art implementations, that have been involved in this work.

Chapter 3 addresses **RQ** I. In particular, the enhanced version of compact differential evolution in Section 3.2, allows real-time optimization in embedded systems. An example of optimization within embedded systems is given by the robotic application in Section 3.3. Moreover, the robotic application in Section 3.4 was tackled by running cherry-picked memory saving optimizers on board of the root, thus performing the optimization on the go. Finally, Section 3.5 concludes with a novel optimizer that, despite storing in the memory a population of only five individuals, performs brilliantly over large scale problems.

Chapter 4 tackles **RQ** II and proposes six novel algorithms for black-box optimization. Varied algorithmic solutions were used. In Section 4.2 have been grouped those approaches making use of mechanisms for tuning problem dependent parameters on-the-fly. Three publications have been produced in this

direction. A brief description of the original papers is given in Section 4.2.1, 4.2.2 and 4.2.3 respectively. Hybrid structures, employing self-adapting coordination logics, are instead placed in Section 4.3. In particular, the three remaining optimizers are introduced in Section 4.3.1, 4.3.2 and 4.3.3 respectively.

Chapter 5 answers to **RQ** III through a theoretical study on population-based algorithms, pointing out the presence of a structural bias in the algorithmic architecture. By showing a relation between bias intensity and population size, this study concludes this piece of work.

Chapter 6 summarizes the achievements obtained during my doctoral studies. Conclusions on how this work addressed the research questions are drawn, and promising future developments are proposed.

2 INTRODUCTION TO CIO

CIO is the subject which embeds AI into optimizers for solving those problems that, due to the lack of an analytic formulation, lack of differentiability or complexity, cannot be addressed by means of classic methods. As previously mentioned in the preface, during the last 4 decades the literature has exploded, and now features a plethora of techniques inspired by the most diverse metaphors. How to classy an algorithm according to such metaphors has become difficult. Researchers are often in contrast and tend to support an approach rather than another, though the difference is minimal. Many attempts are currently made in order to design algorithms based on new metaphors, but the result is often not so differing from previous approaches, in terms of operators employed, and performances. Thus, *it can be deduced that most successful algorithms share the same working principles, and besides the inspiring metaphor they behaves quite similarly. All the optimization algorithms are expression of the same concept, that is the balance between the capability of exploring the search space, and exploiting potential optima*. Taxonomies are often worthless and put to much emphasis on irrelevant particulars. On the other hand, it is useful to highlights properties such a memory footprint, computational overhead and convergence speed, use of the gradient information, deterministic or stochastic nature, etc. These information help the user select the right algorithm in order to meet his needs, and the designer as well in creating effective hybrid algorithms. In this light, the reminder of this chapter has been organized to discuss the most important properties of metaheuristics. A special attention is given to the concept of local and global search, and several single solution local searchers are described in details. A concise literature review is given in Section 2.3, with a strong focus on recent advances in CIO. All the optimizers included in the attached publications are fully described in depth in the following sections.

2.1 Problem definition

With regards to this piece of work[1], the optimization problem can be simply formulated as the search of a real-valued vector $\mathbf{x}^* \in \mathbf{D} \subset \mathbb{R}^{\mathbf{n}}$, minimizing a mathematical function $f : \mathbf{D} \longrightarrow \mathbf{C} \in \mathbb{R}$. The cost-functional f, often called objective function or fitness function, models the optimization problem at hand and has to be defined accurately in order to achieve truthful results. The number $n \in \mathbb{N}$ is the dimensionality of the problem, and the n components of a solution within the the search space D, are usually referred to as design variables. The scalar number C is the cost associate to a specific solution, often referred to as fitness value.

Mathematically, what discussed above can be synthetically formulated as:

$$find\ \mathbf{x}^* \equiv \arg\{\min_{\mathbf{x}\in\mathbf{D}} f(\mathbf{x})\} = \{\mathbf{x}^* \in \mathbf{D} \,|\, f(\mathbf{x}^*) \leqslant f(\mathbf{x})\ \forall\, \mathbf{x} \in \mathbf{D}\} \tag{1}$$

2.2 The metaheuristic approach

Metaheuristic[2] optimizers are those algorithms which do not make/need any assumption on the problem at hand. Such approaches play a crucial role when facing too complex, or black-box, systems. In both the cases, despite the lack of knowledge about the problem, a fairly good (sub)optimal solution has to be found via a trial-and-error process (Michalewicz and Fogel, 2004; Burke et al., 2010).

The most consolidated examples of metaheuristics for optimization are population based algorithms. Their large use during the last years made them become so popular that, often, the word metaheuristic is confused with population based algorithms. Sometimes, this term is improperly used in the literature as synonym of evolutionary framework for black-box optimization (BBO) . There is a strong link between the two concepts, but they do not have the same meaning. EAs have a stochastic nature and are efficient metaheuristics for BBO. Thus, EAs as well as PSO, DE, etc. are contained in the metaheuristics set. But not vice versa. For instance, deterministic[3] optimizers, such as the Rosenbrock method (Rosenbrock, 1960) and the S operator (employed in PIII, PVII and PX), are metaheuristics too as both can function on a generic fitness function. Conversely, Newton/Quasi-Newton/Gradient Descent methods (Xu and Zhang, 2001), that are often integrated within EAs, requires the problem to be differentiable in order to function. Most efficient variants requires the fitness function to be twice

[1] There are other optimization scenarios, where f could be e.g. a discrete function, or a dynamic process, and even subject to both equality and inequality constraints.

[2] Word deriving from ancient Greek, its literal means is "beyond the search". It refers to any approach to discovering or problem solving, based on a practical methodology. They do not guarantee optimal results, but are generally valid and sufficiently accurate.

[3] Algorithm free of any form of randomization logic, performing a predictable sequence of steps.

differentiable, as they work out second order derivatives to build up the Hessian matrix. Thus, their use over discontinuous functions, could lead to numerical instability. A problem that is overcome by those metaheuristics that only approximate the gradient information, such as the Simultaneous Perturbation Stochastic Approximation (SPSA), see (Spall, 1987) for details. A loss in the average performance of metaheuristics, as SPSA, against the Newton method can be experienced over unimodal differentiable problems. However, such loss is paid off with a higher versatility. A mathematical explanation of this phenomenon is given in (Wolpert and Macready, 1997). The popular NFLT rigorously formalized a concept that, against the common belief of that time, can be expressed as: a universal metaheuristic, capable of outperforming all the other algorithms over all the possible problems, does not exist. On average, the performances displayed by two algorithms over all the possible problems are the same. A metaheuristic can potentially handle every problem displaying a decent result. Therefore, cannot compete against tailored algorithms. For this reason, metaheuristics are widely used in real world scenarios as black-box systems, in absence of an analytic model, in presence of high complexity. However, even if just a little is known abut the problem at hand (e.g. grey system), a tailored approach is preferable.

As discussed in the preface, taxonomies are often obtained by comparing algorithms with biological (or other natural) phenomena. These analogies often lead to philosophical rather than practical considerations. Conversely, in this thesis, a strong attention is given to some of the previously mentioned properties of an optimization algorithm. This information can be used for selecting the most suitable optimizer, or for combining complementary algorithms together. As example, it can be of help to know whether or not the nature of an optimizer is purely stochastic or deterministic. Also the kind of search plays an important role: gradient based and derivatives free searches behave differently and display pros and cons. Stochastic mechanisms (perturbations, sampling, etc.) tend to outperform deterministic ones over noisy landscapes. They have a better chance to get out of local minima, while deterministic routines usually get stuck in such critical points. In particular, those employing some sort of gradient information converge faster and are preferable on monomodal problems. They stand out for exploiting and refining a start point quite accurately, but are less adequate for exploring multiple optima. Also the kind of move performed within the search space can make the difference. Separable problems are better tackled with orthogonal moves rather than diagonal ones. The former, can be produced by perturbing a single design variable at a time, and evaluating the fitness value right after the perturbation. The perturbation vector can be both deterministically or randomly generated. The latter, can be obtained by using rotation matrices, see the Rosenbrock method, or via combination of individuals in population based algorithms. Finally, when the problem is completely unknown, and the presence of noise (and/or multiple local optima) is likely, population based algorithms have shown to be preferable.

The benefits of a having a population have been studied and praised in the past years. According to Prügel-Bennett (2010), population based metaheuristics

are stable and reliable (even though stochastic, they tend to locate very close solutions if run multiple times), robust (as the use of a large population size has the effect of filtering out noise in the fitness function), and have a number of other beneficial advantages. On the other hand, undesired effect can occur also when manipulating multiple solutions (e.g. stagnation, premature convergence, etc.). In addition, tuning the population size is not trivial. Performances heavily depend ot the optimal choice for the population size. However, even if a number of studies can be found in the literature, see e.g. (Nannen et al., 2008) and (Mallipeddi and Suganthan, 2008), how to make this choice is still not clear. It also appeared, from the study in PXI, that a large population size can negatively bias the search towards preferential suburbs of the search space. In this light, also single solution metaheuristics deserve some attention, and can play a role in some cases. They usually display simpler structures, and requires a modest amount of memory and computational power. In particular, a number of recent implementations have shown that, if well designed, also simple single-solution structures can have performances as good as those of population base algorithms (Iacca et al., 2012a; Caraffini et al., 2013; Iacca et al., 2013). These can be preferred to tackle real-time applications in systems plagued by memory limitations. In any way, it has to be clarified that in the vast majority of the cases both population-based and single-solution algorithms have a linear memory occupation. Furthermore, several efficient single solution methods, e.g. Rosenbrock and Powell (Rosenbrock, 1960; Powell, 1964), evolve a rotation matrix whose size grows quadratically with the dimensionality of the problem. In these cases, single solution algorithms are are not suitable for memory-saving optimization.

Finally, it is important to further remark that the core part of a metaheuristic for optimization is common to all the approaches. All of them are variations on a main theme, that is the generation of new candidate solutions according to a perturbation mechanism. When this mechanism is efficient, the algorithm is able to learn from previous moves, and the number of unsuccessful perturbations drop down as the optimization process proceeds. The process of acquiring information from past trial-and-error attempts, and exploit the gathered information to learn from past steps, is called *Inductive Learning* (Brownlee, 2011). An implicit assumption is actually made when such process is employed, i.e. the learning mechanism relies on a specific configuration (set of candidate solutions at the previous step), that is trusted to be representative of a broader environment. This is not entirely true, and for this reason, a certain degree of fitness diversity must always be kept at the beginning of the optimization process. To guarantee an appropriate balance of exploration and exploitation, is as important as difficult. Good results have been obtained by coordinating global and local searchers, see Section 2.2.1. However, the need of performing the optimization with a limited computational budget (it occurs in the vast majority of real world scenarios) prevent from a prolonged use of a specific operators, e.g. a local searcher.

2.2.1 On local and global search

A global searcher seeks for a global solution, while a local one refines as much as possible the closer local optimum. As the former usually fails at refining the global optimum adequately, it is necessary to be able to embed some sort of LS behaviour within global searchers. In multimodal optimization, GS is employed to implement niching algorithms. The search aims at locating multiple clusters in the search space, see (Thomsen, 2004), (Li, 2010) and (Yu and Suganthan, 2010), thus exploration plays a major role. Conversely, in global optimization, a unique global solution must be returned with a satisfactory level of precision. It is then desirable that a LS behaviour emerges towards the end of the optimization process. For this reason, general-purpose algorithms combine global and local search capabilities at the same time. Ideally, the landscape has to be explored as much as possible during an initial phase. Then, the search is focussed around potential optima, but a certain degree of exploration is kept to avoid premature convergence and promptly leave local minima. The last amount of computational budget is used to exploit the most promising basins of attraction, in order to converge within a reasonably small neighborhood of the global optimum. This process can be iterated, or restarted, in order to minimize the risk of returning unsatisfactory (or useless) solutions.

To make GS and LS coexist within the same framework is hard. The literature proposes several algorithmic solutions to alternate exploratory and exploitative moments. In population based optimization, as instance, these moments are intrinsically assured by the use of the population. An initial sampling makes sure that the candidate solutions are uniformly distributed in the search space. Then, a trial and error process guides the individuals towards promising basins of attraction. Selection operators play a major role. If they are chosen and tuned properly, fast convergence can be achieved (e.g. pick up fittest individuals for generating new candidate solutions, and accept in the new population only those with a promising fitness value). Other choices can be made to obtain the opposite behaviour. A compromise solution can also be implemented, so that individuals disperse if too close and vice versa. Several solutions can be adopted by the designer, who can modify and tune the algorithm to face a specific scenario. Unfortunately, black-box problems make it very difficult to understand which combination of operators is best. As the problem changes, that combination should change as well. Also the computational budget should be specifically allocated to exploratory and exploitative moments. An option to address black-box problems is too keep the algorithmic structure generic, or even better, adaptable to the problem (see Chapter 4). However, undesired configurations occurs (disposition of the individuals within a particular fitness landscape), causing the algorithm to malfunction. For examples, many GAs fails at handling plateaus due to the lack of fitness diversity that, in return, reduces the selection pressure. In some other cases, e.g. multimodal landscapes, individuals trapped in local minima drag the entire population towards an undesired area of the search space (low diversity). The local minimum get explored and refined, while the global minimum remains

ignored. Thus, premature convergence occurs: the algorithm is not able to explore D any more, and the remaining computational budget is consumed in vain. Modern metaheuristics embed mechanisms to re-sample the population in D, in order to minimize the use of computational budget around local solutions. Usually, metrics are proposed to check and quantify the diversity of the population, as in PV, to prevent premature convergence from arising. Nevertheless, other undesired configurations can affect the search. When stagnation occurs, as instance, individuals reach a configuration such that the algorithm is not able to improve upon their fitness values. Strangely, this happens with a high diversity in the population. Stagnation is typical of DE. It can be tackled by the use of external routines, see PIV, able to further refine the solutions within their neighborhood. Such routines take the name of local searchers and, in opposition to general purpose metaheuristics, are specialized in exploiting solutions rather than exploring the domain.

The observation that some algorithms are better at exploring, while others at refining results, generated new algorithmic solutions. As discussed before, the MA approach is based on the use of LS within general purpose optimizers. Usually, an "evolutionary" component takes care of exploration, while exploitation is guaranteed by the use of a specialized operator. Thus, the balance between exploration and exploitation is given by continuously alternating operators playing different parts. An efficient coordination logic, as those proposed in MC and hyper-heuristics, can repetitively launch GS and LS when is more appropriate. This approach is promising, but does not completely resolve the global-local search conundrum. By including LS within a population based framework, we experience a boost in the performance. This is due to the fact that local searchers have properties making them capable of addressing specific problems. When they are embedded within a general purpose framework, the latter become more specialized in handling a specific class of problems. As a result, the tailored version of that algorithm presents a peak in the performance over that class of problems (see Figure 1), and a slight deterioration over the others (Wolpert and Macready, 1997). As a consequence, researchers tend to equip MAs with a list of different LS routines, to handle multiple scenarios (Tseng and Chen, 2008; Iacca et al., 2014; Caraffini et al., 2012). From the theoretical point of view this approach is flawless. On the other hand, it is difficult to be put into practice. In order to function, the scheme requires local searchers to be run multiple times, thus subtracting a substantial portion of the computational budget to the other operators. Most studies, as well as popular competitions in the field, allow a budget of $5000 \times n$ fitness evaluations (FEs) to assure convergence. However, this budget could be insufficient in some cases, albeit quite high if compared with the budget that can be actually used in many engineering applications. In conclusion, to embed both global and local exploration capabilities is still a problematic task.

Algorithms specialized in exploration do exist in the literature. In the past,

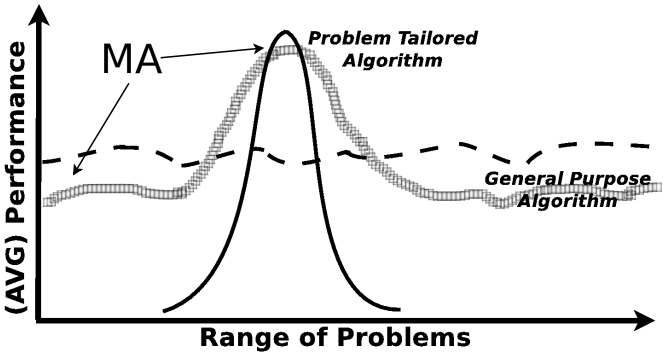


FIGURE 1 Average performance of general-purpose metaheuristics (NFLT) [dashed line], algorithms tailored to the problem [continuous line] and metaheuristics enriched with LS [grey squared line].

simple methods as the random walk (stochastic), or brute force approaches (deterministic) have been proposed. More evolved stochastic methods were successfully designed. For instance, Iacca et al. (2012a) and Caraffini et al. (2012) made use of an evolved random walk algorithms, hybridized with a crossover operator from DE. Conversely, exhaustive approaches based on the brute force principle, are considered outdated and no longer employed. They are indeed inefficient if compared with modern population based algorithms. The latter, are quite suitable for exploration and need less FEs. Moreover, their convergence can be arrested if tuned (or designed) adequately. Selection and variation operators have to be carefully selected in order to guarantee pure exploration, as done e.g. in (Molina et al., 2010a). In this way, it is possible to spot the position of the global optimum, and save the computational budget needed to converge in that area for running a more efficient LS routine. According to this principle, former MAs were designed by integrating LS within a GA. In conclusion, population based algorithms are currently preferred to exhaustive methods and to single solution approaches (as the random walk), and are key to perform GS in memetic algorithms.

A rigorous definition for LS does not exist. Generally, LS algorithms are those routines working within a sub-portion of the search space, looking for local optima. This behaviour can be obtained by tuning general purpose algorithms with a strong emphasis on exploitation. For example, in (Molina et al., 2010a) a population-based ES was used to optimize locally. However, specialized LS routines also exist and are often based on a single solution scheme. This kind of optimizers are extremely sensitive to the initial guess and cannot be used on their own, unless the problem is monomodal. Their search is narrowed to a small portion of D that is usually referred to as neighborhood. The latter is simply defined, in discrete domains, as the set of points reachable with a pre-fixed number of steps. Similarly, in real-valued domains the neighborhood can be defined by considering the set of points reachable with a single perturbation. The perturba-

tion vector can be either deterministically worked out, or randomly generated. In both cases, it can be seen as an exploratory radius, whose maximum length is preferably prefixed and less than a given "small"[4] value. Both discrete and real-valued LS algorithms share the same structure, that is a loop containing a method for generating a new neighboring solution. The new solution is then compared with the previous one, that can potentially get replaced. Despite this similarity, real-valued domains can (and must) be treated differently. Important considerations can be made on real-valued sets, in order to speed up the search, make it more precise, and save computational budget. Even though all the domains encoded in a calculator are necessarily discrete, the precision of modern calculators is quite accurate. Thus, an estimate of the continuity of the domain can be done with a reasonably good approximation. Usually, points within a small neighborhood display smooth variations, i.e. the set is continuous, therefore it makes sense to use the gradient information to guide the search straight to the closest optimum. If a discontinuity is present, the gradient information could lead to numerical instability. So, if the problem is known to be continuous, then a direct use of the gradient is strongly suggested. Conversely, stochastic approximations of the gradient vector (Spall, 1987), Hessian matrix (Powell, 1964; Rosenbrock, 1960; Hansen and Ostermeier, 1996), or just a simple steepest descent method can be used to find the most promising direction to reach the minimum point. Regardless of the nature of the LS routine (e.g. deterministic/stochastic, gradient based/gradient free, greedy, etc.) some sort of gradient information, whether it is mathematically worked out or just heuristically approximated, has to be integrated within the perturbation logic.

It follows a brief review about LS methods. Technical details (e.g. kind of move performed, order of the gradient information used, memory footprint, etc.) are only schematically displayed in Table 1 from (Caraffini, 2014).

Popular local searchers

Important gradient based strategies are those deriving from the popular Newton method (Press et al., 2007). The original algorithm is one of the fastest LS in term of rate of convergence, but not in terms of actual speed. Every steps involves heavy calculations for obtaining and inverting the Hessian matrix. Thus, the elapsed time is usually high due to an elevate algorithmic overhead, quadratically increasing with the dimensionality of the problem. Obviously, the Newton method is not a metaheuristic approach, as problem has to be twice differentiable. However, it is still largely employed. The literature provides lighter alternatives: Quasi-Newton methods, as (Rosen, 1966) and (Xu and Zhang, 2001), requires less computational effort as the Hessian matrix is indirectly computed. Algorithms like the Steepest Descent method (Snyman, 2005) are even lighter, as requiring only the first order derivatives (i.e. gradient). It must be said that the lack of second order information results into a worse rate of convergence. Moreover, the

[4] Reasonably small value, depending on the problem. Can be constant or adapted on the go, e.g. reduced according to a logic.

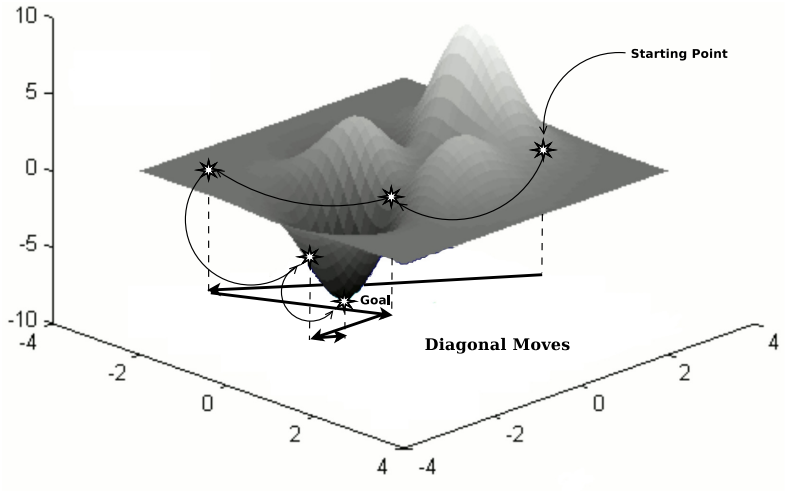


FIGURE 2 Diagonal moves along the search space.

search results quite efficient at the beginning of the optimization process, when the initial solution is far from the target. Conversely, a lack of precision is experienced when approaching the optimum.

A more robust metaheuristic LS is performed by the Powell's Direction Set (Powell, 1964) and the Rosenbrock (Rosenbrock, 1960) methods. The first one makes use of a matrix containing promising linearly independent directions. Similarly, the latter method evolves a rotation matrix for changing the coordinate system before the perturbation. Thus, both the methods make diagonal moves in the search space, as shown in Figure 2. The idea is to find the fastest way to the solution. To some extent, this is equivalent to say that both methods indirectly try to approximate the Hessian matrix. However, the deterministic logic used to build up the two matrices is completely derivative free. The presence of an $n \times n$ matrix makes the memory footprint grow quadratically with the dimensionally of the problem. For this reason, they are less preferable to lighter methods in embedded systems with limited memory capacity.

A peculiar search is performed by the Nelder-Mead method (Nelder and Mead, 1965). Perhaps not the most efficient search, but one of the most versatile. The algorithm does not rely on the gradient information at all, not even indirectly. A simplex is randomly built up within D. Subsequently, its vertices are moved though a succession of of geometrical transformations, according to a deterministic coordination logic. The algorithm can be seen as a population based exploration too, as $n + 1$ vertices are needed, and there is no guarantee that they will quickly converge.

The SPSA algorithm also presents an interesting and light structure (Spall, 1987). The gradient is stochastically estimated and can be employed in the perturbation also in presence of discontinuity points.

The Solis-Wets method, see (Solis and Wets, 1981), is completely randomized. A multivariate normal distribution is centred close to the current best solution, and used to stochastically generate a perturbation vector.

An interesting working principle was adopted in the Hooke-Jeeves method (Hooke and Jeeves, 1961). The main idea was to alternate two strategies, namely exploratory and pattern moves. The exploratory strategy moves the current solution (let us say $\mathbf{x_c}$) along the coordinate axes (i.e. orthogonal perturbations), by adding a quantity ϵ to each design variable at a time. Before perturbing the next design variable the fitness value is evaluated. If the previous configuration was fitter, it is restored and then the same quantity, i.e. ϵ, is subtracted to check also the opposite direction. Otherwise, the new configuration is retained. After perturbing all the variables, if at least an improvement has occurred, the pattern search can be performed to speed up the process. A "jump" is made along the vector pointing from $\mathbf{x_c}$ to the achieved configuration. This algorithm is no longer used in its original form, but the exploratory search has been subsequently improved to converge to the minimum. This was achieved in (Tseng and Chen, 2008) by performing an asymmetric sequence of steps along the axes according to the logic graphically shown in Figure 4. As can be seen, the quantity is first added, but if the fitness value does not improve only half step is performed in the opposite direction. The asymmetry in the search minimizes the probability of revisiting the same point multiple times. Moreover, when all the variables have gone through this procedure, the exploratory radius is updated. In particular, if not even 1 improvement has been registered, the radius is halved. The same length is kept otherwise. The idea is that when the radius no longer able to improve upon the fitness value, then the solution is within the neighborhood. Thus, a shorter distance exploration is attempted. This logic has shown to be able to refine a local minimum efficiently. It has been successfully employed in many studies, e.g. (Iacca et al., 2012a), (Caraffini et al., 2012), and (Caraffini et al., 2013), where it has taken the name of "S"[5] operator. This is a deterministic logic, making orthogonal moves along the coordinate axes, as shown in Figure 3, that appeared to be particularly promising on separable and large scale problems.

2.3 Brief literature review and recent advances in CIO

2.3.1 Evolutionary algorithms

Amongst the most important families of population-based optimizers, EAs are probably the former and most popular implementations. The first ESs appeared in the mid '60s (Schwefel, 1965)[6], even though the idea of using evolution for driving the search of extrema is older, see (Bremermann, 1962). Conversely, the well-known GA (probably the most popular algorithm!) appeared in the mid '70s (Holland, 1975b). Other EA based frameworks are e.g. Evolutionary and Genetic Programming (Fogel et al., 1966; Koza, 1992). All the aforementioned frameworks

[5] It stands for "Short" distance exploration, in opposition to the other L (Long distance exploration) operator in (Iacca et al., 2012a).

[6] A description in English is given in (Schwefel, 1981).

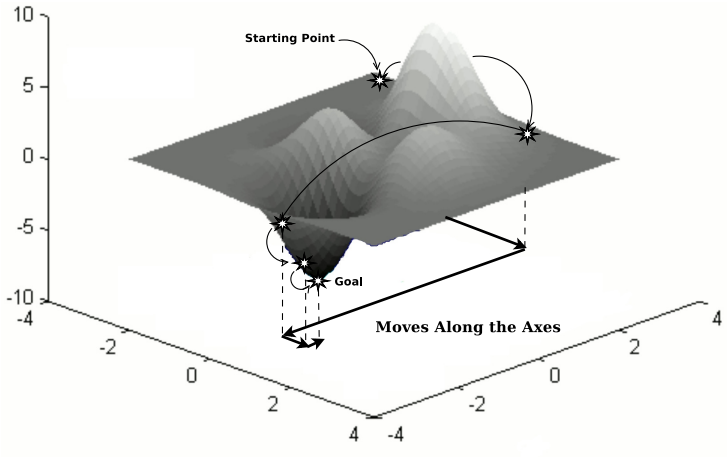


FIGURE 3 Perturbation moves along the axes.

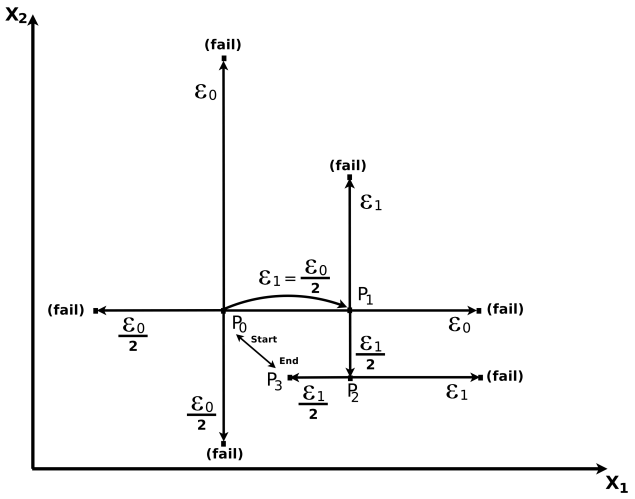


FIGURE 4 S operator logic.

TABLE 1 Local searchers main properties (n refers to the dimensionality of the problem).

LS algorithm	Search Logic	Derivatives	Memory Footprint	Processed Points	Convergence
NEWTON	*Deterministic (gradient descent)*	1^{st} *and* 2^{nd} *order*	$\mathcal{O}(n^2)$ *(Hessian matrix)*	*Single-solution*	*q-quadratically*[a]
HOOK-JEEVES	*Deterministic*	*Derivative free*	$\mathcal{O}(n)$	*Single-solution*	*No proof*
S	*Deterministic (along the axis)*	*Derivative free*	$\mathcal{O}(n)$	*single-solution*	*No proof*
NELDER-MEAD	*Deterministic*	*Derivative free*	$\mathcal{O}(n^2)$ *(simplex vertices)*	*Multiple-solutions*	*Convergence*[b]
ROSENBROCK	*Deterministic (diagonal move)*	*Derivative free*	$\mathcal{O}(n^2)$ *(rotation matrix)*	*Single-solution*	*Convergence*[c]
POWELL	*Deterministic (diagonal move)*	*Derivative free*	$\mathcal{O}(n^2)$ *(directions matrix)*	*Single-solution*	$n(n+1)$ *steps*[d]
SOLIS-WETS	*Stochastic (diagonal move)*	*Derivative free*	$\mathcal{O}(n)$	*Single-solutions*	*Convergence*[e]
SPSA	*Stochastic (gradient descent)*	1^{st} *order*	$\mathcal{O}(n)$	*Single-solutions*	*Convergence*[f]

[a] Only for unimodal *and locally twice Lipschitz continuously differentiable functions.*
[b] For convex functions in 1 and 2 dimensions.
[c] Under hypothesis on the fitness, such as differentiability, and on the line search method employed (Rinaldi, 2012).
[d] Using the classic method in (Powell, 1964) on quadratic forms.
[e] If f quasi-convex and inf-compact, converges in a neighborhood of the local minimum. (Solis and Wets, 1981).
[f] Under conditions on f and the distribution of probability used as in (Spall, 1992).

have their own peculiarities, but share the common EA structure displayed in Figure 5.

Genetic algorithms

In a GA, "selection" refers to both the first and the last step displayed in the general pseudocode (Figure 5). Usually, the first step is indicated as parent selection, while the second as survivor selection. However, many other nomenclatures can be found in the literature, as many variants have been designed. Some selection approaches are usable for both parent and survivor selection. Some other approaches, are specifically meant for a single purpose. Also for variation op-

```
Initialisation                          ▷ Randomly sample initial population
while condition on budget or fitness do
    Selection                           ▷ Pick the parents
    Recombination                       ▷ Generate offspring
    Mutation                            ▷ Alter genes
    Replacement                         ▷ Survivors drive the evolutionary process
end while
Output best individual
```

FIGURE 5 Pseudocode of an evolutionary algorithm

erators, i.e. recombination (aka crossover for GAs) and mutation, the literature provides a wide range of options.

A peculiarity of the GA is that parents are selected in a stochastic way (the same individual can be chosen multiple times for breeding), so that fitter solutions have a higher probability to guide the evolution. As they are more likely to generate a promising offspring than mediocre individuals, the latter are given a smaller, but still positive, chance to be picked up. A null probability of being selected has to be avoided also for very unfit solutions, so that a certain degree of population diversity is maintained. Diversity is important in order to assure exploration, thus preventing the algorithm from getting trapped in a local minimum. Classic methods for parent selection are the Roulette Wheel Selection (RWS), preferable for selecting a single individual at a time, and the Stochastic Universal Sampling (SUS), mostly used when multiple points are to be picked up in a row. Both the methods mimic the functioning of a roulette wheel. However, in the first case an unconventional wheel is adopted, with each sector having a different size proportionate to the fitness function of a corresponding individual. Thus, the wheel has to be spun k times to select k individuals. If the sampling is performed without replacement, a fit individual (e.g. local minimum) could be selected too often. This could lead to premature convergence. Conversely, with SUS only one spin is necessary to get the k desired points in one go, see Eiben and Smith (2003) for details. Generally, both the methods guarantee an appropriate level of selection pressure during the first part of the optimization process. Unfortunately, a dramatic decrease of such pressure can occur later on, when individuals tend to be close to each others. In this case, as well as in presence of plateaus, fitness values are very similar and so are the corresponding selection probabilities. This implicates that every point has the same chance of being selected. The selection mechanisms becomes a uniform selection, that is not desirable. To ease this problem, one of the Sigma Scaling methods proposed by Goldberg (1989) can be used. Such methods help diversify the selection probabilities in presence of steady landscapes. Alternative solutions to this problems are the rank based selections Eiben and Smith (2003). With respect to the aforementioned fitness based method, rank based routines display a slower convergence. On the contrary, they are not sensitive to fitness values, and assign the same rank whether the difference between the fitness of two individuals is huge or tiny. A very simple and used approach is the $k-$Fitness Tournament Selection scheme. To implement it, a set of k individuals have to be randomly picked up from the population to then select the fittest one amongst them. Finally, it is worth mentioning the Positive/Negative Assortative Mating selection. The two variants, i.e. positive and negative, can be chosen to obtain opposite effects. If a strong emphasis on exploration is sought, then the negative scheme is preferable. As example, it can be used to enhance GS capabilities in MAa, as done in (Molina et al., 2010a). The scheme aims at selecting distant individuals, see (Fernandes and Rosa, 2001). The first is usually selected via RWS. Then, a set of k points is also generated by means of RWS. A metric is chosen to work out the distances between the k points and the first parent, e.g. Hamming distance for binary do-

mains, Euclidean distance in real-valued ones, etc. The farthest point gets chosen as second parent. Positive Assortative Mating is conversely used to speed up convergence as selecting close parents (Fernandes et al., 2001).

As for crossover, a plethora of variants exist in the literature, see (Eiben and Smith, 2003). The main purpose of this operator is to generate an offspring by exchanging components (design variables) between two candidate solutions (parents). Multi-parents schemes also exist, but are rarely employed. Obviously, there are specific crossover methods for binary, combinatorial, integer, domains. Some of them can be transferred to the real-valued domain, e.g. see the one-point crossover (keeps the components of the first parent up to a given position, then exchanges the reaming sequence), and the two point crossover (exchanges a burst of genes within two given positions). Multi-point crossover routines have been defined too. Specific real-valued strategies are those performing the arithmetic average between the two parents, or other kind of processing, rather than a simple swap of the components. A peculiar crossover for real-valued domains is the BLX crossover operator (Eshelman and Schaffer, 1992). Given two individuals $\mathbf{x_1}$ and $\mathbf{x_2}$ and a parameter $\alpha \geq 0$, e.g. 0.5, BLX-α generates an offspring $\mathbf{x_{offspring}}$ by means of the following:

$$\mathbf{x_{offspring}}[i] = \mathcal{U}\left(min_i - \delta_i \cdot \alpha, max_i - \delta_i \cdot \alpha\right) \quad i = 0, 1, ..., n \tag{2}$$

with max_i and min_i being the maximum and the minimum value between the $i-th$ component of $\mathbf{x_1}$ and $\mathbf{x_2}$ respectively, and δ_i equal to the interval $max_i - min_i$ ($\mathcal{U}(a, b)$ is a uniform distributed random number within $[a, b]$). A value of 0.5 for α is sufficient to spread the distribution of the chromosome during the evolution, while BLX-0 is used to shrink the population distribution. Many other interesting crossover schemes can be found in (Gwiazda, 2006).

Mutation is conversely responsible for the exploration of new zones of the search space. In real-valued domains this is mainly done by altering the offspring solution via Gaussian or Uniform distribution based perturbations. A particular scheme, i.e. BGA mutation, was proposed in (Mühlenbein and Schlierkamp-Voosen, 1993) as:

$$\mathbf{x}^{\mathbf{mutated}}_{\mathbf{offspring}}[i] = \mathbf{x_{offspring}}[i] + \mathcal{B}^{\pm 1}(0.5) \cdot range_i \cdot \sum_{k=0}^{15} \alpha_k \cdot 2^{-k} \tag{3}$$

with $\mathcal{B}^{\pm 1}(p)$ being a modified Bernoulli distribution (returns 1 with probability p and -1 with probability $1-p$), $range_i = 0.1 \cdot \left(\mathbf{x^U}[i]^{7} - \mathbf{x^L}[i]^{8}\right)$, $i = 1, \ldots, n$ and α_k randomly sampled from $\{0, 1\}$ with a probability $p(\alpha_k = 1) = \frac{1}{16}$. Such probability implicates that the mutation is expected to occur 1 time for each gene, since the summation in Formula 3 iterates 16 times ($k = 0, 1, ..., 15$). As a result, the mutated gene is contained within the interval $\left[\mathbf{x_{offspring}}[i] - range_i, \mathbf{x_{offspring}}[i] - range_i \cdot 2^{-15}\right] \cup \left[\mathbf{x_{offspring}}[i] + range_i \cdot 2^{-15}, \mathbf{x_{offspring}}[i] + range_i\right]$.

7 Upper-bound for the i^{th} design variable in $\mathbf{D}$.

8 Lower-bound for the i^{th} design variable in $\mathbf{D}$.

Finally, also survivor selection can take place with different modalities. From this point of view, the GA is more flexible than other algorithms, where only few options are available. For example, old good individuals can be replaced with newly generated worse solutions to better handle multivariate functions (a logic that i rarely implemented by other optimizers). Moreover, exploratory implementations may require to replace the entire population, or just a few carefully selected individuals. Exploitative replacement schemes also exist to speed up the search. In order to be able to distinguish between survivor selection methods, a nomenclature is needed. Usually, those survivor selection schemes where the fitness value is not taken into consideration (or it is only partially taken into consideration) are referred to as aged-based, i.e. the new individual necessarily replace the old one. The others are instead referred to as fitness-based, e.g. the roulette wheel selection could be employed to select individuals forming a new population. Mixed combinations exist. As example, in the steady-state GA λ new solutions ($\lambda < \mu =$ population size) are produced through variation operators. Usually, $\lambda = 1$ or 2 depending on the crossover being used. The offspring can then simply replace those solutions that have been kept in the population for a longer time (aged-based replacement), or replace the worst between the parents (regardless of the fitness value of the offspring, but considering the fitness value of the parents). When $\lambda = \mu$ and the new population entirely replaces the previous one, the GA is said to be generational. Another commonly used approach is to always insert in the new generation the best μ solutions out of μ old $+ \lambda$ new points, i.e. $(\mu + 1/2)$ survivor selection. The last approach is said to be elitist, as the best individual (i.e. the elite) is always inserted in the new population. This approach, as well as most fitness based approaches, is oriented towards an exploitative behaviour. Aged base and generational approaches tend to be more exploratory, as unfit solutions are accepted to promote the exploration of unexplored sectors of the search space.

Evolution strategies

ESs are completely randomized frameworks based on the ideas of adaptation an evolution. Modern implementations make use of a probabilistic approach to implement adaptation and drive the evolutionary process. In particular, a distribution density function is continuously updated to model the behaviour of a converging population of candidate solutions. Mutants are randomly sampled from the distribution, and their fitness values are evaluated. A parent-distribution is so evolved, rather than a parent-population or mating pool. The produced individuals (offspring) are used to adapt the distribution to the problem at hand, see (Rechenberg, 1971) and (Eiben and Smith, 2003). The search for extrema is then mainly mutation-driven. Mutants are usually thrown from a multivariate Gaussian distribution, whose statistical parameters (i.e. mean vector and covariance matrix) have to be adjusted at every iteration. Unlike EP (Fogel et al., 1966), both the variation operators are employed, as reported in Figure 5. However, recombination plays a secondary role, that is the production of an offspring individual

to serve as mean vector for the distribution. On top of the coordinates of each individual (control variables), variances and angles for rotating the distribution (strategy parameters) have to be stored in memory. To promote adaptations, parameters are linked to individuals and evolved with them. In this light, in the most general case an individual can be encoded as $\langle \mathbf{x}, \boldsymbol{\sigma}, \boldsymbol{\alpha} \rangle$. This representation is heavy from both the computational side (calculations are needed to convert strategy parameters into a covariance matrix) and the memory side (the memory footprint grows quadratically with n). On the other hand, the beauty of this of framework is that strategy parameters are not to be tuned manually, but automatically adapted to the problem (many ESs are parameterless). It can be stated that ESs are the first self-adaptive EAs ever designed, as the strategy parameters are evolved to find the optimal tuning on the fly. While control parameters are obtained from mutation, strategy parameters only undergo a recombination process. The process can be seen as a loop where good strategy parameters are recombined to updated a covariance matrix, that is then used to sample mutants.

As mentioned, recombination plays only a minor role but it is still required to generate the offspring $\mathbf{x}$, where the distribution is centred. There are two principal schemes: *discrete* and *intermediary* recombination. For each scheme, a *local* and *global* variant exists. Thus, four combinations can be used, namely local or global discrete recombination, and local or global intermediary recombination. It has been found that, in practice, ESs tend to perform best if discrete recombination is performed on the control variables and intermediate recombination on the strategy parameters (Eiben and Smith, 2003). To perform the local variant two parents, let us say $\mathbf{P_1}$ and $\mathbf{P_2}$, are randomly picked up from the population. The two parents are used to implement either the discrete and intermediary scheme. In the local discrete case every component of $\mathbf{x}$ has 50% probability to come from $\mathbf{P_1}$ and 50% probability to come from $\mathbf{P_2}$. In the local intermediary case every component of $\mathbf{x}$ is the arithmetic average of the corresponding component in $\mathbf{P_1}$ and the corresponding component in $\mathbf{P_2}$ (i.e. $\mathbf{x}[i] = \frac{\mathbf{P_1}[i]+\mathbf{P_2}[i]}{2}$). To perform the global variant the two parents are made up of components from the entire populations, chosen by balanced[9] roulette wheel selection. Thus, in the global discrete scheme the offspring $\mathbf{x}$ inherits its components from any member of the population with probability $\frac{1}{\mu}$ (where μ is the population size). In the global intermediary scheme it inherits components which are the average of the components inserted in $\mathbf{P_1}$ and $\mathbf{P_2}$ via balanced RWS.

In order to perform mutation a new mean vector and new covariance matrix must be first worked out. While the control variables $\mathbf{x} = x_1, x_2, \ldots, x_n$ of the offspring individual are simply used as mean vector, the covariance matrix $\mathbf{C}$ is obtained by manipulating the strategy parameters ($\boldsymbol{\sigma}$[10]$= \sigma_1, \sigma_2, ..., \sigma_{n_\sigma}$ and $\boldsymbol{\alpha}$[11]$= \alpha_1, \alpha_2, ..., \alpha_{n_\alpha}$), see (Eiben and Smith, 2003) for details. Finally, the new "mutated"

[9] Equally spaced RWS.

[10] σ represents the standard deviation vector, also called step sizes vector, used to work out the variances on the diagonal of $\mathbf{C}$.

[11] $\boldsymbol{\alpha}$ contains the angles of the rotation matrix for the Normal distribution, which therefore can be used to calculate the cross-correlation between two variables in $\mathbf{C}$.

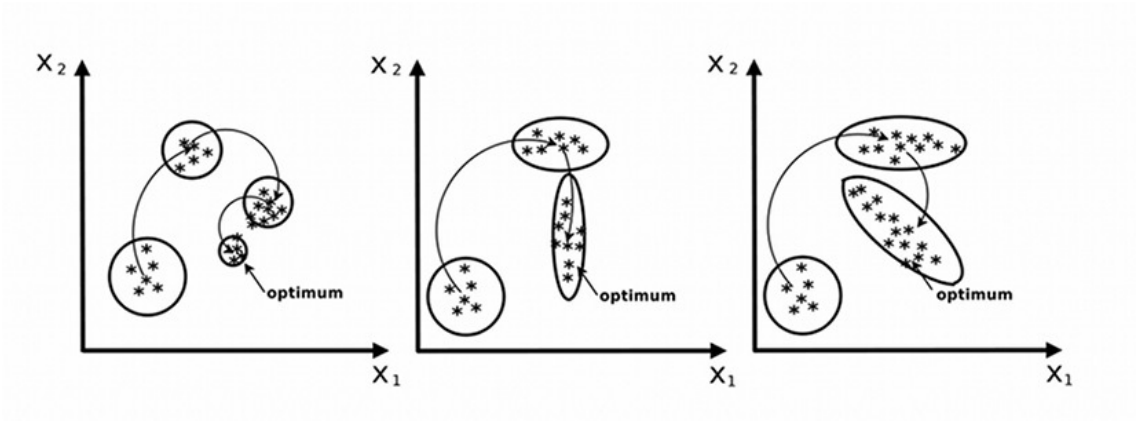


FIGURE 6 Graphical representation of ES mutation operators. Uncorrelated mutation with 1 step size (left), uncorrelated mutation with n step sizes (middle) and correlated mutation wit covariance matrix $\mathbf{C}$ (right). The oval represents the distribution from which points are sampled.

individuals are simply thrown from the distribution:

$$\mathbf{x}^{\mathbf{new}} \sim \mathcal{N}(\mathbf{x}, \mathbf{C})$$

This scheme is efficient at handling ill-conditioned problems, noisy landscapes, and discontinuities. However, it can sometimes be to heavy. Lighter schemes can be implemented by using less strategy parameters. For example, is worth mentioning the *uncorrelated mutation with n step sizes* (i.e. no cross-correlation $\boldsymbol{\alpha}$, $\mathbf{C} = diag\{\sigma_1, \sigma_2, \ldots, \sigma_n\}$) and *uncorrelated mutation with* 1 *step size* (i.e. σ is a scalar value multiplying the identity matrix $\mathbf{I}$, $\mathbf{C} = \sigma\mathbf{I}$). For the sake of clarity, the capabilities of these mutations are graphically shown in Figure 6. As can be seen, the first variant is able to adapt the Gaussian distribution by changing its size and shape. The lack of cross-correlation makes it unable to rotate. In the second variant, instead, only the size can change. Simpler implementations also exist, e.g. the single solution $(1+1)$-ES with $1/5$ rule (Auger, 2009). However, because of their simplicity, they usually return less accurate results.

Finally, there are two principal replacement schemes: (μ, λ) survivor selection (also referred to as comma selection), and $(\mu + \lambda)$ survivor selection (also referred to as plus selection). Unlike what happens in GAs, the first strategy can also have $\lambda > \mu$, and the old population is replaced by choosing the μ fittest individuals out of a pool of λ new points. The second method, preserves old dated solutions (good in dynamic environments) by selecting the best μ individuals out of a set containing all the $\mu + \lambda$ solutions.

The Covariance Matrix Adaptation ES (CMA-ES) is the most significant example of ES. It is designed on a solid and elegant mathematical basis, see (Hansen and Ostermeier, 1996), and is considered a benchmark for real-valued optimization. CMA-ES features several desirable properties, such as lack of problem de-

pendent parameters and invariance to many transformations. It is extremely efficient in solving ill-conditioned and monomodal problems. CMA-ES keeps being used for solving real world applications, see (Iruthayarajan and Baskar, 2010) and (Fang et al., 2011), and for designing MAs as in (Molina et al., 2010a) and (Baggett and Skahill, 2010). Also in this piece of work, it has been involved in the design of PVII, PVI and PX. In particular, the standard CMA-ES with rank-μ-update and weighted recombination, see (Nikolaus and Stefan, 2004), was employed. In this algorithm the sampling of an individual $\mathbf{x_k}$ ($k = 1, 2, \ldots, \lambda$), at a generic generation $g + 1$, is obtained by implementing the following formula:

$$\mathbf{x_k^{(g+1)}} \sim \mathcal{N}\left(\langle \mathbf{x} \rangle_w^{(g)}, (\sigma^g)^2\, \mathbf{C}^{(g)}\right) \tag{4}$$

where $\mathcal{N}\left(\mathbf{m}, \sigma^2\mathbf{C}\right)$ is a multivariate normal distribution of mean vector $\mathbf{m}$, step-size σ, and estimated covariance matrix $\mathbf{C}$. From the implementation point of view, the sampling is made by decomposing the covariance matrix as the product of two components. The first is a diagonal matrix $\mathbf{D}$. The second is an orthonormal matrix $\mathbf{B}$, which is obtained via principal component analysis as indicated in (Hansen and Ostermeier, 2001). The decomposition is done so that $\mathbf{C^(g)} = \mathbf{B^{(g)}D^{(g)}}^2\mathbf{D^{(g)}}^T$. Processing multiple matrices is a slow and computationally expensive (sometimes numerically unstable) procedure, but allows the coding of the sampling routine in the following way: $\mathbf{x_k^{(g+1)}} \sim \langle \mathbf{x} \rangle_w^{(g)} + \sigma^{(g)}\mathbf{B^{(g)}D^{(g)}}\mathcal{N}\left(\emptyset, \mathbf{I}\right)$. With regard to the mean vector $\langle \mathbf{x} \rangle_w^{(g)}$, it is a simple weighted sum of the μ fittest candidate solutions ($\mu \leq \lambda$) of the generation g. This vector corresponds to a recombination result:

$$\langle \mathbf{x} \rangle_w^{(g)} = \sum_{i=1}^{\mu} w_i \mathbf{x}_{i:\lambda}^{g} \tag{5}$$

where $\mathbf{x}_{i:\lambda}^{(g)}$ denotes the i^{th} best individuals at the generation g amongst the λ available, and w_i are weight factors so that $\sum_{i=1}^{\mu} w_i = 1$, see (Nikolaus and Stefan, 2004) for details. In order to work out the covariance matrix $\mathbf{C}$, first, the so called evolution path $\mathbf{P_c}$ has to be considered:

$$\mathbf{p_c}^{(g+1)} = (1 - c_c)\mathbf{p_c}^{(g)} + H_\sigma^{(g+1)}\sqrt{c_c \cdot (2 - c_c)} \cdot \frac{\sqrt{\mu_{eff}}}{\sigma^{(g)}}\left(\langle \mathbf{x} \rangle_w^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)}\right) \tag{6}$$

and used in the formula below:

$$\mathbf{C}^{(g+1)} = (1 - c_{cov})\mathbf{C}^{(g)} + c_{cov} \cdot \frac{1}{\mu_{cov}}\mathbf{p_c}^{(g+1)}\left(\mathbf{p_c}^{(g+1)}\right)^T$$
$$+ c_{cov} \cdot \left(1 - \frac{1}{\mu_{cov}}\right)\sum_{i=1}^{\mu}\left(\mathbf{x}_{i:\lambda}^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)}\right)\left(\mathbf{x}_{i:\lambda}^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)}\right)^T. \tag{7}$$

All the parameters do not depend on the problem at hand and have been accurately pre-tuned, see (Hansen et al., 2003) and (Nikolaus, 2005). For the sake of brevity, a schematic description of the parameters setting is given in Table 2, where the suggested (optimal) values are also reported. It has to be said that some

–problem dependent user defined input parameters–
initialise n, $\langle \mathbf{x} \rangle_w^{(0)}$ and $\sigma^{(0)} \triangleright \sigma^{(0)} = 0.5$, $\langle \mathbf{x} \rangle_w^{(0)} \sim \mathcal{U}(0,1)$ in (Nikolaus and Stefan, 2004)
–strategy parameter setting: selection–
initialise λ, μ, μ_{eff} and $w_{i=1\ldots,\mu}^{(0)}$ as in Table 2 ▷ λ can be modified if needed
–strategy parameter setting: adaptation–
initialise c_c, μ_{cov}, c_σ, d_σ and d_σ as in Table 2
–initialise dynamic (internal) matrices parameters and constants–
$\mathbf{P_c} = \mathbf{P}_\sigma = [\emptyset]$, $\mathbf{B} = \mathbf{D} = \mathbf{I}$, $\mathbf{C} = \mathbf{B} * \mathbf{D} \left(\mathbf{B} * \mathbf{D}\right)^T$
–main loop–
while stop criterion is not met **do**
  sample λ new individuals from distribution ▷ Formula 4
  evaluate individuals and sort them based on their fitness
  update $\langle \mathbf{x} \rangle$ based on a weighted sum of the best μ individuals ▷ Formula 5
  update the evolution paths $\mathbf{P_\sigma}$ and $\mathbf{P_c}$ ▷ Fomula 8 and 6
  update covariance matrix $\mathbf{C}$ and step-size σ consequently ▷ Formula 7 and 9
end while
Output best individual ever found $\mathbf{x_e}$

FIGURE 7 Pseudocode of CMAES

parameters are necessarily auto-generated. Some others, i.e. λ, μ and the initial value for σ, can be customized (as indicated in Figure 7). The function $H_\sigma^{(g+1)}$ is defined such that it returns 1 if

$$\frac{||\mathbf{P}_\sigma^{(\mathbf{g+1})}||}{\sqrt{1-\left(1-c_\sigma^{(g+1)2}\right)}} < \left(1.5 + \frac{1}{n-0.5}\right) E\left(||\mathcal{N}(\emptyset, \mathbf{I})||\right)$$

and 0 otherwise. In the previous formula, n is the dimensionality of the problem, while $E(\ldots)$ the expected value[12]. The matrix $\mathbf{P}_\sigma$ takes the name "conjugate evolution path" and is defined as

$$\mathbf{P}_\sigma^{(\mathbf{g+1})} = (1-c_\sigma) + \sqrt{c_\sigma \cdot (2-c_\sigma)} \underbrace{\mathbf{D}^{(g)}\mathbf{B}^{(g)-1}\mathbf{D}^{(g)T}}_{\mathbf{C}^{(g)-\frac{1}{2}}} \frac{\sqrt{\mu_{eff}}}{\sigma^{(g)}} \left(\langle \mathbf{x} \rangle_w^{(g+1)} - \langle \mathbf{x} \rangle_w^{(g)}\right) \quad (8)$$

where c_σ is a decay factor, see Table 2 for details. This parameter is also used for updating the step-size, as shown below:

$$\sigma^{(g+1)} = \sigma^{(g)} e^{\frac{c_c}{d_\sigma}\left(\frac{||\mathbf{P_c}||}{E||N(\emptyset,I)||}-1\right)}. \quad (9)$$

Subsequently, several variants of CMA-ES have been proposed.

As the standard CMA-ES presents a deterioration of the performances over multimodal functions, a restart mechanism was designed in (Auger and Hansen,

[12] $E\left(||\mathcal{N}(\emptyset, \mathbf{I})||\right)$ is the expectation of the length of the n-dimensional point sampled from $\mathcal{N}(\emptyset, \mathbf{I})$, which is $\approx \sqrt{n}\left(1 - \frac{1}{4n} - \frac{1}{21n^2}\right)$ (Nikolaus and Stefan, 2004).

TABLE 2 Description of parameters in CMA-ES

Parameter	Description	Value
λ	offspring number, new solutions sampled, population size	$\lambda = 4 + \lfloor 3ln\,(n) \rfloor$
μ	parents number, solutions involved in updates of mean value, covariance and step-size	$\mu = \lfloor \frac{\lambda}{2} \rfloor$
$w_{i=1\ldots\mu}$	recombination weights	$w_{i=1\ldots\mu} = \frac{ln(\mu+1)-ln(i)}{\sum_{j=1}^{\mu} ln(\mu+1)-ln(j)}$
μ_{eff}	variance effective selection mass	$\mu_{eff} = \frac{1}{\sum_1^{\mu} w_i^2}$ ($w_{i=1\ldots\mu}$ are chosen so that $\mu_{eff} \approx \frac{\lambda}{4}$)
c_c	decay rate for the evolution path	$c_c = \frac{4}{n+4}$
μ_{cov}	learning coefficient for the covariance matrix	$\mu_{cov} = \mu_{eff}$
c_{cov}	learning rate for the covariance matrix	$c_{cov} = \frac{1}{\mu_{cov}} \frac{2}{\left(n+\sqrt{2}\right)^2} + \left(1 - \frac{1}{\mu_{cov}}\right) min\left(1, \frac{2\mu_{eff}-1}{(n+2)^2+\mu_{eff}}\right)$
c_σ	decay rate for the conjugate evolution path	$c_\sigma = \frac{\mu_{eff}+2}{n+\mu_{eff}+3}$
d_σ	damping parameter for σ-change	$d_\sigma = 1 + 2max\left(0, \sqrt{\frac{\mu_{eff}-1}{1+n}} - 1\right) + c_\sigma$

2005). According to this scheme, the population size has to be increased after every restart. This variant is less prone to premature converge and better handles multiple optima. It is usually refereed to as IPOP-CMA-ES, but is also known under the name G-CMA-ES. Further attempts heave been made to improve upon CMA-ES by tuning the population size on the go. Some interesting implementations have been tested in (Loshchilov, 2013), where also novel step size updating formulas are proposed. Results have shown that restarting the search make new variants outperform the standard implementation on 23 multimodal functions from the test suite in (Liang et al., 2013).

Conversely, the study in (Ros and Hansen, 2008) suggests the use of the uncorrelated mutation with n step sizes to tackle separable problems (sep-CMA-ES). Numerical results empirically showed the suitability of this mutation for separable domains. It was also noted that sep-CMA-ES performs better than standard CMA-ES over large scale problems (n larger than 100), regardless of the separability of the problem.

Memetic algorithms, memetic computing and Hyper-heuristics

Advantages of hybridizing population based algorithms with LS routines, as those described in Section 2.2.1, have been already discussed in this piece of work.

Any time the population based component is an EA, and the LS is performed by activating a local searcher within the evolutionary cycle, the optimizer can be refer to as an MA.

The term "memetic" was introduced for the first time by Moscato (1989), referring to the concept of *meme* coined by Richard Dawkins as: *the basic unit of cultural transmission, or imitation* (Dawkins, 1976). Thus, the main metaphor behind the MA does not rely on biological evolution, but on Dawkins' Universal Darwinism theory. This can be seen as an extension of the Darwinian theory in that evolution is considered to take place also in all those complex systems exhibiting the processes of inheritance, variation and selection. This happens e.g. with elements of culture that pass on to new generations. As an idea, a promising meme can be shared, exchanged within a community (individual of the population in MAs, different operators in MC), adapted or improved.

Regardless of the underlying metaphor, the idea of exchanging information between different operators has attracted the attention of researchers. Thus, it has been studied under different point of views. Studies have been carried out to understand when and how the LS can be applied, how to improve the coordination logic of the single components, etc. The MA concept has been quickly extended. As example, the term MC was introduced in optimization to refer to all those cases where interacting agents (memes) are used to face optimization tasks. Such memes were defined as *units of information encoded in computational representations for the purpose of problem solving* (Ong et al., 2010).

In this light, all hybrid algorithms can be seen as MC implementations. Moreover, PSO and DE frameworks equipped with LS can also be referred to as MC algorithms. MC can be seen a "superclass" that can instantiate MA objects. Unfortunately, a great deal of publications in the literature stress the fact that is important to distinguish between MA and MC, see e.g. (Moscato, 1989) and (Hart et al., 2004), thus forcing designers to refer to rigid definitions that do not have a valuable and quantifiable importance from the scientific point of view. As example, an optimizer designed by adding LS routines within a DE structure is based on the same idea of former MAs, even though DE is not considered to be an EA. From the point of view of the author of this thesis, it is irrelevant to discuss about the MC rather than MA nature of such hybrid optimizers, and disquisitions about new metaphor-driven optimization approaches should be discouraged by the scientific community.

Early implementations of MAs exist can be found in the literature before the definition given by Moscato (1989). Such algorithms were referred to as Lamarckian-EAs and Baldwinian-EAs. As the name could suggest, there is a slight but crucial difference between the two approaches. The former makes use of LS to improve upon the genotype of some individuals, while in the latter only the phenotype gets modified, see (Geoffry E. Hinton and, 1987) and (Whitley et al., 1994). Subsequently, the Genetic Local Search Algorithms were proposed (García-Martínez and Lozano, 2008). A standard GA (Whitley, 1994) was usually equipped with hill-climber routines as shown in (García-Martínez and Lozano, 2008).

As the memetic approach became more employed, the problem of coordinating the operators included in the MA arose. The Meta-Lamarckian Learning mechanism, see e.g. (Ong and Keane, 2004) and (Neri et al., 2012), was introduced to tackle this problem by assigning to each local searcher a probability to be executed. At every iteration, the probability of a specific local searcher his updated on the basis of its previous success. The operator to be run is then selected via RWS. Self-adaptive and Co-evolutionary MAs (Krasnogor and Smith, 2005), (Smith, 2007), and (Smith, 2012), made use of a different mechanisms in which a rule-based representation of LS is co-adapted alongside candidate solutions.

In (Molina et al., 2010a), the LS chains mechanism is instead used to better exploit the potentialities of the local searcher. The proposed algorithm, Memetic Algorithm based on Local Search Chains (MACh), is made up of an instance of a steady-state GA, and several instances of the LS routine (i.e. CMA-ES in this case). In order to enhance exploration, the GA was equipped with the Negative Assortative Mating selection strategy (Fernandes and Rosa, 2001), described in Section 2.3.1, with the BGA mutation displayed in Formula 3, and with the BLX-α crossover, see Formula 2. Individuals undergo LS for a prefixed number of FEs. The CMA-ES is applied by centring its distribution on the solution to be refined. After the application, the CMA-ES instance is frozen, so that if the same individual is selected again the very same instance is unblocked and can start over with its previous setting. This means that parameters, needed to work out the covariance matrices of each CMA-ES instance, have to be kept stored in memory during the optimization process. Numerical results show great performances over several test functions. However, there could be, potentially, a CMA-ES instance for every candidate solution. Evolving a population of CMA-ES objects is computationally expensive. From the memory occupation point of view, it is extremely onerous. Thus, the described approach is not suitable for LSOPs. For this reason, a lighter version was designed to address LSOPs, see (Molina et al., 2010b). This variant makes use of the Solis-Wets method to replace CMA-ES in performing the LS. Obviously, the lighter variant is more versatile and usable in many real-world applications, but cannot compete with the previous one in terms of numerical results.

As antithesis of the previous complex and heavy algorithm, a simple and light MC optimizer was presented in (Iacca et al., 2012a). The so called 3SOME algorithm makes the point that also minimalistic single-solution approaches can display a performance which is as good as that one of a complex population-based structures, if the algorithmic design is done carefully. The three components forming 3SOME, are neither complex population based algorithms, nor efficient optimizers that can be used on their own. However, their efficient coordination displayed good numerical results over benchmark functions and real world applications, see e.g. (Iacca et al., 2013).The success of 3SOME is determined by the fact that the employed operators are not redundant and play complementary roles. Their coordination, shown in Figure 9, is simple. It follows a strong logic based on the bottom-up approach depicted in Figure 8, in order to gradually tackle a black-box problem. In order to reach the final goal, i.e. find

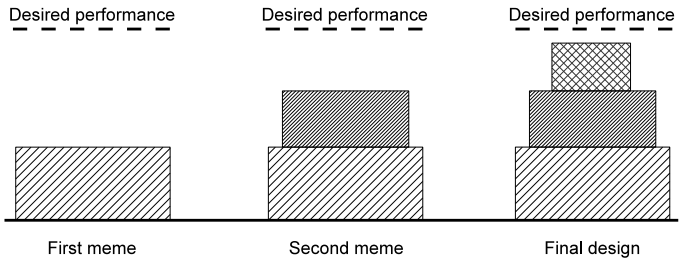


FIGURE 8 Bottom-up design strategy.

the global optimum, the algorithm keeps going through three stages. Such stages are unnecessarily repeated in the same order that is followed the first time. In the first stage, namely "long distance exploration", the L operator performs a random exploration of D. Unlike pure random-walk algorithms, L also exchanges a small portion of the genetic heritage of the newly randomly sampled solution with the elite solution. This inheritance mechanisms is achieved via exponential crossover (Storn and Price, 1995), see Figure 15, and has proven to work nicely in several cases, e.g. (Caraffini et al., 2013) and (Caraffini et al., 2013b). L is stopped as soon as a fitter position if found in D. This solution is passed on to the M operator to undergo the second stage: "middle distance exploration". Once the second stage is activated, it is run as long as M is successful, i.e. produces a fitter solution than the current one. This operator perform the same search logic of L, but with a strong bias towards the elite (elevate crossover rate) and within a narrow portion of D. In details, the random sampling is carried out within a hypercube centred on the elite, whose edge has a width equal to the 20% of the width of D. The final stage, i.e. "short distance exploration", further refines the current solution by running the S algorithm (described in Section 2.2.1). Figure 9 shows the coordination logic of the three stages. The three operators are not simply iterated: if S fails at refining the solution than a new exploratory phase is needed and L is run, otherwise it is skipped. For the sake of clarity it must be said that Figure 9, as well as other figures, follows the representation method introduced in (Caraffini et al., 2013b). The Memetic Node (MN) represents a decisional state. Usually, a MN shows a probability of moving to a state rather than another one. For 3SOME, the decision is deterministically made according to the fitness value of the refined solution. Thus, the algorithm does not know what the next stage will be before that S returns its solution. This coordination logic is simple, and has shown to be as efficient as meta-lamarckian learning and other adaptive logics (Neri et al., 2012). Further studies confirmed the efficiency of this memetic approach and obtained good results with simple modifications. For example, the second stage get sensibly more efficient if the hypercube is progressively shrunk (Poikolainen et al., 2013). Moreover, performances can be significantly increased over separable problems by slightly modifying M and S (Poikolainen et al., 2012; Caraffini et al., 2012b).

The success of 3SOME has been followed by a number of attempt of designing single solution optimizers based on the same principle of simplicity. Se-

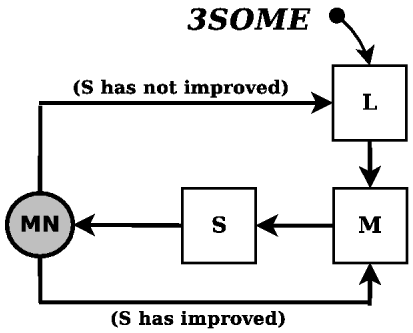


FIGURE 9 3SOME coordination logic.

riously simple algorithms have been designed and proven to perform extremely well, also on large scale problems. The Re-sampled Search (RS) algorithm, for instance, is an efficient implementation of iterated local search (Caraffini et al., 2013a). It can be seen as an extreme simplification of the 3SOME algorithm, and performs even better that its predecessor on large scale problems. The M operator turned up to have a minor impact on the overall performance of 3SOME, see (Caraffini et al., 2012a), hence it was removed to obtain a simpler structure. Moreover, it was clear from (Caraffini et al., 2012a) that such good performances displayed by 3SOME were the outcome of its efficient coordination logic rather than the the merit of each single operators. Therefore, there was no need of replacing S with a more complex LS to design an efficient variant. On the contrary, both S and L were further simplified to save computational budget. S was equipped with an exit condition on the exploratory radius, while L was reduced to a single random sampling within D. The RS algorithm was thus designed by simplifying an existing optimizer, in contrast with the common practice of adding extra components. An improvement in the performance of RS was obtained by simply employing the exponential crossover, as in L, after the random sampling. This variant, namely Re-sampled Inheritance Search (RIS) performs superbly on large scale problems, and can spot good solutions even with a limited computational budget (Caraffini et al., 2013). Finally, the Very Intelligent Single Particle Optimization (VISPO) further confirmed that large scale problems can be handled with seriously simple algorithmic structures. VISPO stochastically perturbs a single design variable at a time for a fixed amount of FEs, and with decreasing magnitude. When the exploratory radius is too short, or the search is unsuccessful, a restart procedure occurs. Details can be found in (Iacca et al., 2013).

An interesting algorithmic structure was proposed in (Caraffini et al., 2013b). In the Paralle Memetic Structure (PMS) algorithm, the L operator can be followed, with equal probability, by either S or the Rosenbrock method. Hence, after exploration, LS can take place according to two different logics. The search space is better covered as diagonal and orthogonal paths are exploited. A boost in the average performance was noted with respect to 3SOME, as PMS is more versatile and robust.

In SPAM (Caraffini et al., 2014), a preliminary analysis of the landscape was

performed to enhance PMS. CMA-ES is executed for a limited amount of FEs, and the covariance matrix evaluated to work out the degree of linkage between the variables. As can be seen in Figure 20, such information is used to adapt the activation probability for S and the Rosenbrock method to the problem. The SPAM framework makes use of simple operators to carry out the actual optimization process, but requires a heavy component, i.e. CMA-ES, to process the covariance matrix. This philosophy is in contrast with some popular solutions that can be found in the literature. As instance, the portfolios approach to algorithms selection consists of a set of several algorithms, picked up according to a simple criterion (Vrugt et al., 2009; Peng et al., 2010). This is similar to the hyper-heuristic approach, where multiple heuristics (algorithms, operators, etc.) are coordinated by a supervisor (Özcan et al., 2008; Burke et al., 2010). Also in this case, the approach is more oriented towards selecting an algorithm every time is needed during the optimization process, rather than fixing the algorithmic structure (e.g. activation probabilities for operators) with a preliminary procedure. In this light, SPAM is quite peculiar as differing from adaptive MC algorithms too. In effect, the latter adapt their parameters during the optimization process. Conversely, SPAM analyzes the landscape to "design" an optimizer by tuning some parameters in advance. Potentially, in particular cases the outcome of the preliminary analysis can result into an iterated local search made up of only L+S or L+Rosenbrock. Thus, if the linkage (correlation) between the variables is nearly 0, an algorithm similar to RIS is used. Conversely, a high correlation is handled by activating only Rosenbrock. All the intermediary cases can occur according to different degrees of correlation amongst the variables. In these cases, a MC with two LS routines is employed. The method proposed for SPAM has a great potential. Researchers talk about automatic generation of optimization algorithms, see (Ong et al., 2009) and (Ong et al., 2010), but SPAM is the first actual implementation for real-valued domains. In fact, other ideas have been presented e.g. in (Meuth et al., 2009; Zhu et al., 2010) and also in (Hoos, 2012; Wu et al., 2012), but they have not been formalized in technical terms yet. If software platforms for the automatic design of optimization algorithms already exist for combinatorial domains, e.g. the SATzilla portfolio-based selection platform (Xu et al., 2008; Hamadi et al., 2012), or the software package F-RACE and its iterated version IRACE (Birattari et al., 2010; Lopez-Ibanez et al., 2011), only little has been done for the real-valued case. A proper software platform capable of designing optimizers for real-valued problems is yet to come. The SPAM framework could be enhanced with more advanced fitness analysis methods, as those in (Malan and Engelbrecht, 2013) and (Malan and Engelbrecht, 2014), to combine together more than just two local searchers.

Regardless of the definition, MC and hyper-heuristics have a lot in common. Memetic computing is an umbrella name, and most algorithms can be seen from the MC perspective. The same can be said for hyper-heuristics. In the latter, the emphasis is put in the coordination logic. The aim of hyper-heuristic approaches is to optimize the coordination of multiple optimization algorithms, rather than trying to optimize the actual problem. The search is taken to a higher level and

the optimization takes pace in the search space of heuristics. However, this means that in practise some sort of adaptive logic must be adopted to supervise the single algorithms. Similar logics are used in a MA to guarantee a fair balance between exploration and exploitation, or in a MC approach to run the most appropriate operator. The hyper-heuristic point of view resulted very useful in discrete and combinatorial problems, where the gradient information cannot be used to guide the search. For this reason, moving the attention to an equivalent, but different, search space (i.e. of heuristics) presents its advantages. Great results have been obtained by adopting this approach for timetabling and rostering problems, see (Cowling et al., 2000) and (Burke et al., 2003a) respectively. Successful supervisory schemes are e.g. those employing the so called choice function (Cowling et al., 2000), or a combination of choice function and randomized criteria (Kendall et al., 2002). More sophisticated approaches make use of reinforcement learning as in (Burke et al., 2003a) and (Dowsland et al., 2007), but also memory-based mechanisms, see (Burke and Bykov, 2008). The multi-agents approach (Acampora et al., 2011a,b) also displayed a good performance. In Chapter 4.3.3 of this piece of work , a novel hyper-heuristic structures is proposed by improving the coordination logic proposed in (Caraffini et al., 2014) for the SPAM algorithm.

2.3.2 Swarm Intelligence

A number of popular nature-inspired optimizers follow the SI logic, which was firstly introduced in robotics (Beni and Wang, 1989), and then adopted in all areas of AI, including CIO. The general structure of a SI based optimizer is given in Figure 10. The main idea is to mimic the behaviour of groups of animals, e.g. a flock of birds crossing the sky or a school of fish turning together in the sea. These communities display remarkable self-organisation skills, despite the lack of a central coordination logic. Optimizers can simulate this behaviour. A "swarm" of solutions move in the search space. A single solution can only wander in the search space, keeping track of its best e worst position. However, by exchanging this information within its neighborhood, solutions can be perturbed more efficiently. Thus, a collective intelligence can arise from non intelligent units. Several approaches exploit this concept. Examples are: Artificial Bee Colony (ABC), see (Karaboga and Basturk, 2007) and (Pham and Castellani, 2009), Bacterial Foraging Optimization (BFO), see (Passino, 2002), and Ant Colony Optimization (ACO), as in (Colorni et al., 1991) and (Socha and Dorigo, 2008). Amongst them stood the PSO algorithm by Kennedy and Eberhart (1995), which is undoubtedly the most used.

Particle swarm optimization

In PSO a set (or swarm) of P solutions (i.e. the particles) are perturbed taking into consideration the history of their neighbors, and of the entire swarm. In order to implement this logic for each particle $p = 1, 2, \ldots, P$ is required to store: cur-

rent position ($\mathbf{x_p}$), personal best position ($\mathbf{x_p^b}$) and personal worst position ($\mathbf{x_p^w}$[13]). In addition to these positions, a velocity[14] vector $\mathbf{v_p}$ is also associated to each particle. So, in the most general case, each particle is associated with its design variables, plus 3 more vectors: $\left\langle \mathbf{x_p}, \mathbf{x_p^b}, \mathbf{x_p^w}, \mathbf{v_p} \right\rangle$. The first three vectors contain positions, while the latter contains perturbation parameters. In order to perturb a particle, $\mathbf{v_p}$ is first updated according to Formula 10, and then used to move $\mathbf{x_p}$ as shown in Formula 11. The fitness value of the particle in its new position has to be evaluated to understand whether or not $\mathbf{x_p^b}$ and $\mathbf{x_p^w}$ are changed. The particle with the fittest personal best position is copied into the global best position $\mathbf{x_{gb}}$. Such point is always kept up to date and returned as global solution at the end of the optimization process.

$$\mathbf{v_p'} = \phi_1 \mathbf{v_p} + \phi_2 \left(\mathbf{x_p^b} - \mathbf{x_p} \right) + \phi_3 \left(\mathbf{x_{gb}} - \mathbf{x_p} \right) \tag{10}$$

$$\mathbf{x_p'} = \mathbf{x_p} + \mathbf{v_p} \tag{11}$$

With reference to Formula 10, ϕ_1, ϕ_2, and ϕ_3 are three weights, usually containing a random component (Clerc and Kennedy, 2002). Since performances heavily depend on the choice made for these problem depending parameters, see (Yuhui and Eberhart, 1998), a great deal of research have been done to make them self-adaptive, e.g. in (Zheng et al., 2003) a dynamic time-decreasing inertia weight ϕ_1 is employed. It can be noted that there are two moves embedded in Formula 10: a first one towards the personal best position of the particle being perturbed, and a second one towards the global best solution. These two forces and the old velocity vector are weighted and summed up in order to generate a new velocity vector. Thus, the perturbation takes into consideration local and global information. The particle is moved and left in the new position regardless of the corresponding fitness value. This is not a problem as $\mathbf{x_p^b}$ and $\mathbf{x_p^w}$ keep track of best and worst explored positions. This logic can be seen as the evolution of multiple populations. One is used to explore D and contains the particles. This population does not converge. Conversely, the one containing the local best positions converges to the optimum. It is evident that PSO does not need selection: parents do not have to be picked up as each velocity vector is applied to its particle, and not to a selected one from the swarm. Moreover, replacement does not happen in the classic way, as particles are moved by adding a component to their previous position, which is lost. Replacement occurs only for the local best positions, that can be seen as the real population of PSO. Solutions are usually perturbed one at a time from the first to the last one. If an improvement occurs, it is immediately registered by replacing the corresponding local best vector. This logic is called 1-to-1 spawning.

Newer perturbations schemes can be found in the literature and many variations on the theme of PSO have been designed. As instance, a compact variant

[13] This vector is rarely used.

[14] It must not be confused with the physical meaning of this term, strictly speaking it is a displacement rather than a velocity.

Initialisation ▷ Randomly sample initial swarm
while *Condition on budget* **do**
  for each $\mathbf{x} \in$ *Swarm* **do**
    Update perturbation parameters and apply perturbation: $\mathbf{x} \Longrightarrow \mathbf{x}'$
    Update swarm
  end for
end while
Output *Best Individual*

FIGURE 10 Pseudocode of swarm intelligence optimization

(it does not employ a proper swarm but a probabilistic representation of it) is presented in (Neri et al., 2013), while multiple perturbation strategies have been used together in the well-known Frankenstein's PSO (Montes de Oca et al., 2009).

Amongst the most representative PSO variants, is worth mentioning the Comprehensive Learning PSO (CLPSO). As the name suggests, CLPSO uses a learning strategy whereby all the particles take part and share their history to update t velocity vector as below:

$$\mathbf{v_p} = \phi_1 \mathbf{v_p} + \phi_2 \mathbf{U} \times \left(\mathbf{x_p^{rb}} - \mathbf{x_p} \right) \tag{12}$$

where $\mathbf{U}$ is a $n \times n$ matrix of uniform distributed random numbers, $\mathbf{x_p^{rb}}$ is a vector built up by randomly sampling components from all the local bests vectors $\mathbf{x_p^b}$. The sampling procedure makes use of a threshold, P_c, which has to be generated for each dimension $i = 1, 2, \ldots, n$:

$$P_{c-i} = 0.05 + 0.45 \cdot \frac{e^{\frac{10(i-1)}{P-1}} - 1}{e^{10} - 1} \tag{13}$$

where P is the swarm size (number of particles). A random number is drawn and, if higher than P_c, the $i-th$ component of the corresponding local best particle is taken. Otherwise, two randomly selected local best positions are compared according to their fitness value. The fittest of the two donates the $i-th$ component to $\mathbf{x_p^{rb}}$. This scheme has shown to be versatile and efficient for addressing up to 50 design variables (Liang et al., 2006).

The Cooperatively Coevolving Particle Swarms for large scale optimization (CCPSO2), proposed in (Li and Yao, 2012), features an interesting sub-grouping mechanism for decomposing the swarm in lower-dimensional swarms. These sub-swarms are evolved at the same time, so facing easier sub-problems, and return partial components of the global solutions of the LSOP. An example of the sub-grouping routine is graphically given in Figure 11, where N is the dimensionality of the problems, S is a randomly chosen admissible divisor (dimensionality of each sub-swarm) and K is the number of sub-swarms, i.e. $k = \frac{N}{S}$. Unexpectedly, this logic has shown to perform well also for low-dimensional problems (e.g. 30, 50 and 100 design variables). The perturbation scheme slightly differs

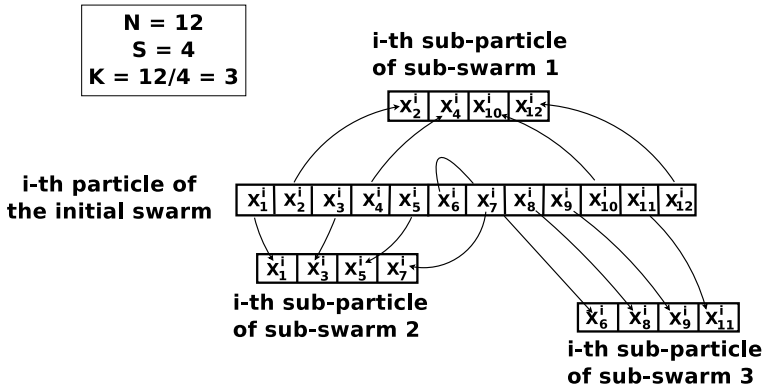


FIGURE 11 Graphical representation of the sub-grouping strategy in CCPSO2 for large scale optimization. N is the number of variables, S is the number of variables per sub-particle, and K is the number of sub-particles.

from the classic one for PSO algorithms, since a Gaussian/Cauchy perturbation is directly applied to the position vector:

$$\mathbf{x}_i = \begin{cases} \mathbf{x^b}_i + \mathcal{C}(0,1)\,|\mathbf{x^b}_i - \mathbf{x^{gb}}_i| & if\ \mathcal{U}(0,1) < p \\ \mathbf{x^{gb}}_i + \mathcal{N}(0,1)\,|\mathbf{x^b}_i - \mathbf{x_i^{gb}}| & otherwise \end{cases} \qquad i = 1,2,\ldots,S \tag{14}$$

where the probability p can be set equal to 0.5, $\mathbf{x^b}_i$ refers to the personal best of the i^{th} sub-particle, and $\mathbf{x_i^{gb}}$ to the global best position within a small neighborhood of the i^{th} sub-particle (i.e. fittest among i^{th} particle and its immediate left and right neighbors). Fitness functional calls are performed by reconstructing the design vector with the concatenation of all the components coming from the sub-swarms.

Finally, (Zhen et al., 2010) introduces an extraordinarily simple single-solution variant, namely Intelligent Single Particle Optimizer (ISPO). Thanks to its minimalistic structure, the algorithm could be successfully used to perform memory-saving optimization in PIII. ISPO simply perturbs the n design variables ($i = 1,2,\ldots,n$) of a single particle for a number $H = 30$ of attempts in a row. The following velocity vector is used to perturb the particle:

$$v_i^{h+1} = \frac{A}{(h+1)^P} \cdot \mathcal{U}(-0.5, 0.5) + B \cdot L^h, \;\; h = 1,2,\ldots,H \tag{15}$$

where A, P, and B are three problem dependent parameters called acceleration, acceleration power factor, and learning coefficient. The learning factor L is instead initially set to zero and then updated after each perturbation t, i.e. $x\,[i]^{t+1} = x\,[i]^t + v[i]^{t+1}$. If the new particle improves upon (or equals) the old one, then $L^{h+1} = v^{h+1}[i]$, otherwise (if $L \neq 0$) $L^{h+1} = \frac{L^h}{S_F}$, where S_F is a shrinking factor. L is reinitialized to zero if its absolute value gets too small, i.e $|L| < E = 10^{-5}$, due to a sequence of unsuccessful perturbations. This mechanism ensures a randomised search along each dimension, whose exploratory radius is larger at the beginning of the optimization process and progressively shrinks as h is increased.

2.3.3 Simulated annealing

A popular single solution optimizer for global optimization is the SA algorithm. Unlike the aforementioned ISPO, SA does not focus on a single design variable at a time. Conversely, all the design variables are changed through a stochastic perturbation. SA is based on a working principle inspired from metallurgy. When a metal is heated and subsequently cooled down, its crystal structure is strengthened as defects fade. At high temperatures, atoms are not confined within the crystal structure but free to move chaotically (exploration of the landscape). They settle down as soon as the temperature decreases (exploitation). By controlling the cooling, atoms are placed in a configuration of minimum energy within the new crystal. A similar behaviour is obtained in SA by means of the procedure reported in Figure 12. Atoms represent the design variables.Their motion is the outcome of a stochastic Gaussian or Cauchy perturbation. Local minima make the search difficult, and in order to reach the position of minimum energy the heating and cooling procedures have to be alternated several times. Poor configurations may occur. These configurations are accepted according to a stochastic rule, see Figure 12, that tends to promote exploration at the early stages of the optimization process, to gradually became more selective. Thus, towards the end of the optimization process, a new solution is accepted only if fitter than the previous one. This is achieved by decreasing the probability of accepting bad configurations exponentially: $exp\left\{-\frac{f(\mathbf{x_k})-f(\mathbf{x_{k-1}})}{T_k}\right\}$. The temperature T_k is commonly updated as follows:

$$T_k = \frac{T_0}{ln\,(1+k)} \tag{16}$$

where T_0 is the initial (or maximum) temperature. SA was first proposed for combinatorial optimization in (Kirkpatrick et al., 1983), but it was quickly adapted for real-valued optimization.

An interesting SA is presented in (Xinchao, 2011), under the name nuSA (non uniform SA). The name refers to the use of the following perturbation:

$$\mathbf{x_k}[i] = \begin{cases} \mathbf{x_{k-1}}[\mathbf{i}] + \Delta\left(k, \mathbf{x^U}[i] - \mathbf{x_k}[i]\right) & if\ \mathcal{U}\,(0,1) > 0.5 \\ \mathbf{x_{k-1}}[\mathbf{i}] - \Delta\left(k, \mathbf{x_k}[i] - \mathbf{x^L}[i]\right) & otherwise \end{cases} \quad i = 1, 2, \ldots, n \tag{17}$$

that depends on the generation counter k. As k increases, the perturbation acts differently. At the early the stages of the optimization process Formula 17 approximates a Cauchy perturbation, i.e. long steps are made within D, and subsequently a Gaussian perturbation, i.e. locally distributed around its mean value. This behaviour is guaranteed by $\Delta\,(k,y) = y \cdot \left(1 - \mathcal{U}\,(0,1)^{\left(1-\frac{k}{K}\right)^b}\right)$, which converges towards zero as k increases. The b parameter is problem dependent and takes integers values, e.g. $b = 5$ in (Xinchao, 2011).The following geometric progression is used for cool-down:

$$T_k = \alpha \cdot T_{k-1} \tag{18}$$

$k \leftarrow 0$
$\mathbf{x_k} \leftarrow randomSampling\,(1, N, \mathcal{D})$
$\mathbf{x_{sa}} \leftarrow \mathbf{x_{sa}}$
while *budget condition* **do**
  $k \leftarrow k + 1$
  $\mathbf{x_k} \leftarrow createNeighborSolution\,(k)$ ▷ e.g. Equation 17 in nuSA
  $T_k \leftarrow cooling\,(k)$ ▷ e.g. Equation 16 and 18 for SA ans nuSA respectively
  if $f\,(\mathbf{x_k}) \leq f\,(\mathbf{x_{k-1}})\ ||\ e^{-\frac{f(\mathbf{x_k})-f(\mathbf{x_{k-1}})}{T_k}} < \mathcal{U}\,(0,1)$ **then**
    if $f\,(\mathbf{x_k}) \leq f\,(\mathbf{x_{sa}})$ **then**
      $\mathbf{x_{sa}} \leftarrow \mathbf{x_k}$
    end if
  else
    $\mathbf{x_k} \leftarrow \mathbf{x_{k-1}}$
  end if
end while
Output $\mathbf{x_{sa}}$

FIGURE 12 Pseudocode of simulated annealing

with $\alpha \in [0.9, 0.99]$. The initial temperature T_0 is automatically tuned with a simple procedure: a set of 10 points is uniformly sampled and the following value is worked out $T_0 = -\frac{f(\mathbf{x_{worst}})-f(\mathbf{x_{best}})}{ln(\xi)}$. The parameter ξ is the initial acceptance (a value of $\xi = 0.9$ assures a exploration of D by accepting mediocre solutions). The vectors $\mathbf{x_{best}}$ and $\mathbf{x_{worst}}$ are respectively the fittest and the worst solution in the set.

2.3.4 Differential evolution

DE was designed by Storn and Price (1995) in the attempt to solve a fitting problem. It was designed on the simple idea of computing a scaled difference between randomly picked individuals to generate new solutions. An idea that turned up to be a very efficient mutation strategy. DE displayed outstanding GS capability, flexibility, and reliability. This algorithm follows the scheme reported in Figure 13, see (Rainer and Kenneth, 1997) for technical details. It can be noted that DE implementation is simple with respect to other algorithms, as its structure is linear. This was a crucial factor promoting the use of this optimizer in various fields of engineering, see e.g. (Storn, 1996), (Storn, 1999), (Storn, 2005) and (Price et al., 2005). Individuals are perturbed one at a time from the first one to the last one, according to the 1-to-1 spawning scheme from SI. No parent selection is needed. From each individual a new one is generated, but unlike PSO is temporarily saved and and left unused. Only when a complete new generation is produced, the new individuals are compared with the ones located in the same position. Replacement occurs if a new individual displays a better fitness value than its predecessor. Variation operators follow an unconventional order,

$g \leftarrow 1$ ▷ First generation
$\mathbf{Pop^g} \leftarrow$ *randomly sample M n-dimensional individuals within* $\mathbf{D}$
$\mathbf{x_{best}} \leftarrow$ *fittest individual* $\in \mathbf{Pop^g}$
while *Condition on budget* **do**
  for each $\mathbf{x_j} \in \mathbf{Pop^g}$ **do** ▷ $j = 0, 1, 2, ..., M$
    $\mathbf{x_m} \leftarrow$*Mutation* ▷ e.g. Formula 19 in (Storn and Price, 1995)
    $\mathbf{x_{off}} \leftarrow$*CrossOver*$(\mathbf{x_j}, \mathbf{x_m})$ ▷ e.g. Figure 14 in (Storn and Price, 1995)
    if $f(\mathbf{x_{off}}) \leq f(\mathbf{x_j})$ **then**
      $\mathbf{Pop^{g+1}}[j] \leftarrow \mathbf{x_{off}}$
    else
      $\mathbf{Pop^{g+1}}[j] \leftarrow \mathbf{x_j}$
    end if
  end for
  $g \leftarrow g + 1$ ▷ Replace the old with the new generation
  $\mathbf{x_{best}} \leftarrow$ *fittest individual* $\in \mathbf{Pop^g}$ ▷ update best individual
end while
Output *Best Individual* $\mathbf{x_{best}}$

FIGURE 13 Pseudocode of differential evolution

as mutation take place before crossover. This is the peculiarity of DE. During the mutation process Individuals from the population are linearly combined to detect promising directions. This approach distinguishes DE from other optimizers, as linear combinations are usually employed to perform crossover in real-valued GAs. The outcome of mutation is referred to as "mutant" vector. The latter is used to generate an "offspring" solution. This is obtained by breeding the individual under consideration with the mutant vector by means of crossover operator.

The former implementation from 1995, made use of the so called *rand/1* mutation followed by the "binomial" crossover. The *rand/1* mutation is performed by implementing Formula 19. The three points $\mathbf{x_{r_1}}$, $\mathbf{x_{r_2}}$ and $\mathbf{x_{r_3}}$ have to be picked up so that $r_3 \neq r_2 \neq r_1 \neq j$, and linearly combined by using a scale factor F properly chosen in $[0, 2]$. The mutant $\mathbf{x_m}$ can be seen as the summation of two component: $\mathbf{x_{r_1}}$ and a difference vector, whose aim is to move $\mathbf{x_{r_1}}$ towards the optimum. With reference to Figure 13, $\mathbf{x_m}$ is thereafter mated with the j^{th} individual via binomial crossover. The latter, see Figure 14, exchanges design variables according to a given probability CR. A generation is completed after m executions of the sequence mutation→crossover, with m being the population size ($j = 1, 2 \ldots, m$).

A second crossover routine, namely "exponential" crossover, was also presented by Storn and Price. The latter, exchanges a burst of consecutive design variables whose length depends on the CR value. The probability of having one more gene in the exchanged sequence follows the geometric progression, and decays exponentially (from which the name). Implementation details are given in Figure 15.

The DE community has been growing quickly during the last 2 decades. Several alternative mutation formulas have been proposed by researchers in the

filed. The most employed are listed below:

- *rand/1*:

$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) \tag{19}$$

- *best/1*:

$$\mathbf{x_m} = \mathbf{x_{best}} + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right) \tag{20}$$

- *rand/2*:

$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) + F\left(\mathbf{x_{r_4}} - \mathbf{x_{r_5}}\right) \tag{21}$$

- *best/2*:

$$\mathbf{x_m} = \mathbf{x_{best}} + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right) + F\left(\mathbf{x}_{r_3} - \mathbf{x_{r_4}}\right) \tag{22}$$

- *current-to-best/1*:

$$\mathbf{x_m} = \mathbf{x_j} + F\left(\mathbf{x_{best}} - \mathbf{x_j}\right) + F\left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}}\right) \tag{23}$$

- *current-to-rand/1*:

$$\mathbf{x_m} = \mathbf{x_j} + K\left(\mathbf{x_{r_1}} - \mathbf{x_j}\right) + F^{'}\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) \tag{24}$$

- *rand-to-best/1*:

$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{best}} - \mathbf{x_j}\right) + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) \tag{25}$$

- *rand-to-best/2*:

$$\mathbf{x_m} = \mathbf{x_{r_1}} + F\left(\mathbf{x_{best}} - \mathbf{x_j}\right) + F\left(\mathbf{x_{r_2}} - \mathbf{x_{r_3}}\right) + F\left(\mathbf{x_{r_4}} - \mathbf{x_{r_5}}\right) \tag{26}$$

A particular DE implementation can be obtained by combining the aforementioned mutations and crossover strategies together. Despite several crossover strategies have been proposed, binomial and exponential are the most common choices for DE. Conversely, a wide range of mutations is available for implementing a DE algorithm. A specific implementation is usually identified by means of the DE/x/y/z notation. The first member, x, refers to the vector being mutated (namely the one to which difference vectors are added), y is the number of difference vectors employed, and z refers to the crossover strategy. For example, x could be *rand* (Formulas 19 and 21), *best* (Formulas 20 and 22) or even a combination of two vectors as in *current-to-best*, *current-to-rand* and *rand-to-best* (Formulas 23, 24, 25 and 26).

Different mutations correspond to different moves across D. The *current-to-best/1* strategy, for instance, can be of help in speeding up the convergence when dealing with non-critical fitness functions. However, it could be inadequate for highly multimodal functions, since privileging the direction toward the current best solution (a basic *rand/1* would be preferred in this case). The *current-to-rand/1* mutation (Equation 24 with $K = \mathcal{U}(0,1)$ and $F^{'} = K \cdot F$) is instead recommended for problems having a strong linkage among variables (e.g. rotated functions). It is worth stressing the fact that the latter scheme does not require crossover, since

```
procedure XOVERBIN(x_1, x_2)
    Index ← I(1, n)^a
    for i = 1 : n do
        if U(0, 1) ≤ CR || i == Index then^b
            x_off[i] ← x_1[i]
        else
            x_off[i] ← x_2[i]
        end if
    end for
    Output x_off
end procedure
```

[a] Random integer number uniformly distributed in $[0, n] \subset \mathbb{N}$
[b] Random real number uniformly distributed in $[0, 1] \subset \mathbb{R}$

FIGURE 14 Pseudocode of binomial crossover

already contains a built-in arithmetic crossover. This crossover strategy, unlike binomial and exponential, is rotational invariant. This aspect has been stressed and commended in (Takahama and Sakai, 2010), (Neri and Tirronen, 2010) and (Das and Suganthan, 2011). However, it can be questioned that having a rotational invariant operator is necessarily beneficial. Black-box problems are unknown and is not possible to know if the problem is a rotated version of another one. Moreover, the algorithm cannot benefit from knowing this information as the problem cannot be rotated back, and even if it could be brought back to an equivalent problem, general purpose metaheuristics have no guarantee of working well on the non rotated one. Real world problems are rarely similar to benchmark functions.

The selling point of DE is undoubtedly the simplicity of its algorithmic structures. Only two parameters, i.e. the scale factor F and the crossover rate CR, are required. Nonetheless, they need to be accurately tuned in order to avoid a deterioration of the performance. Originally, F was allowed in $[0, 2]$ and CR, being a probability, in $[0, 1]$. However, further experimentations in (Liu and Lampinen, 2005) have found $F \in [0.5, 1]$ and $CR \in [0.8, 1]$ to be the optimal ranges to use. The same study also suggested to tune the population size so that it is equal to ten times the dimensionality of the problem at hand. This result is arguable, as in LSOPs such high number of individuals would require an elevate computational budget to get convergence, i.e. optimization would be infeasible in the real world case. Moreover, a too large population size could negatively bias the search, as shown in PXI. On the other hand, a small population size leads to quick convergence. An observation that is not necessarily true for DE. Micro-populations (μ-DE) of about 5 individuals have shown to perform well, in particular in LSOPs (Parsopoulos, 2009). The population size is the third problematic parameter to tune. A general rule for choosing the population size does not exist yet. Moreover, unlike other EAs the DE framework seems to be particu-

```
procedure XOVEREXP(x_1, x_2)
    x_off ← x_2
    Index ← I(1, n)^a
    x_off[Index] ← x_2[Index]
    i ← Index + 1
    while U(0, 1) ≤ CR || i ≠ Index do^b
        x_off[i] ← x_1[i]
        i ← i + 1
        if i > n then
            i ← 1
        end if
    end while
    Output x_off
end procedure
```

[a] Random integer number uniformly distributed in $[0, n] \subset \mathbb{N}$

[b] Random real number uniformly distributed in $[0, 1] \subset \mathbb{R}$

FIGURE 15 Pseudocode of exponential crossover

larly prone to stagnation rather than premature convergence. This phenomenon takes place even in presence of high population diversity, and can occur in μ-DE as well. The algorithm simply fails at improving upon individuals and needs an external force to exit the impasse. As example, in PIV a LS is launched to overcome stagnation. Other methods alternate multiple mutations to have a higher diversity of the moves performed across D (Iacca et al., 2014, 2015). Others, see the CoDE algorithm in (Wang et al., 2011), use multiple mutations to generate multiple mutants, and then make use of the fittest one while the others are discarded. Strategies resizing the population size on the go have also been proposed to ease stagnation, see (Brest and Maučec, 2008). Finally, in order to remove the burden of tuning F and CR, several self-adapting DE variants were designed. The most significant approaches are described in the remainder of this Section.

In Self-adaptive Differential Evolution (SADE), parameters are continuously adjusted on the strength of successful and unsuccessful past attempts (Qin and Suganthan, 2005). Two mutation schemes, *DE/rand/1/bin* (Formula 19) and *DE/current-to-Best/2/bin* (Formula 24), are alternately activated according to a probability p. Initially, a value of p[15]$= 50\%$ assures a balanced execution of the two schemes. Subsequently, p is changed in order to privilege the most successful one according to:

$$p = \frac{ns_1 \cdot (ns_2 + nf_2)}{ns_2 \cdot (ns_1 + nf_1) + ns_1 \cdot (ns_2 + nf_2)} \tag{27}$$

where ns_1 and nf_1 are counters to register the number of successes and failures

[15] p is the probability of selecting the first mutation scheme, therefore there is a $1 - p$ probability of selecting the second mutation scheme. The selection between the two mutation is done by generating a random number, that is compared with p.

of mutation number 1. Similarly, ns_2 and nf_2 are increased to control mutation number 2. The probability p is updated after a learning period $LP = 50$. Thus, the four counters are reset every 50 generations. For each individual scale factor F and crossover rate CR are randomly generated every 5 generations. In particular, F is drawn from a normal distribution with fixed mean value (i.e. 0.5) and standard deviation (i.e. 0.3). As for CR, the standard deviation is kept equal to 0.1, while the mean value CR_m is changed every 25 generations (initialized to 0.5). During this period promising CR values, i.e the offspring solution is accepted in the population, are stored. The median value is extracted from that set to update CR_m. In conclusion, SADE features multiple mutation schemes and capabilities. A newer version employing 4 mutation straegies, i.e. *DE/rand/2* (Formula 21) and *DE/current-to-rand/1* (Formula 24) were added, was subsequently proposed in (Qin et al., 2009).

A similar idea was adopted in(Omran et al., 2005). This study also proposes a Self-adaptive DE (SDE) where the strategy parameters are drawn from a Gaussian distribution.

A simple approach to implement self-adaptive control parameters in DE was proposed by Brest et al. (2006). The jDE algorithm is a basic *DE/rand/1/bin* with randomized scale factor and crossover rate. Each individual in the population has its own F and CR, and can be seen as the 3-tuple $\langle \mathbf{x}, F, CR \rangle$. A fit offspring solution gets a place in the new population, making the strategy parameters that led to its production available for future iterations. Conversely, a poor offspring gets discarded together with its unsuccessful F and CR values.This idea was already used in EAs. In jDE, control parameters have a probability to be randomly re-sampled. Before applying the mutation operator, a uniformly distributed random number is thrown and compared with the threshold τ_1. If $\mathcal{U}(0,1) < \tau_1$ then $F = F_l + \mathcal{U}(0,1) \cdot F_u$. The same is done for the crossover rate: if $\mathcal{U}(0,1) < \tau_2$ then $CR = \mathcal{U}(0,1)$. According to the authors, $\tau_1 = \tau_2 = 0.1$ leads to good results. As for the lower and upper bounds of the scale factor, F_u was set up equal to 1, while F_l to 0.1. The last value was picked up to prevent F from being equal to zero. With $F = 0$ only the crossover would be performed. Despite its simplicity, this method has shown to be efficient.

The popular JADE algorithm, see (Zhang and Sanderson, 2009), introduced a novel mutation strategy called *DE/current-to-pbest/1*. It can be seen as a modification of Formula 23 (*current-to-best*), where the best individual $\mathbf{x_{best}}$ is replaced with $\mathbf{x}_{\mathbf{best}}^{\mathbf{p}\%}$. The latter vector is randomly selected from a set containing a percentage $p\%$ of the fitter individuals in the population. Let us consider a population $\mathbf{P}$ of size M, the mutation is defined as:

$$\mathbf{x_m} = \mathbf{x_j} + F_j \left(\mathbf{x}_{\mathbf{best}}^{\mathbf{p}\%} - \mathbf{x_j} \right) + F_j \left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}} \right) \tag{28}$$

with $\mathbf{x_{r_1}}$ randomly chosen from $\mathbf{P}$, and $\mathbf{x_{r_2}}$ from $\mathbf{P} \cup \mathbf{A}$ ($\mathbf{x_{r_1}} \neq \mathbf{x_{r_2}} \neq \mathbf{x_{r_j}}$). $\mathbf{A}$ is an optional archive that can either be empty $\mathbf{A} = \emptyset$ or filled up with M individuals. The archive is used to promote fitness diversity in the population. The idea is to have an empty set $\mathbf{A}$, that occasionally accepts those solutions failing the survivor selection procedure. Thus, at most M individuals can be inserted in a

single generation cycle. At the end of every generation, if $|\mathbf{A}| > M$, a number of $|\mathbf{A}| - M$ solutions are randomly deleted. Self-adaptation is also guaranteed. For each j^{th} individual in $\mathbf{P}$ a scale factor F_j is sampled from a Cauchy distribution, i.e. $F_j = \mathcal{C}(\mu_F, 0.1)$, while CR_j from a Gaussian distribution, i.e. $CR_j = \mathcal{N}(\mu_{CR}, 0.1)$. Unacceptable values, i.e. $F_j = 0$ and $CR_j \notin [0, 1]$, are discarded. Initial values to be used in the first generation are $\mu_F = \mu_{CR} = 0.5$. Subsequently, μ_F and μ_{CR} get updated after each generation by means of

$$\mu_F = (1-c) \cdot \mu_F + c \cdot \frac{\sum_{F \in \mathbf{S_F}} F^2}{\sum_{F \in \mathbf{S_F}} F}, \qquad \mu_{CR} = (1-c) \cdot \mu_{CR} + c \cdot \frac{\sum_{CR \in \mathbf{S_{CR}}} CR}{|\mathbf{S_{CR}}|} \tag{29}$$

where $c \in [0, 1]$, $\mathbf{S_F}$ and $\mathbf{S_{CR}}$ are two memory stacks. The latter have to be filled up with successful combinations of F_j and CR_j. Parameters are successful if led to the production of a new individual wining the survivor process.

The Ensemble of Parameters and Mutation Strategies Differential Evolution (EPSDE) proposed in (Mallipeddi et al., 2011), is instead based on a completely different working principle. The scale factor F and crossover rate CR are simply randomly selected from the sets $\{0.5, 0.9\}$ and $\{0.1, 0.5, 0.9\}$ respectively. These figures are carefully picked values as suggested in the literature. Similarly, mutations and crossover operators are also picked up from two separated pools. In (Mallipeddi et al., 2011) one between *DE/current-to-pbest/1* and *DE/current-to-rand/1* mutation is randomly selected and equipped with a crossover strategy. The crossover is also uniformly chosen from a pool containing the binomial and exponential options. This method allows multiple combinations of strategies and parameters during the optimization.

The Modified Differential Evolution with p-best crossover (MDE-pBX) proposed in (Islam et al., 2012) implements the scheme *DE/current-to-gr_best/pBX*. Similarly to JADE, the algorithm makes use of a novel mutation where the best individual of the population is replaced by the best individual of a subset of the population. Such point, i.e. $\mathbf{x_{q\_best}}$, is extracted from a set containing a percentage q of randomly sampled individuals. Mutation can be more or less exploitative according to this percentage. If high, there is a higher probability to include the best solution in the set, thus promoting exploitation. As usual, the mutation is a linear combination of points:

$$\mathbf{x_m} = \mathbf{x_j} + F_j \left(\mathbf{x_{q\_best}} - \mathbf{x_j} \right) + F_j \left(\mathbf{x_{r_1}} - \mathbf{x_{r_2}} \right) \tag{30}$$

where $\mathbf{x_{r_1}}$ and $\mathbf{x_{r_2}}$ are chosen at random and $\mathbf{x_j}$ is the j^{th} individual. As for crossover, the best p points in the population are selected to form a second set. A point is randomly picked up from the set to undergo binomial crossover with the mutant vector. Therefore, the current solution $\mathbf{x_j}$ is only indirectly involved in the generation of the offspring, as taking part in the mutation. Unlike q, which is fixed during the optimization (suggested value is 15%), p changes at run-time with the number of passed generations: $p = ceil \left[\frac{n}{2} \cdot \left(1 - \frac{g-1}{g_{max}} \right) \right]$ (g is the counter, g_{max} is the maximum amount of generations). Scale factor and crossover rate are drawn

from Cauchy and Gaussian distributions with variable mean value μ_F and location parameter μ_{CR}. In the wake of JADE, two memory stacks, i.e. $\mathbf{S_F}$ and $\mathbf{S_{CR}}$, are used to keep track of promising control parameters. Mean value and locate factor of the Gaussian and a Cauchy distributions are updated as follow:

$$\mu_F = W_F \cdot \mu_F + (1 - W_F) \cdot \sum_{F \in \mathbf{S_F}} \left(\frac{F^z}{|\mathbf{S_F}|} \right)^{\frac{1}{z}} \tag{31}$$

$$\mu_{CR} = W_{CR} \cdot \mu_{CR} + (1 - W_{CR}) \cdot \sum_{F \in \mathbf{S_{CR}}} \left(\frac{CR^z}{|\mathbf{S_{CR}}|} \right)^{\frac{1}{z}} \tag{32}$$

with $z = 1.5$, $W_f = 0.8 + 0.2 \cdot \mathcal{U}(0,1)$ and $W_f = 0.9 + 0.1 \cdot \mathcal{U}(0,1)$. These distributions are employed to automatically generate F_j and CR_j for each individual, at each generation. The use of power mean and Cauchy distribution for μ_F leads to large perturbations. These are preferable to minimize the risk of premature convergence. Gaussian distribution is preferable for CR_m, which is limited in a smaller interval, i.e. $[0, 1]$.

The last algorithm is an example of complex structure. Numerical results show that MDE-pBX can be used to address a vast range of benchmark functions. However, similar results can be obtained with simpler structure, e.g. see comparisons in (Caraffini et al., 2012, 2014). Usually simper solutions are to prefer as it is quite difficult to comprehend the rationale behind a too complex algorithm. Consequently, it is difficult to build knowledge and draw conclusions when using unnecessarily complex algorithms. Some operators could be redundant. To support this statement, one can consider (Iacca et al., 2015). The study proposes a simplified version of a precedent DE based algorithm, designed by removing parts rather than complicating the original framework. This process appears not to jeopardize the performance of the algorithm.

2.3.5 Compact algorithms

Compact algorithms implement the logic of popular population based algorithms without the need of storing the actual population. The lack of candidate solutions to process is compensated for the presence a probabilistic representation of the population, in the fashion of Estimation of Distribution Algorithms (EDA)s, see (Baluja, 1994) and (Pelikan et al., 2000). This approach attempts to reproduce the beneficial aspects of having a population, see (Prügel-Bennett, 2010), without the problem of tuning the population size. A first compact GA (cGA) was implemented for combinatorial domains in (Harik et al., 1999) and a second in (**?**). A real-valued compact GA (rcGA) was then designed, see (Mininno et al., 2008). The use of a distribution for sampling individuals when needed is suitable for real-valued domains. The design variables can be easily sampled and scaled within the search space. Therefore, other compact implementations followed for BFO, see (Iacca et al., 2012b), and PSO, see (Neri et al., 2013). A compact Differential Evolution (cDE) was also proposed in (Mininno et al., 2011) and further improved in PI of this thesis.

Compact differential evolution

In cDE the optimization process takes place into the normalized domain $[-1, 1]$. Solutions do not need to be saturated or corrected. However, a normalization procedure must proceed the optimization process, as well as the final solution has to be re-scaled within the original domain at the end. A $(2 \times n)$-matrix, namely perturbation vector $\mathbf{PV} = [\boldsymbol{\mu}, \boldsymbol{\sigma}]$, must be pre-allocated with $\boldsymbol{\mu} = \emptyset$ and $\boldsymbol{\sigma} = 10\, ones\,(n)$[16]. Such a high value for $\boldsymbol{\sigma}$ flattens the **PV** distribution within the range $[-1, 1]$, and makes it behave/look like a uniform distribution at the early stage of the optimization process. This is important to assure an unbiased initial sampling. Then, the perturbation vector will adapt to the landscape by shrinking (the standard deviation) and moving (the mean value) towards to promising basins of attraction. With reference to the pseudocode in Figure 16, it can be seen that the only individual to be stored in memory is the elite $\mathbf{x_e}$, i.e. current best solution found by the algorithm during the optimization. The three solutions required for performing the well-known DE/rand/1 mutation scheme (Formula 19) are instead drawn from the distribution obtained by using mean value and standard deviation from **PV** in the truncated[17] Gaussian PDF:

$$PDF\left(truncNorm\left(\mathbf{x}[i]\right)\right) = \frac{e^{-\frac{(\mathbf{x}-\boldsymbol{\mu}[i])^2}{2\boldsymbol{\sigma}[i]^2}} \cdot \sqrt{\frac{2}{\pi}}}{\boldsymbol{\sigma}\,[i] \cdot \left(\operatorname{erf}\left(\frac{\boldsymbol{\mu}[i]+1}{\sqrt{2}\boldsymbol{\sigma}[i]}\right) - \operatorname{erf}\left(\frac{\boldsymbol{\mu}[i]-1}{\sqrt{2}\boldsymbol{\sigma}[i]}\right)\right)} \tag{33}$$

where erf is the error function, see (Gautschi, 1972), and i iterates from the first to the last design variable. The corresponding CDF must be worked out in order to sample a point. This is done by means of the Chebyshev polynomials approximation described in (Cody, 1969). This procedure is the bottleneck of cDE, since computational expensive. However, it cannot be completely avoided as necessary to sample each design variable. In PI (Chapter 3.2), the number of required sampling was minimized to have a faster structure. When the candidate solutions are sampled, the mutation strategy is performed. The resulting mutant is then mated with $\mathbf{x_e}$ via exponential crossover. If the offspring is fitter than $\mathbf{x_e}$, it takes the name of "winner" while the elite solution is labelled as "looser" (and vice-versa). Winner and looser are used to update **PV**, as shown in Formula 34 and 35 in Figure 16. The symbol $\circ$ is used to represent the element-by-element product between two vectors.

16 $ones\,(n)$ is a vector of ones, with size n.

17 The tails are taken off and then bell resized in $[-1, 1]$ to guarantee a unitary area.

$t = 1$
$\boldsymbol{\mu}^t \leftarrow \emptyset$
$\boldsymbol{\sigma}^t \leftarrow \lambda ones\,(n)$ ▷ e.g. $\lambda = 10$
$\mathbf{x_{elite}} \sim \mathbf{PV}$
while *Budget available* **do**
$\left.\begin{matrix}\mathbf{x_r}\\ \mathbf{x_s}\\ \mathbf{x_t}\end{matrix}\right\} \sim \mathbf{PV^{(t)}}$
$\mathbf{x_m} \leftarrow \mathbf{x_{r_r}} + F\,(\mathbf{x_{r_s}} - \mathbf{x_{r_t}})$ ▷ DE/rand/1, Formula 19
$\mathbf{x_{offspring}} \leftarrow$XOVEREXP$(\mathbf{x_{elite}}, \mathbf{x_m})$ ▷ Figure 15
if $f\,(\mathbf{x_{offspring}}) \leq f\,(\mathbf{x_{elite}})$ **then**
$\mathbf{x_{loser}} \leftarrow \mathbf{x_{elite}}$
$\mathbf{x_{elite}} \leftarrow \mathbf{x_{offspring}}$
$\mathbf{x_{winner}} \leftarrow \mathbf{x_{offspring}}$
else
$\mathbf{x_{winner}} \leftarrow \mathbf{x_{elite}}$
$\mathbf{x_{loser}} \leftarrow \mathbf{x_{offspring}}$
end if

$$\boldsymbol{\mu}^{(t+1)} = \boldsymbol{\mu}^{(t)} + \frac{1}{N_p}\,(\mathbf{x_{winner}} - \mathbf{x_{loser}}) \tag{34}$$

$$\boldsymbol{\sigma}^{2\ (t+1)} = \boldsymbol{\sigma}^{2\ (t)} + \boldsymbol{\mu} \circ \boldsymbol{\mu}^{\ (t)} - \boldsymbol{\mu} \circ \boldsymbol{\mu}^{\ (t+1)} + \frac{1}{N_p}\,(\mathbf{x_{winner}} \circ \mathbf{x_{winner}} - \mathbf{x_{loser}} \circ \mathbf{x_{loser}}) \tag{35}$$

$t \leftarrow t + 1$ ▷ $\mathbf{PV}^{(t+1)}$ becomes the current **PV**
$\mathbf{PV^{(t)}} = [\boldsymbol{\mu}^{(t)}, \boldsymbol{\sigma}^{(t)}]$ ▷ Update next generation **PV**
end while
Output $\mathbf{x_{elite}}$

FIGURE 16 Pseudocode of compact differential evolution

3 TAILORED ALGORITHMS

FOR MEMORY SAVING AND REAL-TIME OPTIMIZATION

This chapter proposes algorithmic solutions for tackling real-world applications in the field of Robotics. In several cases, optimization problems in this discipline have to be carried out on board of modest microcontrollers and in real-time. Thus, the need of using simple algorithms based on efficient working principles arises. Optimizers with these features display a negligible computational overhead and memory footprint. Approaches like 3SOME, ISPO and nuSA, are suitable candidate and can have a major role in tackling these problems. Also cDE can be easily implemented in embedded systems, and thanks to its stochastic nature is suitable for handling noisy fitness functions. However, the sampling procedure requires some processing time that can be inappropriate when the solution has to be provided quickly. In this light, an improved (faster) version of cDE was proposed in PI. The mechanism employed to reduced the number of sampling is shortly described in Section 3.2. The improved algorithm, cDE-light, is tested on a robotic application in PII, as described in Section 3.3. In PIII, 5 memory-saving algorithms, i.e. 3SOME, cDE, cDE-light, ISPO and nuSA, were plugged into a basic microcontroller and compared in solving a control task on the go. This real-world application is described in Section 3.4. Finally, a light μDE empowered with an extra move along the axes, i.e. μ-DEA see PIV, is presented in Section 3.5.

3.1 Relation with this piece of research

The real-world applications presented in this chapter are plagued by memory and computational limitations. The cDE-light algorithm have been specifically designed to overcome those limitations. Implementation details, rather than theoretical ons, had to be taken into consideration before performing the design. As example, it was important to understand how much memory was left after loading the operating system, the control system, drivers, etc., into the device. Compatibility and synchronization of the optimizer with the other software compo-

nents had to be assured to guarantee the functioning of the whole system. Thus, the implementation point of view preceded the design. Moreover, an existing implementation of cDE was taken into consideration and modified to produce cDE-light. This approach moved *from implementation to design*, so addressing **RQ** I. A similar consideration can be done for μ-DEA, and the single solution algorithms involved in the studies presented in this chapter.

3.2 PI: compact differential evolution light

Objectives and goal

The aim of this piece of work is to improve upon cDE, in order to make it faster and suitable for real-time applications in embedded systems, without deteriorating the performance with respect to the original algorithm.

Methodologies

Two algorithmic solutions were adopted. First, the *DE/rand/1* scheme was replaced with an equivalent mutation. The proposed option is faster, as it does not need to sample three individuals to produce a mutant vector. Second, a light implementation for the exponential crossover was employed.

The *DE/rand/1* schemes is a simple linear combination of three points. However, to obtain a point the CDF has to be evaluated. This is a computational expensive and slow process. Fortunately, mathematics help us avoid an excessive use of the sampling procedure. In cDE the population is probabilistically represented. Therefore, solutions can be seen as stochastic variables, whose realization is sampled from **PV** (which is a fair approximation of a Gaussian distribution). The three realizations are then combined according to Formula 19. The produced mutant can also be seen as a stochastic variable with Gaussian distribution. This distribution can be obtained by exploiting the properties of stochastic Gaussian variables, and is equivalent to: $\mathcal{N}\left(\boldsymbol{\mu}, (1+2F^2)\boldsymbol{\sigma^2}\right)$, with F being the scale factor, while $\boldsymbol{\mu}$ and $\boldsymbol{\sigma^2}$ are directly taken from **PV**. This "trick" can be used to sample mutants without having to sample three solutions and implement Formula 19. A single point is sampled per iteration.

The mutant and the elite solutions mate via exponential crossover light. The original exponential crossover exchanges a sequence of l design variables from the elite solution into the mutant vector. A loop is sued to swap variables: while $\mathcal{U}(0,1) \leq Cr$ is true, a variable is exchanged. Therefore, the length l of the exchanged sequence is not known a-priori. However, in the light variant, l is estimated in function of CR value: $l \sim round\left(\frac{\log(rand(0,1))}{\log(CR)}\right)$ (see PI for explanations). Hence, the l design variables can be simply copied from a random position.

Results

The cDE-light algorithm has shown to perform as well as its predecessor when applied to benchmark problems, simulations of a path following application, and two real-world robotic applications (PII and PIII). The algorithmic computational overhead has been measured to be lower than the one of cDE. In the real-world case, cDE-light is sensibly faster than cDE.

3.3 PII: space robot base disturbance optimization

Objectives and goal

This study faces a complex space applications. A robotic arm is mounted on a spacecraft for maintenance purposes. The trajectory of its end effector has to be planned by taking into consideration the disturbance at the bases of the robotic arm. The disturbance has to be minimized to prevent the spacecraft from changing its motion. This would results in a waste of fuel to re-adjust the orbit. In this light, the aim of this work is to plan the trajectory causing the least disturbance. This is a control task that can be formulated as an optimization problem.

Methodologies

The algorithm proposed in PI showed to work well on noisy environments. It is also light and suitable for embedded systems. For these reasons, it was chosen to act as controller for this application. In order to perform the optimization the system has been studied thoroughly. Useful inputs have been found in the literature. In particular, the interpolation method proposed in (Ren et al., 2006) was considered for designing the trajectory, while the study in (Xu, 1993) for evaluating the dynamic coupling between the manipulator and the base. The latter is a complex task that requires a deep knowledge of Robotics. By using a similar approach to the one proposed in (Xu, 1993), the optimization problem was formulated as: to detect angular positions, angular velocities and angular accelerations of each joint, for each knot k, so that the corresponding trajectory is the one with the least base disturbance. The fitness function was defined as the integral over time of the norm of the acceleration vector on the base. The functions describing the position of each joint over time were modelled via 5^{th} order polynomial splines. Considering that the robot manipulator under study has 3 joints, that the trajectory was marked by $k = 2$ knots, and that each joint is fed with angular position, velocity, and acceleration, the problem requires $3 \times 2 \times 3 = 18$ design variables.

Results

In comparison with other memory-saving algorithms, i.e. ISPO, nuSA and cDE, cDE-light displayed the best results.

3.4 PIII: MC for path-following mobile robots

Objectives and goal

The goal of this piece of work is to equip an Automated Guided Vehicle (AGV), e.g. (Müller, 1983; Hollier, 1987), with self-tuning capabilities. The AVG, has to be able to "learn" to follow a path by tuning and continuously adjusting its controller. In order to achieve this result, a suitable optimization algorithm has to be embedded within the AGV to minimize the error in following a path. If the path gets changed, such system will be able to promptly adapt the control system to the new configuration, without the need of manually programming the controller from scratch.

Methodologies

In order to keep the budget, a Lego Mindstorms NXT robot was chosen to serve as AGV. Despite its simplicity, the robot is suitable for this task as its NXT brick contains a common 32 bit ARM7 CPU. This device is widely used in industrial applications. It provides 256 KB flash memory and 64 KB RAM. On the same board is also present an 8 bit AVR microcontroller, with 4 KB flash memory and 512 bytes RAM. In order to handle sensors and run the control system as in a proper AGV, the NXT brick was equipped with a custom open source multi-tasking Real-Time Operating System (RTOS), namely the nxtOSEK RTOS (Chikamasa, 2012). The rapid prototyping approach was used to integrate control system and the optimization algorithms within the robot. This allowed for a quick design by means of Simulink. The Simulink scheme was subsequently converted into C code via the Matlab Real Time Workshop (Embedded Coder) Toolbox. The generated C code was cross-compiled using a GNU ARM tool-chain, thus obtaining the executable binary for the ARM7 architecture. Once flashed into the microcontroller, the software was able to perform all the required tasks, i.e. from sensor calibration to optimization of the control system. Moreover, the system was continuously monitored by keeping alive a bluetooth communication during the entire optimization process. The operating system did not leave much memory for the remaining tasks. An effort was made to save as much memory as possible in designing the control system. The latter was measured to occupy 28.528 KB of memory (part of which was required by the BIOS, which was linked to the user application). Memory saving algorithms were key to achieve optimization. The cDE and cDE-light algorithms required only 15 KB. Even less memory was needed for 3SOME, ISPO and nuSA (about 10 KB each). Despite their simple structure, the selected algorithms successfully optimized the parameters of the PID regulator on the fly.

Results

The 3SOME algorithm showed to perform best for this specific application. Further simulations confirmed that 3SOME is actually generally suitable for tuning PID regulators. Popular metaheuristics as Hooke-Jeeves and Nelder-Mead were also outperformed, as well as ad-hoc procedures like the Ziegler-Nichols open-loop method (Ziegler and Nichols, 1942), and the areas method (Nishikawa et al., 1984). Unlike ISPO, that perturbs the same variables several times before moving onto the next one, 3SOME performs at most 2 steps on each design variable. Thus, 3SOME's structure is not efficient only on separable problems. Moreover, it is suitable for tasks where variables have to be tuned according to the value of the previous ones. This is the case of PID regulators. For tuning such regulators, the proportional term is usually adjusted first, while the derivative and integral terms accordingly.

3.5 PIV: μ-DE with extra moves along the axes

Objectives and goal

To design a light but still efficient general purpose algorithm. Ideally, the proposed optimizer has to be fast, versatile and able to address LSOPs. The idea was to provide one more simple approach that can be used in high dimensional problems. This algorithm can be an alternative to cDE-light, that performs well up to 30 design variables, but displays a deterioration in the performance at 1000 dimension values.

Methodologies

Unlike other EAs, DE seems not to be affected from premature convergence, even when a small population size is employed. Micro-populations, with a size not superior to 5 individuals, have been used in several studies (Das and Suganthan, 2011). Despite the small number of individuals, they can still suffer form stagnation. In this light, a μ-DE was considered in this study for its small memory footprint, but further enhanced in the fashion of MC. The S operator was integrated within the DE framework to add a complementary move, i.e. along the axes, in support of the diagonal move performed by means of the *DE/rand/1* scheme. S was also chosen as it was proven to be efficient on LSOPs (Caraffini et al., 2013a). Moreover, the local search performed by the S operator is beneficial in presence of stagnation. The proposed μ-DEA adopts a coordination logic similar to the LS chains method described in (Molina et al., 2010a). The DE scheme is iterated for a fixed number of generations. Subsequently, there is a probability $\eta = 20\%$ to apply S on the best individual. In this case, the initial value for the exploratory radius ρ is equal to 40% of the size of D. Such value assures an initial exploration.

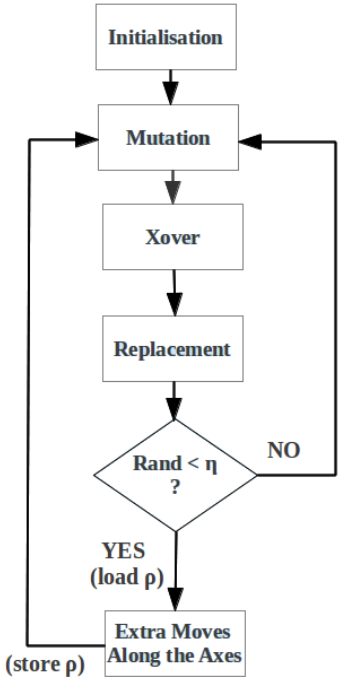


FIGURE 17 μ-DEA coordination logic.

After a fixed number of FEs, S is paused. According to the S logic, see Section 2.2.1, the radius decreases during the optimization to gradually converge and behave as a proper LS. The idea is not to restart S for the future iterations. The new value of ρ is saved for the next activation. S is unpaused, rather than restarted. In case of stagnation, the DE part is not able to improve upon individuals. Therefore, the S logic will work on the same solution as if it was not stopped. Basically, in presence of stagnation the S algorithm keeps on working on the same solution until stagnation is overcome. The activation logic for the S operator is depicted in Figure 17.

Results

This work proposes a DE-based MC algorithm with a modest memory footprint and a high performance. The μ-DEA algorithm improves upon classic DE, and equals state-of-the-art DE variants, over several test suites for real-valued optimization. In particular, similar numerical results to those of SADE, JADE and MDE-pBX are obtained at low dimension values. Superior results are instead obtained on LSOPs. The extra move along the axes has shown to be efficient and seems to help avoid stagnation.

4 ALGORITHMS WHICH ADAPT TO THE PROBLEM

This chapter contains 6 novel optimizers with self-adaptive capabilities. In Section 4.2 are included classic approaches. For instance, PV presents an innovative mechanisms for tuning the parameters of the proposed algorithm. In PVI and PVII, the super-fit approach (Caponio et al., 2009a) is proposed. Conversely, Section 4.3 includes hyper-heuristic algorithms. Three different algorithmic solution are presented. The first one in PVIII sees the activation of multiple LS routines, in the fashion of MAs. The algorithms in PIX and PX make use of more complicated coordination strategies, e.g. an adaptive logic from hyper-heuristic methods is employed in PX.

4.1 Relation with this piece of research

The novel optimizers presented in this chapter were designed to be robust and versatile, thus facing black-box problems. They contains mechanisms to implement adaptation to the problem at hand based on multiple working principles. Popular adaptive systems have been employed. New ones have been designed, to generate dynamic algorithmic structures able to self-tune their parameters, see e.g. PV. For this reasons, they have been grouped in this chapter to thoroughly address **RQ** II. Obviously, these optimizers display a different algorithmic structure to those in Chapter 3. In this case, a major attention was paid to actualize adaptivity. Hence, the proposed optimizers tend to be heavier, and more complex to increase their versatility. The optimization process generally takes place off-line, in personal computers and a higher number of FEs can be used with respect to real-time applications. Thus, the implementation aspect plays a secondary role with respect to other algorithmic considerations. In this regard, the opposite approach, i.e.*from design to the implementation*, is presented.

4.2 Empiric approach

This Section is meant to introduce a sophisticated adaptive DE algorithm, Section 4.2.1, and its super-fit variant. A third super-fit approach is also presented.

4.2.1 PV: multicriteria adaptive differential evolution

Objectives and goal

This work aims at designing a versatile DE algorithm. The algorithm is supposed to display several desirable features, such as adaptation to the problem. In order to achieve adaptation, an empiric mechanism to automatically tune parameters, i.e. F and CR, has to be designed. The same has to be done for efficiently handling the computational budget. Versatile DE implementations make use of several mutation schemes. A novel approach to efficiently distribute the budget amongst the perturbation strategies is needed.

Methodologies

Multicriteria Adaptive Differential Evolution (MADE), was designed around 4 classic perturbation schemes from DE: *DE/rand/1/bin*, *DE/rand/2/bin*, *DE/rand-to-best/2/bin* and *current-to-rand/1* (see Section 2.3.4). Control parameters F and CR are adjusted during the optimization process. Similarly to JADE, and other successful adaptive DE variants, F is sampled from a Cauchy distribution, while CR from a Gaussian distribution. Both the distributions are updated according to intermediary results, obtained during the execution of the algorithm. Details are given in PV.

Novel methods have been proposed to select the most appropriate perturbation scheme according to the specific problem. Three approaches have been compared in this study. They requires each perturbation scheme to be associated to a probability p_k, with $k = \{1, 2, 3, 4\}$. Initially, such probability is set up equal to $p_k = 0.25$ ($\forall k$), to be subsequently updated after every generation. Specifically, at a generic generation g the Normalized Relative Fitness Improvement (NRFI) index, and the Normalized Distance to the Best Individual (NDBI) index, are calculated as follows:

1. Normalized relative fitness improvement (NRFI):

$$\eta_i^g = \frac{\tilde{\eta}_i^g}{\max\{\tilde{\eta}_i^g | i = 1, 2, \ldots, NP\}} \tag{36}$$

where

$$\tilde{\eta}_i^g = \frac{f(\mathbf{x}_i^g) - f(\mathbf{x}_i^{g+1})}{f(\mathbf{x}_i^g) - f(\mathbf{x}_{best}^g)} \tag{37}$$

2. Normalized distance to the best individual (NDBI):

$$\tau_i^g = \frac{\tilde{\tau}_i^g}{\max\{\tilde{\tau}_i^g | i = 1, 2, \ldots, NP\}} \tag{38}$$

where

$$\tilde{\tau}_i^g = \sqrt{\sum_{j=1}^{D} \left(\mathbf{x}_{i,j}^g - \mathbf{x}_{best,j}^g\right)^2}. \tag{39}$$

The index η_i^g is used to have information abut the quality of the i^{th} individual. To some extent, it measures the potential of the individual in leading the search. Conversely, τ_i^g is used to maintain some degree of fitness diversity. The two indexes are combined to generate the "impact" γ_i^g (for each i^{th} individual at a generic generation g). The impact is used to work out the selection probability. Amongst the three method proposed in PV to work out γ_i^g, the most robust appears to be the *number-of-dominating-impacts* method. It consists of the following formula:

$$\gamma_i^g = |\{(\eta_j^g, \tau_j^g) | (\eta_i^g, \tau_i^g) \succeq (\eta_j^g, \tau_j^g), j \in h_i^g\}| \tag{40}$$

where $|\cdot|$ denotes the cardinality of a set; $\mathbf{a} \succeq \mathbf{b}$ means that vector $\mathbf{a} = (a_1, \ldots, a_k)$ dominates vector $\mathbf{b} = (b_1, \ldots, b_k)$, i.e. $\forall i \in \{1, \ldots, k\}, a_i \geq b_i \wedge \exists i \in \{1, \ldots, k\}, a_i > b_i$. Details on how to generate the set $h_i^g = \{\bigcup_{k=1}^{K} s_k^g\} \backslash s_{n_i}^g$ are in PV. The calculation of a good impact index is key to obtain p_k. On the basis of the impact of the operator, the credit of each perturbation strategy is calculated by

$$r_k^g = \frac{\sum_{n_i \in s_k^g} \gamma_i^g}{\left|s_k^g\right|} \tag{41}$$

where $k = 1, \ldots, K$ ($K = 4$ as four recombination strategies were used). Subsequently, the quality q_k^g of each perturbation scheme is updated by the following additive relaxation mechanism:

$$q_k^g = (1 - \beta) \cdot q_k^g + \beta \cdot r_k^g \tag{42}$$

where β ($\beta \in [0, 1]$) is the adaptation rate, and q_k^0 is initialized with 1. Finally, the probabilities p_k for the incoming generation are computed:

$$p_k^{g+1} = p_{min} + (1 - K \cdot p_{min}) \frac{q_k^g}{\sum_{k=1}^{K} q_k^g}. \tag{43}$$

Results

MADE was executed on 38 scalable problems at 30 and 50 dimensionality values, and used to solve a real-world problem, i.e. Lennard-Jones potential minimization at multiple dimension values. Numerical results suggested that, over the considered problems, MADE is superior to jDE, SDE, SADE and EPSDE, but also competitive with JADE and CoDE in terms quality of the solution and rate of convergence.

4.2.2 PVI: super-fit MADE

Objectives and goal

The main objective of this work is to improve upon the performance of the MADE algorithm, with a particular attention on solving the problems in the CEC 2013 test suite for real-valued optimization (Liang et al., 2013).

Methodologies

Several papers in the literature present "super-fit" schemes for improving upon the classic DE framework. As can be seen in (Caponio et al., 2009b; Iacca et al., 2011), the presence of a fit individual in the initial population is beneficial. Exploration is not jeopardized, and the risk of stagnation lowered.

In the attempt of tackling the problems in (Liang et al., 2013), it was clear that CMA-ES was able to solve monomodal problems by using only 30% of the allowed computational budget. Moreover, "mildly" multimodal problems were nearly solved with such limited amount of FEs, meaning that promising basin of attraction were spotted, but not adequately refined by CMA-ES.

In this light, a Super-fit MADE (SMADE) variant was designed by using a standard CMA-ES, as described in Section 2.3.1. A number of FEs equal to 30% of the total computational budget is used to run CMA-ES. The returned solution takes the place of the worse individual in the initial population of a MADE instance. The remaining 70% of the total budget is left to MADE to complete the process.

Results

SMADE showed to be extremely efficient over the whole test suite in (Liang et al., 2013), optimized at 10, 30 and 50 dimension values. Experiments were performed according to the specifications for the "Special Session & Competition on Real-Parameter Single objective Optimization of the IEEE Congress on Evolutionary Computation (CEC) 2013". As expected, unimodal or mildly multimodal problems, even when non-separable and ill-conditioned, were solved during the early stages of the optimization. Highly multimodal landscapes were efficiently tackled by the presence of MADE. SMADE has been ranked 7^{th} best algorithm in the CEC 2013 competition.

4.2.3 PVII: super-fit RIS

Objectives

Similarly to PV, the study in PVII presents another attempt to tackle the problems of the CEC 2013 benchmark for the special session in real-valued optimization.

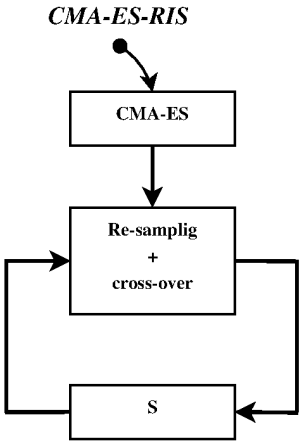


FIGURE 18 CMA-ES-RIS coordination logic.

Methodologies

As discussed in Section 4.2.2, a too promulgated use of CMA-ES over the problems listed in (Liang et al., 2013) is meaningful. Monomodal problems are optimized by using only a portion, i.e. 30%, of the allowed budget. While addressing multimodal problems, CMA-ES stops improving after that amount of FEs. On mildly multimodal landscapes mediocre results are returned, as the solution is not refined adequately. Thus, was decided to add a light and fast LS to refine it. As the risk of refining a local optimum is still present on highly multimodal functions, the LS should be re-started as many times as possible with the renaming 70% of the total budget. The RIS algorithm, see Section 2.3.1, was the best option to implement such task. The resulting algorithm, namely CMA-ES-RIS, can be seen as the execution of 3 memes, coordinated as in Figure 18.

Results

Numerical results showed that CMA-ES-RIS outperforms state-of-the-art algorithms, such as CCPSO2, MDE-pBX and CMA-ES, over the problems in (Liang et al., 2013) in 10, 30 and 50 dimension values. This optimizers was ranked 12^{th} across the 21 algorithms in the CEC 2013 competition on on real-parameter single objective optimization 2013.

4.3 Hyper-heuristic approach

Flexible algorithms can be obtained with the coordination of multiple heuristics. Hyper-heuristics make use a supervisory logic to execute single heuristics. Successful logics can be simple, e.g. random activation, or more elaborated. Three different examples are shown in the following sections, i.e. Section 4.3.1, 4.3.2 and 4.3.3.

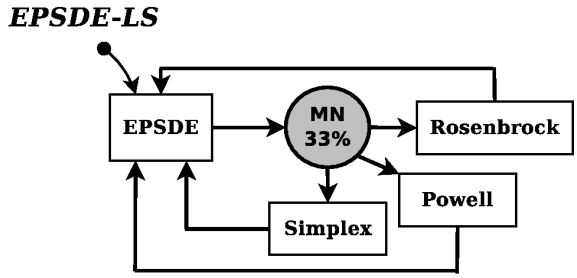


FIGURE 19 EPSDE-LS coordination logic.

4.3.1 PVIII: EPSDE with a pool of LS algorithms

Objectives

To make the EPSDE algorithm in (Mallipeddi et al., 2011) more efficient and robust against stagnation.

Methodologies

EPSDE is based on a simple idea, that is the ensemble of control parameters and mutations. In a nutshell, this approach can be described as the random selection of parameters from a pool containing the most successful values for F and CR. The same happens for mutations and crossover strategies, that are piked up from separate pools. Combinations of parameters and variation operators, that manage to generate fitter individuals, are used again in the future generations. This logic can be easily extended by placing novel operators within the pools, or by adding more pools, e.g. containing LS routines. Adding LS within a population based framework usually results into a boost in the performance of the algorithm, and in case of DE, also helps prevent stagnation. EPSDE-LS was designed based on this consideration. As shown in Figure 19, one amongst Nelder-Mead simplex method, Powell's conjugate direction method and Rosenbrock's algorithm (see Section 2.2.1), is activated with equal probability. The LS is applied to the best individual of the population. If the simplex algorithm is selected, the current best solution in the population takes the place of the first point of the initial randomly generated polytope. The diversity of moves performed by the three different LS methods, helps handle different landscapes. Although the last two methods share a similar working principle, they differ since Rosenbrock performs a single diagonal move, while Powell detects n conjugate directions where the fitness is optimized by means of a line-search method. The EPSDE-LS algorithm can be seen as a MA activating LS routines within a DE evolutionary cycle.

Results

EPSDE-LS showed to be able to improve upon its predecessor EPSDE, and also to outperform state-of-the-art algorithms such as MDE-pBX, CCPSO2 and CMA-ES on a significant number of benchmark problems.

4.3.2 PIX: Multi-strategy coevolving aging particle optimization

Objectives

To design a robust and versatile general purpose algorithm, able to address complex problems at different dimensionality values. The algorithms was supposed to be used to train a Feedforward Neural Network, modelling the forward kinematics of an 8-link an all-revolute robot manipulator. In order to try multiple configurations with different numbers of of layer and nodes, the algorithm is expected to perform well in terms of scalability, as the dimensionality of the problem can change.

Methodologies

In order to achieve robustness, complementary algorithm logics have been combined together in a solid optimizer, able to handle multiple scenarios. As the name suggests, the Multi-Strategy Coevolving Aging Particles (MS-CAP) can make use of different variation operators and perturbation strategies. It also makes use of an aging system to check the evolution of every single solution so that, if needed, it can be either replaced or re-sampled. This approach was created to escape from local basin of attractions or arrest stagnation.

MS-CAP was thought as a population-based algorithm alternating two unrelated logics to evolve the same population.

In the first stage, namely the Coevolving Aging Particles stage, each solution is seen as a particle that is perturbed independently. The perturbation acts along each dimension with a progressively decaying radius. Similarly to PSO, the particle is subject to a force pointing towards the best solution. Unlike PSO, such force is not constant but increasing, thus turning the search from being exploratory into exploitative. Additionally, when the perturbed particle improves upon its parent, a replacement occurs, and the particle's lifetime is set to zero. Otherwise, the lifetime is increased by one. The lifetime of each particle is used to work out a decay factor, i.e. $decay = e^{-lifetime}$. If this factor is too small, the solution get replaced by another randomly picked up solution from the swarm. Conversely, the solution is re-sampled. Subsequently, the exploratory radius is shrunk according to the value of the decay factor, see PIX for details. This mechanism is tangled but leads to good results. Its rationale is that each particle, while being attracted to the best solution, is also perturbed along each dimension with a radius decaying exponentially with the particle's "age". In other words, the particles in the swarm act as local searchers with an embedded restart mechanism

based on the particle decay.

In the second stage, the particles are mutated and recombined according to a Multi-Strategy approach, in the fashion of the ensemble of control parameters and mutation strategies in DE, see EPSDE (Mallipeddi et al., 2011). Hence, also the second component implements a complex but efficient framework. The latter is executed, for a fixed amount of FEs, when the first search logic fails at improving upon the best individual of the population.

The applicability of the MS-CAP algorithm was thoroughly tested and It was compared against SADE, JADE, jDE, MDE-pBX, EPSDE,CLPSO, CCPSO2, PMS and MACh (the two variant with CMA-ES and Solis-Wets were used according to the dimensionality of the problem). Experiments were executed by using 28 problems from CEC 2013 (Liang et al., 2013) at 30, 50 and 100 dimension values, but also 20 problems from CEC 2010 (Hansen et al., 2010) at 1000 dimension values. Moreover, 6 further optimization problems were considered. Three neural networks, with 8 input nodes, and 3, 4, and 5 hidden nodes respectively, were employed to approximate the forward kinematics of a robot. The three configurations were trained by using two different level of noise in the data set. Thus, three models with 27, 36, and 45 variables respectively, were optimized twice (with moderate and high noise level). Ad-hoc algorithms, i.e. Error Back Propagation (Rumelhart et al., 1986) and Resilient Propagation (Riedmiller and Braun, 1993), were also added on top of the aforementioned comparison algorithms.

Results

MS-CAP showed a very competitive performance, in training the neural networks, with respect to both general-purpose optimizers and ad-hoc algorithms. Numerical results, from benchmark functions, further confirmed that MS-CAP is capable of outperforming the comparison algorithms at multiple dimensionality values.

4.3.3 PX: Hyper-SPAM with Adaptive Operator Secletion

Objectives

The goal of this study was to improve upon the performance of the SPAM algorithm (Caraffini et al., 2014).

Methodologies

The SPAM algorithm was proven to be efficient but can still be improved in some directions. As example, its algorithmic structure is heavy, due to the use of the CMA-ES algorithm to work out the "separability index". On the other hand, this meme plays a major role as it also perform part of the optimization process. Moreover, CMA-ES seems to be the best method for estimating the Pearson correlation matrix $|\boldsymbol{\rho}|$ and subsequently work out the separability index ς. As

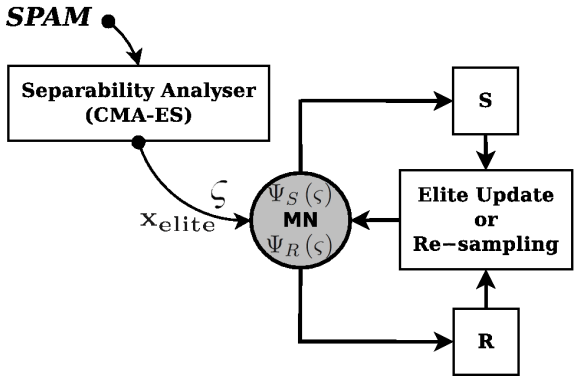


FIGURE 20 SPAM coordination logic.

can be seen from Figure 20 the probabilities $\psi_S(\varsigma)$, for the activation of S, and $\psi_R(\varsigma) = 1 - \psi_S(\varsigma)$, for the activation of Rosenbrock, depend on that value. In particular, the MN works out such probability at the beginning of the optimization process by means of a linear function ψ_S, see (Caraffini et al., 2014) for details. The ς index showed to be accurate in estimating the linkage between variables in most problems from popular test suites, e.g. (Hansen et al., 2010; Suganthan et al., 2005; Liang et al., 2013). However, in some problems the index is not truthful.

Thus, the idea of re-adjusting the probabilities during the optimization process appeared to be valid to improve upon the SPAM scheme. This was achieved by means of an adaptive model used in hyper-heuristics, i.e. the Adaptive Operator Selection (AOS) method. The CMA-ES part is still employed as beneficial in off-line cases. However, it could be removed for handling real-time applications, without preventing the adaptive model from working. The proposed SPAM-AOS variant requires two modules to function: a credit assignment module $\mathcal{C}$ and the actual adaptive model $\mathcal{M}$. The main role of the former is to estimate the quality of an operators after its application, based on feedback from the obtained fitness values (Fialho, 2010; Epitropakis et al., 2012). The latter, takes as input the estimated quality of each operator and returns, by means of the Probability Matching method in (Fialho, 2010; Thierens, 2005, 2007), the adjusted selection probabilities.

Results

SPAM-AOS improves upon the SPAM algorithm, and is competitive against modern state-of-the-art metaheuristics over 28 problems of CEC 2013 (Liang et al., 2013) at 10, 30 and 50 dimension values, as well as the 20 problems of BBOB 2010 (Hansen et al., 2010).

This optimizer can be seen as an attempt to think about algorithmic design from metaphor-free perspective, and in the specific case, to show how MC and hyper-heuristics are essentially very similar ideas (if not even the same idea).

5 A THEORETICAL STUDY ON POPULATION-BASED ALGORITHMS

The investigation performed PXI, concludes this thesis with a theoretical study on the dynamic behaviours of population-based algorithms for black-box optimization.

5.1 Relation with this piece of research

Despite the plethora of nature-inspired approaches that can be found in the literature, there are actually few significant strategies, and several variants. All the metaheuristics for general purpose optimization are expression of the same principle, and share similar issues. As example, one can think of premature convergence, stagnation etc. It is difficult to achieve versatility. Population based approaches claimed to be versatile and able to efficiently explore any possible fitness landscape. However, if computationally efficient perturbation strategies can be easily designed (for generating new solutions), and control mechanisms can be implemented to prevent algorithms from revisiting the same areas of D, it is obviously not entirely true that populations of candidate solutions can display the same degree of exploration for all the problems. In this investigation, the last point is addressed and formalized, in the form of a structural bias intrinsically nested in some population based algorithms. When it arises, such bias plagues the search as it limits exploration capabilities towards specific regions of D. By discussing the structural bias, Chapter 5 addresses **RQ** III.

5.2 PXI: structural bias in population-based algorithms

Objectives

When an algorithm is not capable of reaching all the parts of the search space with equal efficiency, favouring certain areas of the search space over others independently of the fitness function, it is exhibiting a structural bias. This study aims to provide theoretical and empirical analyses for displaying and understanding the structural bias in population-based algorithms for optimization, in order to be able to reduced it.

Methodologies

In this study two popular population based algorithms, i.e. GA and PSO, were taken under consideration. A particular test-function, presenting the same properties on each possible direction, had to be implemented to be able to study the bias. This function, named f_0, maps real numbers given in the interval $[0,1]^n$ into a uniformly distributed random real number within $[0,1]$. With such function, an algorithm like a typical GA is expected to keep the individuals disperse within $[0,1]$. Conversely, by displaying the best individuals over 50 runs in parallel coordinates, for both GA and PSO, was clear that, in particular when increasing the population size up to 100 individuals, a structural bias arose and manifested as non-uniform clustering of the population over time.

In order to formalize what graphically seen, a different approach was sued. A simplified GA was designed to state and prove a theorem on the variance of the population (See PXI for details). The implications of such theorem make the point that individuals cannot spread over the entire interval $[0,1]^n$ during the search, but necessarily converge in a smaller sub-portion of D (as confirmed by numerical results). Moreover, it was shown that such variance decreases as the population size increases, i.e. the higher the population size, the stronger the effect of the bias.

Finally, the Kolmogorov-Smirnov test was also performed on the distribution of the best individuals obtained over 50 runs on the problems of the CEC 2005 test suite (Suganthan et al., 2005). This statistic test further confirmed the presence of a stronger bias for problems with a complex superior to that of f_0.

Results

According to this study, theory predicts that structural bias in population-based optimizers is exacerbated with increasing population size, and increasing problem difficulty. These predictions, supported by the empirical analyses in PXI, revealed two previously unrecognised aspects of structural bias, that would seem vital for algorithm designers and practitioners. First, increasing the population size, though ostensibly promoting diversity, will magnify any inherent structural bias. Second, the effects of structural bias are more apparent when faced with

difficult problems.

In this paper was argued, from both theoretical and empirical standpoints, that structural bias is likely to be common in practice, and amplified when we would least expect it (when we increase the population size in hope of a more exploratory search) and when it may cause most damage (on difficult problems).

The proposed approach to revealing the structural bias can be easily replicated, and its use is recommended prior to finalising the parametric and design configurations of any optimization algorithm to be deployed on real-world problems.

6 CONCLUSIONS AND WAY FORWARD

This thesis is a collection of 11 papers covering multiple heterogeneous scenarios. The design of novel algorithms is shown from different perspectives and several winning strategies for tackling real-world applications are reported. A theoretical study was also performed to better understand and overcome limitations of population based algorithms. The amount of material produced for this thesis provides several alternatives for tackling heterogeneous optimization problems, in particular in the field of robotic engineering. In other words, the **GRQ** can be considered fully addressed.

Some algorithmic solutions presented in this thesis give inspiration for future developments.

For instance, the compact approach appeared to be promising and suitable for optimization in embedded systems. In particular, the proposed light variant of cDE is based on an elegant mathematical procedure for generating the mutant vector. Future studies can further enhance compact algorithms and make them more robust. For example, a potential study could be done by adopting alternative distributions to the Normal one, as there is no proof that Gaussian distributions are the best choice. The use of different distributions, or mixture of Gaussian distributions, could be beneficial in some cases. Obviously, it will complicate the sampling procedure, as well as the entire algorithmic structure. Thus, this idea requires a great deal of attention during the algorithmic design.

As for black-box optimization, the SPAM-AOS framework displays great adaptation capabilities. However, challenges for the future would see the replacement of the CMA-ES meme with a lighter operator. Possible alternatives to work out an index of operational separability could be based on the concept of variables linkage, see (Harik et al., 2006; Chen, 2012). Even lighter methods should be tried, based on simple empiric rules.

Finally, the study on the structural bias in population based algorithms must be completed. As done for GA and PSO in 5, also DE and other paradigms for real-valued optimization will have to be examined. The same procedure employed in 5 can be simply used to reveal the presence, and the strength, of a structural bias.

YHTEENVETO (FINNISH SUMMARY)

Algoritmien suunnittelusta ja implementoinnista laskennallista älykkyyttä hyödyntävässä optimoinnissa

Tietotekniikan huima kehitys viime vuosikymmenien aikana on tuonut useita tehokkaita ja monimutkaisia tekoälyä sisältäviä järjestelmiä teollisuuden ja tavallisen kuluttajan saataville. Tarjolla on laitteita, jossa on elektroniikka ja sen tarvitsema ohjelmisto sulautettuna, ja jotka käyttävät tietojärjestelmiä internetin kautta, käyttäjän sitä oikeastaan huomaamatta. Tällaiset laitteet suorittavat monimutkaisiakin tehtäviä, mutta esimerkiksi laitteen ”opettaminen” niiden suorittamiseen hyvin edellyttää usein vaikeiden optimointiongelmien ratkaisemista. Tässä työssä perehdytään tällaisten optimointitehtävien ratkaisemiseen laskennallista älykkyyttä hyväksikäyttämällä.

Ensimmäinen tutkimuskysymys liittyy optimointialgoritmien suunnitteluun tapauksessa jossa laskentakapasiteetti, muisti ja käytettävissä oleva laskenta-aika ovat etukäteen rajoitettu hyvin pieniksi. Tällainen tilanne tulee vastaan esimerkiksi kun halutaan sisällyttää laskennallista älykkyyttä hyödyntävä optimointialgoritmi reaaliaikaiseen sulautettuun järjestelmään. Työssä suunniteltiin muistia säästäviä optimointialgoritmeja, jotka voidaan implementoida mikrokontrolleriin. Lisäksi esiteltiin ja ratkaistiin kolme robotiikan sovellusta.

Toisessa tutkimuskysymyksessä lähestymistapa oli päinvastainen. Työssä kehitettiin kuusi uutta ”musta laatikko” -tyyppisten kustannusfunktioiden optimointiin sopivaa laskennallista älykkyyttä hyödyntävää optimointialgoritmia. Näiden suorituskykyä testattiin tunnettuihin benchmark-tehtäviin.

Työn kolmantena tutkimuskysymyksenä tarkasteltiin populaatiopohjaisiin optimointialgoritmeihin liittyvää ns. rakenteellista vinoutumaa kahden yleisesti käytetyn laskennallista älykkyyttä hyödyntävän algoritmin, geneettisten algoritmien (GA) ja parvioptimoinnin (PSO), tapauksessa. Rakenteellinen vinoutuma tarkoittaa sitä, että optimointialgoritmi tutkii sallittujen ratkaisujen joukkoa epätasaisesti, mutta tämä epätasaisuus ei ole peräisin optimoitavan kohdefunktion ominaisuuksista, vaan menetelmästä itsestään. Vinoutumaa analysoitiin teoreettisesti ja visualisoitiin myös graafisesti.

REFERENCES

Acampora, G., Cadenas, J. M., Loia, V. & Ballester, E. M. 2011a. A multi-agent memetic system for human-based knowledge selection. IEEE Transactions on Systems, Man, and Cybernetics, Part A 41 (5), 946-960.

Acampora, G., Gaeta, M. & Loia, V. 2011b. Hierarchical optimization of personalized experiences for e-learning systems through evolutionary models. Neural Computing and Applications 20 (5), 641-657.

Auger, A. & Hansen, N. 2005. A restart cma evolution strategy with increasing population size. In Proceedings of the IEEE Congress on Evolutionary Computation, 1769-1776.

Auger, A. 2009. Benchmarking the (1+ 1) evolution strategy with one-fifth success rule on the bbob-2009 function testbed. In Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers. ACM, 2447–2452.

Baggett, J. & Skahill, B. E. 2010. Hybrid optimization using evolutionary and approximate gradient search for expensive functions. In Proc. 2nd Int. Conf. Eng. Opt.

Baluja, S. 1994. Population-based incremental learning. Techical reports.

Beni, G. & Wang, J. 1989. Swarm intelligence in cellular robotic systems. In NATO Advanced Workshop on Robots and Biological Systems.

Beyer, H.-G. & Schwefel, H.-P. 2002. Evolution strategies–a comprehensive introduction. Natural computing 1 (1), 3–52.

Birattari, M., Yuan, Z., Balaprakash, P. & Stützle, T. 2010. F-race and iterated f-race: An overview. In T. Bartz-Beielstein, M. Chiarandini, L. Paquete & M. Preuss (Eds.) Experimental Methods for the Analysis of Optimization Algorithms. Springer.

Bremermann, H. J. 1962. Optimization through evolution and recombination. In M. C. Yovits, G. T. Jacobi & G. D. Golstine (Eds.) Proceedings of the Conference on Self-Organizing Systems – 1962. Washington, DC: Spartan Books, 93-106.

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.

Brownlee, J. 2011. Clever algorithms: nature-inspired programming recipes. Jason Brownlee.

Burke, E. K., Hyde, M., Kendall, G., Ochoa, G., Ozcan, E. & Woodward, J. 2010. Classification of hyper-heuristic approaches. In Handbook of Meta-Heuristics. Springer, 449–468.

Burke, E. K., Kendall, G. & Soubeiga, E. 2003a. A tabu search hyperheuristic for timetabling and rostering. Journal of Heuristics 9 (6), 451–470.

Burke, E., Kendall, G., Newall, J., Hart, E., Ross, P. & Schulenburg, S. 2003b. Hyper-heuristics: An emerging direction in modern search technology. International series in operations research and management science, 457–474.

Burke, E. K. & Bykov, Y. 2008. A late acceptance strategy in hill-climbing for exam timetabling problems. In PATAT '08 Proceedings of the 7th International Conference on the Practice and Theory of Automated Timetabling.

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

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

Caraffini, F., Iacca, G., Neri, F. & Mininno, E. 2012a. The importance of being structured: A comparative study on multi stage memetic approaches. In Computational Intelligence (UKCI), 2012 12th UK Workshop on, 1–8.

Caraffini, F., Iacca, G., Neri, F. & Mininno, E. 2012b. Three variants of three stage optimal memetic exploration for handling non-separable fitness landscapes. In Computational Intelligence (UKCI), 2012 12th UK Workshop on, 1–8.

Caraffini, F., Neri, F., Gongora, M. & Passow, B. 2013a. Re-sampling search: A seriously simple memetic approach with a high performance. In IEEE Symposium Series on Computational Intelligence, Workshop on Memetic Computing, 52–59.

Caraffini, F., Neri, F., Iacca, G. & Mol, A. 2013b. Parallel memetic structures. Information Sciences 227 (0), 60 - 82.

Caraffini, F., Neri, F., Iacca, G. & Mol, A. 2012. Parallel memetic structures. Information Sciences 227 (0), 60–82.

Caraffini, F., Neri, F., Passow, B. N. & Iacca, G. 2013. Re-sampled inheritance search: high performance despite the simplicity. Soft Computing 17 (12), 2235–2256.

Caraffini, F., Neri, F. & Picinali, L. 2014. An analysis on separability for memetic computing automatic design. Information Sciences 265 (0), 1 - 22.

Caraffini, F. 2014. Novel Memetic Computing Structures for Continuous Optimisation. De Montfort University. Ph. D. Thesis. ⟨URL:http://hdl.handle.net/2086/10629⟩.

Chen, Y.-P. 2012. Exploitation of Linkage Learning in Evolutionary Algorithms, Vol. 3. Springer.

Chikamasa, T. 2012. nxtOSEK/JSP - ANSI C/C++ with OSEK/μITRON RTOS for Lego Mindstorms NXT. Website http://lejos-osek.sourceforge.net.

Chiong, R., Ferrante, N. & Robert, I. M. 2012. Nature that breeds solutions. International Journal of Signs and Semiotic Systems 2 (2), 23–44.

Chiong, R. 2009. Nature-inspired algorithms for optimisation, Vol. 193. Springer Science & Business Media.

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.

Colorni, A., Dorigo, M., Maniezzo, V. & others 1991. Distributed optimization by ant colonies. In Proceedings of the first European conference on artificial life, Vol. 142. Paris, France, 134-142.

Cowling, P., Kendall, G. & Soubeiga, E. 2000. A hyperheuristic approach to scheduling: a sales summit. In Proceedings of the Third International Conference on Practice and Theory of Automated Timetabling, Vol. 2079. Springer. Lecture Notes in Computer Science, 176–190.

Das, S. & Suganthan, P. 2011. Differential evolution: A survey of the state-of-the-art. Evolutionary Computation, IEEE Transactions on 15 (1), 4–31.

Dawkins, R. 1976. The Selfish Gene. Press, Oxford University.

Dowsland, K. A., Soubeiga, E. & Burke, E. 2007. A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation. European Journal of Operational Research 179 (3), 759–774.

Eiben, A. E. & Smith, J. E. 2003. Introduction to Evolutionary Computation. Berlin: Springer-verlag.

Epitropakis, M. G., Tasoulis, D. K., Pavlidis, N. G., Plagianakos, V. P. & Vrahatis, M. N. 2012. Tracking differential evolution algorithms: An adaptive approach

through multinomial distribution tracking with exponential forgetting. In Maglogiannis, Plagianakos & Vlahavas (Eds.) Artificial Intelligence: Theories and Applications. Springer. LNCS 7297.

Eshelman, L. J. & Schaffer, J. D. 1992. Real-coded genetic algorithms and interval-schemata. In L. Whitley (Ed.) Foundations of Genetic Algorithms II. San Mateo CA: Morgan Kaufmann, 187–202.

Fang, X. S., Chow, C. K., Leung, K. W. & Lim, E. H. 2011. New single-/dual-mode design formulas of the rectangular dielectric resonator antenna using covariance matrix adaptation evolutionary strategy. IEEE Antennas and Wireless Propagation Letters 10, 734–737.

Fernandes, C., Tavares, R., Munteanu, C., Rosa, A. & Tavares, R. 2001. Using assortative mating in genetic algorithms for vector quantization problems. In in Proceedings of the 2001 ACM symposium on Cpplied Computing, 361–365.

Fernandes, C. & Rosa, A. 2001. A study on non-random mating and varying population size in genetic algorithms using a royal road function. In Evolutionary Computation, 2001. Proceedings of the 2001 Congress on, Vol. 1. IEEE, 60–66.

Fialho, A. 2010. Adaptive Operator Selection for Optimization. Université Paris-Sud XI. Ph. D. Thesis.

Fletcher, R. & Powell, M. J. 1963. A rapidly convergent descent method for minimization. The Computer Journal 6 (2), 163–168.

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

García-Martínez, C. & Lozano, M. 2008. Local search based on genetic algorithms. In Advances in metaheuristics for hard optimization. Springer, 199–221.

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.

Geoffry E. Hinton and, S. J. N. 1987. How learning can guide evolution. Complex systems 1 (3), 495–502.

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

Gwiazda, T. D. 2006. Crossover for single-objective numerical optimization problems, Vol. 1. Tomasz Gwiazda E-book.

Hamadi, Y., Jabbour, S. & Sais, J. 2012. Control-based clause sharing in parallel sat solving. In Y. Hamadi, E. Monfroy & F. Saubion (Eds.) Autonomous Search. Springer, 245–267.

Hansen, N., Auger, A., Finck, S., Ros, R. & others 2010. Real-Parameter Black-Box Optimization Benchmarking 2010: Noiseless Functions Definitions. INRIA.

Hansen, N., Müller, S. D. & Koumoutsakos, P. 2003. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (cma-es). Evolutionary Computation 11 (1), 1-18.

Hansen, N. & Ostermeier, A. 1996. Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation. In Proceedings of the IEEE International Conference on Evolutionary Computation, 312-317.

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

Hansen, N., Auger, A., Finck, S. & Ros, R. 2010. Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup. INRIA. Research Report.

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.

Holland, J. 1975a. Adaptation in Natural and Artificial Systems.

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

Hollier, R. 1987. Automated guided vehicle systems. IFS (Publications). International trends in manufacturing technology.

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

Hoos, H. H. 2012. Programming by optimization. Communication of the ACM 55 (22), 70–80.

Iacca, G., Caraffini, F., Neri, F. & Mininno, E. 2013. Single particle algorithms for continuous optimization. In Evolutionary Computation (CEC), 2013 IEEE Congress on, 1610–1617.

Iacca, G., Mallipeddi, R., Mininno, E., Neri, F. & Suganthan, P. N. 2011. Super-fit and population size reduction mechanisms in compact differential evolution. In Proceedings of IEEE Symposium on Memetic Computing, 21–28.

Iacca, G., Caraffini, F. & Neri, F. 2013. Memory-saving memetic computing for path-following mobile robots. Applied Soft Computing 13 (4), 2003–2016.

Iacca, G., Caraffini, F. & Neri, F. 2015. Continuous parameter pools in ensemble differential evolution. In Computational Intelligence, 2015 IEEE Symposium Series on. IEEE, 1529–1536.

Iacca, G., Neri, F., Caraffini, F. & Suganthan, P. N. 2014. A differential evolution framework with ensemble of parameters and strategies and pool of local search algorithms. In Applications of Evolutionary Computation. Springer Berlin Heidelberg, 615–626.

Iacca, G., Neri, F., Mininno, E., Ong, Y.-S. & Lim, M.-H. 2012a. Ockham's razor in memetic computing: three stage optimal memetic exploration. Information Sciences 188, 17–43.

Iacca, G., Neri, F. & Mininno, E. 2012b. Compact bacterial foraging optimization. In Proceedings of the 2012 International Conference on Swarm and Evolutionary Computation. Springer-Verlag, 84–92.

Iacca, G. 2013. Distributed optimization in wireless sensor networks: an island-model framework. Soft Computing 17 (12), 2257–2277. doi:10.1007/s00500-013-1091-x. ⟨URL:http://dx.doi.org/10.1007/s00500-013-1091-x⟩.

Iruthayarajan, M. W. & Baskar, S. 2010. Covariance matrix adaptation evolution strategy based design of centralized pid controller. Expert systems with Applications 37 (8), 5775–5781.

Islam, S., Das, S., Ghosh, S., Roy, S. & Suganthan, P. 2012. An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization. Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on 42 (2), 482–500.

Karaboga, D. & Basturk, B. 2007. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. Journal of Global Optimization 39 (3), 459–471.

Kendall, G., Soubeiga, E. & Cowling, P. 2002. Choice function and random hyperheuristics. In Proceedings of the fourth Asia-Pacific Conference on Simulated Evolution And Learning, SEAL. Springer, 667–671.

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

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

Koza, J. R. 1992. Genetic Programming: vol. 1, On the programming of computers by means of natural selection, Vol. 1. MIT press.

Krasnogor, N. & Smith, J. 2005. A tutorial for competent memetic algorithms: model, taxonomy, and design issues. IEEE Transactions on Evolutionary Computation 9, 474–488.

Li, X. & Yao, X. 2012. Cooperatively coevolving particle swarms for large scale optimization. Evolutionary Computation, IEEE Transactions on 16 (2), 210-224.

Li, X. 2010. Niching without niching parameters: particle swarm optimization using a ring topology. IEEE Transactions on Evolutionary Computation 14 (1), 150–169.

Liang, J. J., Qin, A. K., Suganthan, P. N. & Baskar, S. 2006. Comprehensive learning particle swarm optimizer for global optimization of multimodal functions. IEEE Transactions on Evolutionary Computation 10 (3), 281–295.

Liang, J. J., Qu, B. Y., Suganthan, P. N. & HernÃ¡ndez-DÃaz, A. G. 2013. Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization. Zhengzhou University and Nanyang Technological University.

Liu, J. & Lampinen, J. 2005. A fuzzy adaptive differential evolution algorithm. Soft Computing - A Fusion of Foundations, Methodologies and Applications 9 (6), 448–462.

Lopez-Ibanez, M., Dubois-Lacoste, J., Stützle, T. & Birattari, M. 2011. The irace package, iterated race for automatic algorithm configuration.

Loshchilov, I. 2013. Cma-es with restarts for solving cec 2013 benchmark problems. In Evolutionary Computation (CEC), 2013 IEEE Congress on, 369–376.

Malan, K. M. & Engelbrecht, A. P. 2013. A survey of techniques for characterising fitness landscapes and some possible ways forward. Information Sciences 241 (0), 148 -163.

Malan, K. M. & Engelbrecht, A. P. 2014. Fitness landscape analysis for metaheuristic performance prediction. In H. Richter & A. Engelbrecht (Eds.) Recent Advances in the Theory and Application of Fitness Landscapes, Vol. 6. Springer Berlin Heidelberg. Emergence, Complexity and Computation, 103-132.

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.

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

Meuth, R., M. Lim, H., Ong, Y. S. & Wunsch-II, D. C. 2009. A proposition on memes and meta-memes in computing for higher-order learning. Memetic Computing Journal 1 (2), 85–100.

Michalewicz, Z. & Fogel, D. B. 2004. How to solve it: modern heuristics. Springer Science & Business Media.

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., Lozano, M., García-Martínez, C. & Herrera, F. 2010a. Memetic algorithms for continuous optimisation based on local search chains. Evolutionary Computing 18 (1), 27–63.

Molina, D., Lozano, M. & Herrera, F. 2010b. Ma-sw-chains: Memetic algorithm based on local search chains for large scale continuous global optimization. In IEEE Congress on Evolutionary Computation, 1–8.

Moscato, P. 1989. On Evolution, Search, Optimization, Genetic Algorithms and Martial Arts: Towards Memetic Algorithms.

Mühlenbein, H. & Schlierkamp-Voosen, D. 1993. Predictive models for the breeder genetic algorithm in continuous parameter optimization. Evolutionary computation 1 (1), 25–49.

Müller, T. 1983. Automated guided vehicles. IFS (Publications).

Nannen, V., Smit, S. K. & Eiben, A. E. 2008. Costs and benefits of tuning parameters of evolutionary algorithms. In G. Rudolph, T. Jansen, S. Lucas, C. Poloni & N. Beume (Eds.) Parallel Problem Solving from Nature - PPSN X, Vol. 5199. Springer Berlin Heidelberg. Lecture Notes in Computer Science, 528–538.

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

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., Weber, M., Caraffini, F. & Poikolainen, I. 2012. Meta-lamarckian learning in three stage optimal memetic exploration. In Computational Intelligence (UKCI), 2012 12th UK Workshop on, 1–8.

Neri, F., Mininno, E. & Iacca, G. 2013. Compact particle swarm optimization. Information Sciences 239 (0), 96 - 121.

Nguyen, T. T. & Yao, X. 2008. An experimental study of hybridizing cultural algorithms and local search. International Journal of Neural Systems 18 (1), 1–17.

Nikolaus, H. & Stefan, K. 2004. Evaluating the cma evolution strategy on multimodal test functions. In Parallel Problem Solving from Nature-PPSN VIII. Springer, 282–291.

Nikolaus, H. 2005. The cma evolution strategy: A tutorial. Vu le 29.

Nishikawa, Y., Sannomiya, N., Ohta, T. & Tanaka, H. 1984. A method for auto-tuning of pid control parameters. Automatica 20 (3), 321–332.

Montes de Oca, M., Stutzle, T., Birattari, M. & Dorigo, M. 2009. Frankenstein's pso: A composite particle swarm optimization algorithm. IEEE Transactions on Evolutionary Computation 13 (5), 1120-1132.

Omran, M. G., Salman, A. & Engelbrecht, A. P. 2005. Self-adaptive differential evolution. In Computational intelligence and security. Springer, 192–199.

Ong, Y.-S., Lim, M.-H. & Chen, X. 2009. Research Frontier: Towards Memetic Computing.

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.

Parsopoulos, K. E. 2009. Cooperative micro-differential evolution for high-dimensional problems. In Proceedings of the conference on Genetic and evolutionary computation, 531–538.

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

Pelikan, M., Goldberg, D. & Lobo, F. 2000. A survey of optimization by building and using probabilistic models. In American Control Conference, 2000. Proceedings of the 2000, Vol. 5, 3289-3293.

Peng, F., Tang, K., Chen, G. & Yao, X. 2010. Population-based algorithm portfolios for numerical optimization. IEEE Transactions on Evolutionary Computation 14 (5), 782–800.

Pham, D. T. & Castellani, M. 2009. The bees algorithm: modelling foraging behaviour to solve continuous optimization problems. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 223 (12), 2919–2938.

Poikolainen, I., Caraffini, F., Neri, F. & Weber, M. 2012. Handling non-separability in three stage memetic exploration. In Proceedings of the Fifth International Conference on Bioinspired Optimization Methods and their Applications, 195–205.

Poikolainen, I., Iacca, G., Caraffini, F. & Neri, F. 2013. Focusing the search: a progressively shrinking memetic computing framework. International Journal of Innovative Computing and Applications 5 (3), 127–142.

Powell, M. J. D. 1964. An efficient method for finding the minimum of a function of several variables without calculating derivatives. The Computer Journal 7 (2), 155–162.

Press, W. H., Teukolsky, S. A., Vetterling, W. T. & Flannery, B. P. 2007. Numerical Recipes: the art of scientific computing Third Edition (C++), Vol. 994. Cambridge University Press.

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

Prügel-Bennett, A. 2010. Benefits of a population: Five mechanisms that advantage population-based algorithms. IEEE Transactions on Evolutionary Computation 14 (4), 500–517.

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

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

Rainer, S. & Kenneth, P. 1997. Differential evolution - a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11 (4), 341-359.

Ramírez-Chavez, L. E., Coello Coello, C. & Rodríguez-Tello, E. 2011. A gpu-based implementation of differential evolution for solving the gene regulatory network model inference problem. In Proceedings of the Fourth International Workshop on Parallel Architectures and Bioinspired Algorithms, WPABA, 21–30.

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

Ren, K., Fu, J. Z. & Chen, Z. C. 2006. A new linear interpolation method with lookahead for high speed machining. In Technology and Innovation Conference, 1056–1059.

Riedmiller, M. & Braun, H. 1993. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In IEEE International Conference on Neural Networks, 586–591.

Rinaldi, F. 2012. A class of derivative-free nonmonotone algorithms for unconstrained optimization. Seminario Dottorato 2012/13, 107.

Ros, R. & Hansen, N. 2008. A simple modification in cma-es achieving linear time and space complexity. In Proceesdings of the Parallel Problem Solving in Nature, 296-305.

Rosen, E. M. 1966. A review of quasi-newton methods in nonlinear equation solving and unconstrained optimization. In Proceedings of the 1966 21st National Conference. ACM. ACM '66, 37–41.

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

Rumelhart, D., Hintont, G. & Williams, R. 1986. Learning representations by back-propagating errors. Nature 323 (6088), 533–536.

Schwefel, H. 1981. Numerical Optimization of Computer Models. Chichester, England, UK: Wiley.

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

Smith, J. E. 2012. Estimating meme fitness in adaptive memetic algorithms for combinatorial problems. Evolutionary Computation 20 (2), 165–188.

Smith, J. E. 2007. Coevolving memetic algorithms: A review and progress report. IEEE Transactions on Systems, Man, and Cybernetics, Part B 37 (1), 6–17.

Snyman, J. 2005. Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms, Vol. 97. Springer Science & Business Media.

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

Solis, F. J. & Wets, R. J.-B. 1981. Minimization by random search techniques. Mathematics of Operations Research 6 (1), 19-30.

Spall, J. 1987. A stochastic approximation technique for generating maximum likelihood parameter estimates. In American Control Conference, 1987, 1161-1167.

Spall, J. C. 1992. Multivariate stochastic approximation using a simultaneous perturbation gradient approximation. IEEE Transactions on Automatic Control 372, 332-341.

Storn, R. 2005. Designing nonstandard filters with differential evolution. IEEE Signal Processing Magazine 22 (1), 103–106.

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

Storn, R. 1996. Differential evolution design of an IIR-filter. In Proceedings of IEEE International Conference on Evolutionary Computation, 268–273.

Storn, R. 1999. System design by constraint adaptation and differential evolution. IEEE Transactions on Evolutionary Computation 3 (1), 22–34.

Suganthan, P. N., 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.

Takahama, T. & Sakai, S. 2010. Solving nonlinear optimization problems by differential evolution with a rotation-invariant crossover operation using gram-schmidt process. In Nature and Biologically Inspired Computing (NaBIC), 2010 Second World Congress on, 526-533.

Tan, Y. & Zhou, Y. 2011. Parallel particle swarm optimization algorithm based on graphic processing units. In Handbook of Swarm Intelligence.

Thierens, D. 2005. An adaptive pursuit strategy for allocating operator probabilities. In Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation. New York, NY, USA: ACM. GECCO 2005, 1539–154.

Thierens, D. 2007. Adaptive strategies for operator allocation. In F. G. Lobo, C. F. Lima & Z. Michalewicz (Eds.) Parameter Setting in Evolutionary Algorithms. Springer Berlin Heidelberg. Studies in Computational Intelligence 54, 77–90. 00042.

Thomsen, R. 2004. Multimodal optimization using crowding-based differential evolution. In Evolutionary Computation, 2004. CEC2004. Congress on, Vol. 2. IEEE, 1382–1389.

Tseng, L.-Y. & Chen, C. 2008. Multiple trajectory search for large scale global optimization. In Evolutionary Computation, 2008. CEC 2008. (IEEE World Congress on Computational Intelligence). IEEE Congress on, 3052-3059.

Vrugt, J. A., Robinson, B. A. & Hyman, J. M. 2009. Self-adaptive multimethod search for global optimization in real-parameter spaces. IEEE Transactions on Evolutionary Computation 13 (2), 243–259.

Wang, Y., Cai, Z. & Zhang, Q. 2011. Differential evolution with composite trial vector generation strategies and control parameters. Evolutionary Computation, IEEE Transactions on 15 (1), 55–66.

Weise, T. & Chiong, R. 2010. Evolutionary approaches and their applications to distributed systems. In Intelligent Systems for Automated Learning and Adaptation: Emerging Trends and Applications, 114–149.

Whitley, D., Gordon, V. & Mathias, K. 1994. Lamarckian evolution, the baldwin effect and function optimization. In Y. Davidor, H.-P. Schwefel & R. MÃ€nner (Eds.) Parallel Problem Solving from Nature-PPSN II, Vol. 866. Springer Berlin Heidelberg. Lecture Notes in Computer Science, 5–15.

Whitley, D. 1994. A genetic algorithm tutorial. Statistics and computing 4 (2), 65–85.

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

Wu, Y., McCall, J. A. W., Corne, D. & Regnier-Coudert, O. 2012. Landscape analysis for hyperheuristic bayesian network structure learning on unseen problems. In IEEE Congress on Evolutionary Computation, 1-8.

Xinchao, Z. 2011. Simulated annealing algorithm with adaptive neighborhood. Applied Soft Compututing 11, 1827–1836.

Xu, C. & Zhang, J. 2001. A survey of quasi-newton equations and quasi-newton methods for optimization. Annals of Operations Research 103 (1-4), 213-234.

Xu, L., Hutter, F., Hoos, H. & Leyton-Brown, K. 2008. SATzilla: Portfolio-based algorithm selection for SAT. Journal of Artificial Intelligence Research 32, 565–606.

Xu, Y. 1993. The measure of dynamic coupling of space robot system. In Proceedings of the IEEE Conference on Robotics and Automation, 615–620.

Yu, E. & Suganthan, P. N. 2010. Ensemble of niching algorithms. Information Sciences 180 (15), 2815–2833.

Yuhui, S. & Eberhart, R. C. 1998. Parameter selection in particle swarm optimization. In V. Porto, N. Saravanan, D. Waagen & A. E. Eiben (Eds.) Evolutionary Programming VII, Vol. 1447. Springer Berlin Heidelberg. Lecture Notes in Computer Science, 591-600.

Zhang, J. & Sanderson, A. C. 2009. Jade: Adaptive differential evolution with optional external archive. IEEE Transactions on Evolutionary Computation 13 (5), 945–958.

Zhen, J., Jiarui, Z., Huilian, L. & Qing-Hua, W. 2010. A novel intelligent single particle optimizer. Chinese Journal of computers, vol33 3, 556–561.

Zheng, Y.-L., Ma, L.-H., Zhang, L.-Y. & Qian, J.-X. 2003. Empirical study of particle swarm optimizer with an increasing inertia weight. In Proc. IEEE Congr. Evol. Comput, 221–226.

Zhu, Z., Jia, S. & Ji, Z. 2010. Towards a memetic feature selection paradigm. IEEE Computational Intelligence Magazine 5 (2), 41–53.

Ziegler, J. G. & Nichols, N. B. 1942. Optimum settings for automatic controllers. Transactions of ASME 64, 759–768.

Özcan, E., Bilgin, B. & Korkmaz, E. E. 2008. A comprehensive analysis of hyper-heuristics. Intelligent Data Analysis 12 (1), 3–23.

ORIGINAL PAPERS

PI

COMPACT DIFFERENTIAL EVOLUTION LIGHT: HIGH PERFORMANCE DESPITE LIMITED MEMORY REQUIREMENT AND MODEST COMPUTATIONAL OVERHEAD

by

Giovanni Iacca, Fabio Caraffini and Ferrante Neri 2012

Journal of Computer Science and Technology, volume 27, number 5, pages 1056-1076

PII

ROBOT BASE DISTURBANCE OPTIMIZATION WITH COMPACT DIFFERENTIAL EVOLUTION LIGHT

by

Giovanni Iacca, Fabio Caraffini, Ferrante Neri and Ernesto Mininno 2012

Applications of Evolutionary Computation (EvoApplications), Lecture Notes in Computer Science Volume 7248, pages 285-294

PIII

MEMORY-SAVING MEMETIC COMPUTING FOR PATH-FOLLOWING MOBILE ROBOTS

by

Giovanni Iacca, Fabio Caraffini and Ferrante Neri 2013

Applied Soft Computing, volume 13, number 4, pages 2003-2016

Applied Soft Computing 13 (2013) 2003–2016

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Memory-saving memetic computing for path-following mobile robots

Giovanni Iacca[a], Fabio Caraffini[b,c], Ferrante Neri[b,c,*]

[a] INCAS[3] – Innovation Centre for Advanced Sensors and Sensor Systems, P.O. Box 797, 9400 AT Assen, The Netherlands
[b] Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, The Gateway, Leicester LE1 9BH, England, United Kingdom
[c] University of Jyväskylä, Department of Mathematical Information Technology, P.O. Box 35 (Agora), 40014 Jyväskylä, Finland

ARTICLE INFO

Article history:
Received 24 January 2012
Received in revised form
23 September 2012
Accepted 19 November 2012
Available online 11 December 2012

Keywords:
Computational intelligence optimization
Memetic computing
Real-time optimization
Mobile robotics
Path-following

ABSTRACT

In this paper, a recently proposed single-solution memetic computing optimization method, namely three stage optimization memetic exploration (3SOME), is used to implement a self-tuning PID controller on board of a mobile robot. More specifically, the optimal PID parameters minimizing a measure of the following error on a path-following operation are found, in real-time, during the execution of the control loop. The proposed approach separates the control and the optimization tasks, and uses simple operating system primitives to share data. The system is able to react to modifications of the trajectory, thus endowing the robot with intelligent learning and self-configuration capabilities. A popular commercial robotic tool, i.e. the Lego Mindstorms robot, has been used for testing and implementing this system. Tests have been performed both in simulations and in a real Lego robot. Experimental results show that, compared to other online optimization techniques and to empiric PID tuning procedures, 3SOME guarantees a robust and efficient control behaviour, thus representing a valid alternative for self-tuning control systems.

1. Introduction

Some real-world problems, due to real-time, space, and cost requirements, often impose the solution of an optimization problem within limited computational resources. This situation is typical in mobile robots where the specific application might require that all the computation is embedded within the robot hardware (a computer is not involved in the optimization process). In addition, there are some engineering applications that require the solution of complex optimization problems despite a limited hardware. An example of this class of problems is the space shuttle control. Despite the constant growth of the power in computational devices, space applications are an interesting exception. In order to reduce fault risks, very simple hardware is often used on purpose on space shuttles. This choice allows a high reliability of the computational cores. For example, since over 20 years, National Aeronautics and Space Administration (NASA) employs, within the space shuttles, IBM AP-101S computers, see [1]. These computers constitute an embedded system for performing the control operations. The memory of computational devices is of only 1 Mb, i.e. much less capacious than any modern device. It must be remarked that the computational devices on board of a space shuttle should reliably work without any reboot for months or even for years. Thus, the necessity of having an efficient control notwithstanding the hardware limitations (both memory and computational power) arises.

In these cases, the optimization algorithms should perform the task without a high employment of memory and computational resources. Unfortunately, high performance algorithms are usually fairly complex structures employing a population of candidate solutions and other computationally expensive components such as learning systems or classifiers, see e.g. [2].

In the present paper, a mobile robot path-following application is presented, in which the following controller, namely a proportional-integrative-derivative (PID), is tuned on-line by means of an optimization algorithm. Path-following robots, also called automated guided vehicles (AGVs) [3,4], are mobile robots whose main task is following a generic path, denoted e.g. by means of markers or wires on a surface. Since they increase efficiency and reduce costs, path-following robots are nowadays widely used in industry (see Fig. 1) for moving raw materials, transporting pallets and finished goods, removing scrap, etc. They are becoming increasingly popular also in the health-care industry, e.g. for efficient transportation of linens, trash and medical waste, patient meals, soiled food trays, and surgical case carts, and in many other application domains.

In this study, a test path-following application is implemented on a popular commercial hardware, namely the embedded microcontroller of Lego Mindstorms NXT, see Section 2. The robot

* Corresponding author at: University of Jyväskylä, Department of Mathematical Information Technology, P.O. Box 35 (Agora), 40014 Jyväskylä, Finland.
Tel.: +358 14 260 1211; fax: +358 14 260 1021.
E-mail addresses: giovanniiacca@incas3.eu (G. Iacca), fcaraffini@dmu.ac.uk, fabio.caraffini@jyu.fi (F. Caraffini), fneri@dmu.ac.uk, ferrante.neri@jyu.fi (F. Neri).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2012.11.039

Fig. 1. An industrial path-following mobile robot.

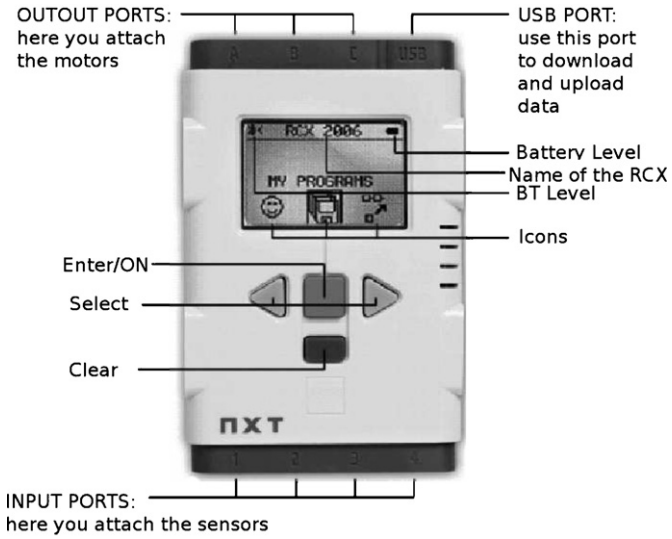


Fig. 2. The NXT brick.

configuration is a two driving-wheels rover, following a black elliptic path over a white background. Two sensors have been used, an ultrasonic sensor for registering the beginning and the end of each iteration of the closed trajectory, and a light sensor to follow the black path. More specifically, the light sensor measures the "amount" of black and white, i.e. the percentage of light, thus allowing a raw measure of the following error. In other words, the path-following task is translated into the requirement that, at each step of the control loop, the light sensor must measure 50% of white and 50% of black during the elliptic path. When the light/darkness proportion changes, an error is measured. The maximum error, obviously, occurs when 100% of white and 0% of black (or dually 100% of black and 0% white) is measured. A global measure of the error along the path is then computed as the integral absolute error (IAE). The optimization problem consists of finding those parameters proportional-integrative-derivative (PID) which allow a minimal IAE: in this way, the robot capable of learning the best set of parameters to follow a generic path. In addition, at every step of the control loop, if the IAE error goes beyond a fixed threshold, a bang-bang control is provided in order to move the robot to the correct place, and another PID parameter set is quickly computed by the optimization algorithm.

It is important to remark that in this work we propose an architecture in which both the control scheme and the optimization algorithm are implemented on board of the Lego Mindstorms, despite its severely limited computational and memory resources, thus avoiding any external computing device. In addition, it should be noted that, due to hardware limitations, the implementation of classical population based algorithms would not be a viable option on an embedded system of this kind. In other words, in this paper we show that the application of an advanced optimization algorithm can allow the accomplishment of a complex industrial task despite the employment of extremely limited hardware conditions.

Although some recent studies proved that population-based algorithms usually have a better performance than algorithms processing a single candidate solution, see [5], still there are some population-less (not only single-solution) methods which are able to provide relatively good results despite a limited memory footprint. If properly designed, population-less algorithms can be competitive with population-based algorithms in specific applications (in accordance with the no free lunch theorem, see [6]) and thus can be a satisfactory alternative when the hardware limitations forbid the use of a complex algorithms. We will refer in these papers to these methods as "memory-saving" algorithms, i.e. algorithms which do not make use of a population of solutions or the support of memory structures. Since this article considers a specific application characterized by severe hardware limitations we will focus on memory-saving algorithms.

The remainder of this paper is structured as follows. Section 2 describes the experimental hardware and software setup. Section 3 introduces the optimization problem and the proposed approach for solving it. More specifically, the real-time tasks which compose the control architecture are described in detail. Section 4 briefly explains the working principles of the optimization algorithm used, namely 3SOME. Section 5 presents the experimental results: a first subsection describes preliminary simulation experiments in which 3SOME was compared with two classical local search algorithms, namely the Hooke–Jeeves and the Nelder–Mead methods, and four different state-of-the-art memory-saving global optimization algorithms. A validation of the simulation results in the real-world case is then presented in the second subsection. Finally, in Section 6 the conclusion of this work is given. For the interested reader, Appendix A presents an additional experimental setup in which two empiric tuning procedures were compared with the 3SOME-based proposed online optimization approach.

2. The mobile robot path follower: hardware and software setup

2.1. Hardware configuration

Lego Mindstorms [7] is a line of the Lego products which includes a programmable NXT brick, electric motors, sensors, and other useful pieces such as gears, axles and pneumatics to build robots or other automated systems. One of the most important features of the Lego Mindstorms is the wide selection of pieces one can choose for designing the robot, which allows many different kinds of robotics applications, such as pick-and-place, fixed and mobile robotics. Originally designed to be used like a toy, the Lego Mindstorms has quickly become an important academic and educational instrument.

What makes the Lego Mindstorms an interesting tool is the NXT brick, see Fig. 2, which includes four input ports for connecting sensors, three output ports for connecting motors, Bluetooth wireless communication, and a reasonably powerful micro-controller, namely a 32 bit ARM7 micro-controller, with 256 KB flash memory

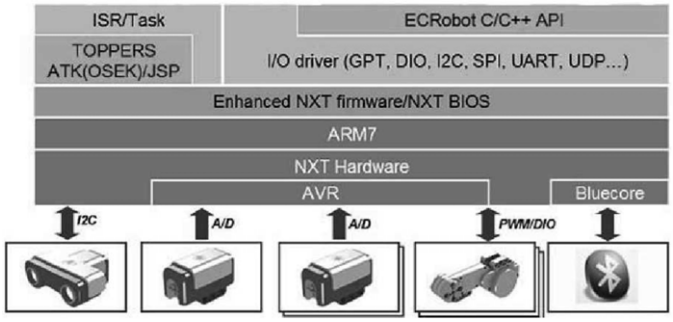


Fig. 3. The NXT brick: hardware and software architecture.

and 64 KB RAM, a 8 bit AVR micro-controller, with 4 KB flash memory and 512 bytes RAM. It is interesting to note that, even though the NXT brick is basically a toy, its processing power makes it even more powerful than the computer used on the Apollo 11 Moon mission.

2.2. Software configuration

The NXT brick can be programmed by means of the Lego visual programming environment called NXT-G, but a variety of unofficial languages and interface libraries publicly available exists, such as NXC, NBC, leJOS NXJ, and RobotC. Although NXT-G provides a very intuitive and simple graphical interface to develop simple tasks, high-level programming languages are way more powerful, as they allow to develop complex applications, like a control system or an optimization algorithm.

In this work, the NXT brick was equipped with a custom open source multi-tasking real-time operating system (RTOS), namely the nxtOSEK (or simply OSEK) RTOS [8]. To implement the control system and the optimization algorithms and execute it on the OSEK RTOS on board of the NXT brick, a rapid prototyping approach was adopted: the application was designed and implemented in Simulink, and then the C code was automatically generated using the Matlab Real Time Workshop (Embedded Coder) Toolbox. The generated C code was cross-compiled using a GNU ARM tool-chain, thus obtaining the executable binary for the ARM7 architecture flashed to the micro-controller via a USB cable.

Fig. 3 is an outline of the NXT architecture described so far. The lowest layer shows the internal bus system of the NXT brick. For each sensor connected to the brick, a proper bus is used to handle the communication. The ultrasonic sensor and the Bluetooth TX/RX are handled separately: the first device needs a continuous data exchange with the main core by means of I2C bus to keep the communication alive, while the Bluetooth protocol must be executed by a specific Bluecore circuit. The other sensors are instead handled by the AVR co-processor: since the servo motor revolution sensor provides a digital signal the co-processor can read its data without any processing, while the signals from the others sensors must be acquired by the analog to digital (A/D) converter. The middle layer in Fig. 3 represents the firmware. It contains the basic input output service (BIOS) that must be flashed with the program into the micro-controller to handle the peripheral devices. On the topmost layer the interrupt service routine (ISR) is shown. The main ARM7 processor accesses sensors (to read sensor A/D value) and servo motors (to set the PWM duty ratio and brake mode) independently through periodical ISRs. Motor revolutions are directly captured by pulse-triggered ISRs. The ultrasonic sensor has its own ISR to directly communicate interrupt to the main ARM7 processor via the I2C communication channel.

The OSEK RTOS comes with a complete C/C++Application Programming Interface (API), called Embedded Coder Robot (ECRobot), which allows developers to access interrupts and data of every sensor and actuator. In addition to that, ECRobot includes a custom Matlab Simulink toolbox named ECRobot NXT, through which it is possible to implement programs for Lego Mindstorms robots as Simulink schemes, and generate the C/C++code for the NXT architecture. It contains several blocks to access all the features of the Lego NXT sensors, actuators and communications services (USB/Bluetooth), thus providing powerful modelling capabilities for NXT control strategies and plant dynamics. ECRobot NXT also includes a Real-Time Workshop Embedded Coder target to generate binaries for the OSEK RTOS and a 3-D virtual reality graphical environment.

3. Self-tuning PID controller for a path-following robot

This section describes the design of the control and optimization scheme for the path-follower robot, based on the HW/SW platform described in the previous section. The robot under consideration was equipped with two servo-drives (i.e. two driving wheels), a light sensor and an ultrasonic sensor. The light sensor was used to "see" the path to follow. The robotic system was firstly simulated in Simulink (using the ECRobot NXT toolbox) and then tested in a real-world configuration. In the latter case, in addition to the light sensor the Bluetooth channel was used to send data from the robot to a logger PC, and the ultrasonic sensor was used to recognize the robot initial position, in order to detect when the closed path is completed and a new set of PID parameters must be computed.

Both in the simulation and in the real-world implementation, the optimization process is performed as follows. Whenever the path is completed, the optimization algorithm on board the robot attempts to adjust the parameters of a PID controller, which in turn affect the following capability, to make the movement as smooth as possible. After a predetermined number of evaluations, the robot uses the "best" PID parameters found so far, that is those ones minimizing the IAE cost-function. It is important to remark that, in order to get different robot behaviours, other objective functions could be used without modifying the general scheme of the application. In this case, we minimize the IAE to obtain a smooth trajectory. If a fast motion was also required, the objective function could be modified, e.g. adding a time-dependent term.

3.1. System overview

With respect to Fig. 4, the entire control and optimization system is wrapped within the *SYSTEM* block, supervised by the *ExpFcnCall* scheduler. The *SYSTEM* block contains four tasks in which the application has been decomposed, namely (1) initialization, (2) control system, (3) optimization and (4) data logging, see Fig. 5. The scheduler allows them to work at different operating frequencies and cooperate. The tasks can communicate with each other by writing data to shared memories, allocated into a selected region of the micro-controller memory stack.

The system has two inputs, namely the measures read by the *NXT Light Sensor Interface* and *Ultrasonic Sensor Interface* blocks of the ECRobot NXT toolbox, and two outputs, i.e. the actuation signals on the two motors (see the two *Sensor Motor Interface* blocks on the right side of Fig. 4).

3.2. Task 1: initialization

The first task is a simple initialization task. Its priority is the highest, so it is executed before all the others tasks and only once. This task performs all the procedures necessary to enable motors, encoders, and reset memories of all the integrators used. In this way it assures the correct start of the system. It also provides the initial solution for the optimization algorithm, in order to initialize the

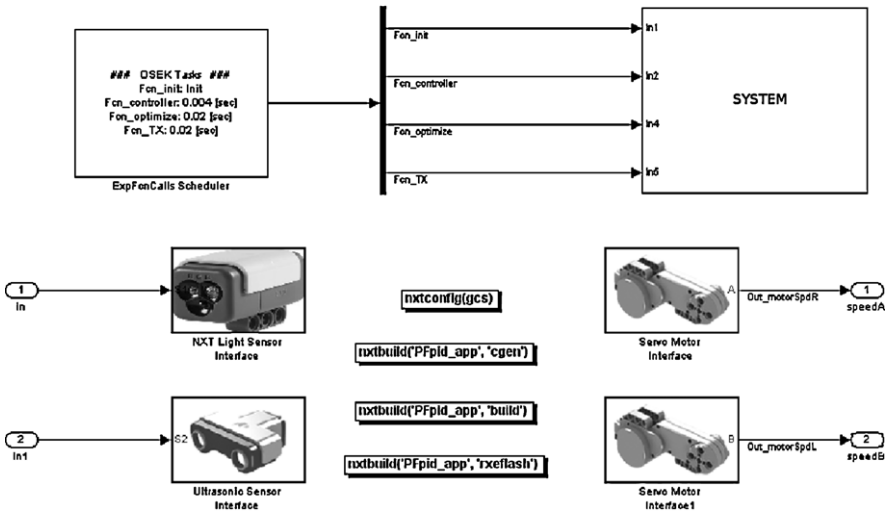


Fig. 4. System overview: scheduler, tasks and sensors interfaces.

optimization process and keep it ready to work. The first solution is not completely random. The derivative and integrative terms, in fact, are set to zero while the proportional term assumes a random value within the given search space (see later). Thus, at the beginning a simple random proportional regulator is used, while as the optimization goes on the various terms are adjusted. At the end of the initialization, a flag called *startFlag* is set to zero. This indicates that the system is initialized but not yet ready, since the light sensor must still be calibrated before starting the optimization process. This flag is saved into a memory shared with the other tasks, so that every task is able to read it and wait it to become non-zero.

3.3. Task 2: control system

This task is responsible for the motion of the robot. It is the fastest and it is activated every 1 ms. At each execution, a new value from the light sensor is acquired. Comparing the reading with a predetermined value (set-point), a controller is be able to know if the robot is in the correct position or not, and updates the motor speed accordingly. However, these operations are executed only if the *startFlag* is set as true. Whenever the control system task is activated, the *startFlag* is checked. If the read value is equal to zero, the control scheme is disabled and the calibration of the light sensor starts. The calibration is performed only once, so that when it is completed the *startFlag* value is permanently set to one.

3.3.1. Light sensor calibration

One of the problem that arises when light is used as an input source, is that it changes with time, and from place to place. In other words, different environments have different light conditions. The idea behind calibration is to adjust the sensor to the conditions expected in the room. Depending on the room, the sensor can be calibrated only once, even if the robot must be used multiple times within the same day. But if the environment light changes, e.g. because there is a large window that lets the natural sunlight come into the room, the sensor readings could be corrupted. In an ideal

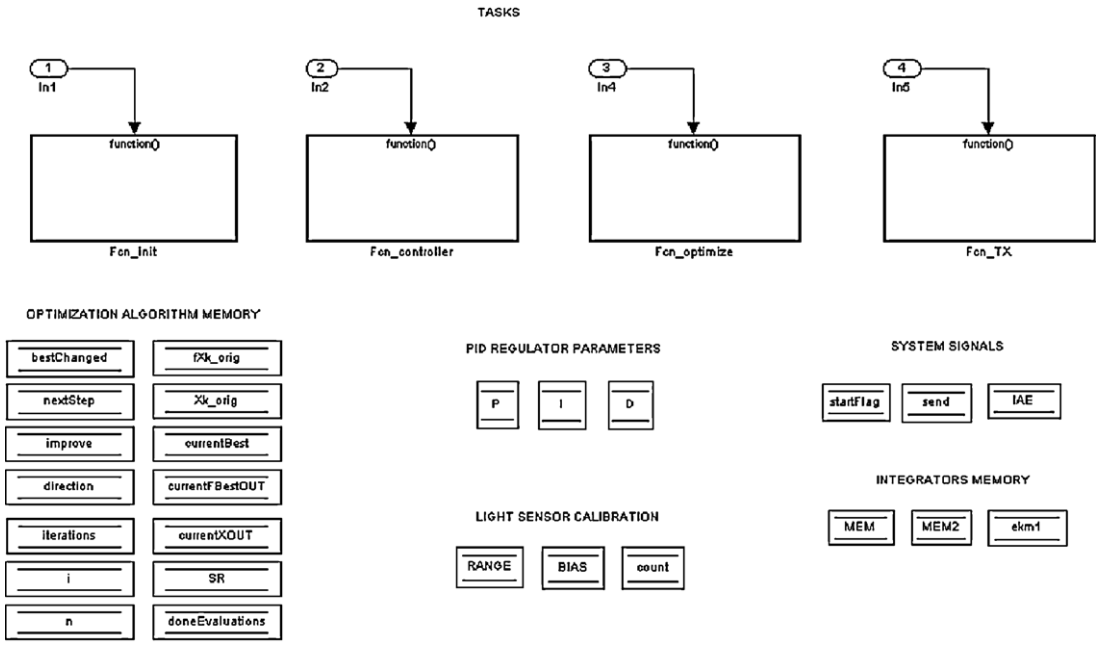


Fig. 5. Overview of the four tasks (top) and of the shared memories (bottom).

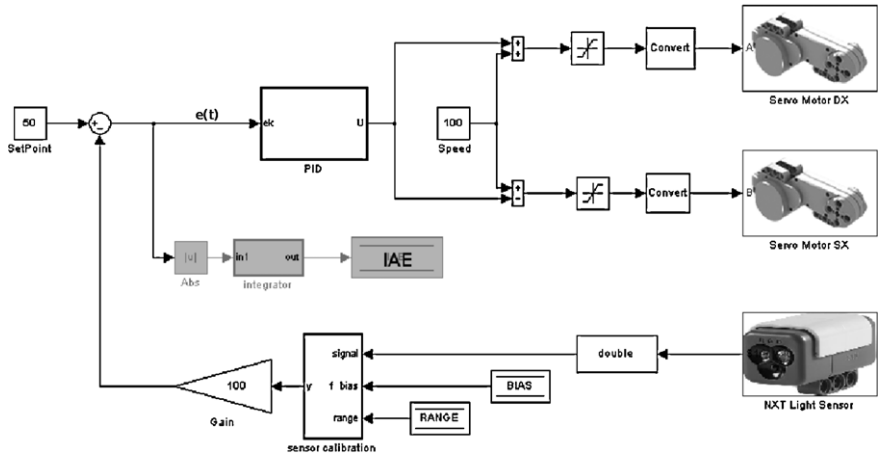


Fig. 6. The control scheme.

environment, the NXT returns the maximum light value when it "sees" white and the minimum value when black is "seen". These values are represented in a scale [0,100], but rarely an uncalibrated sensor would return them correctly. In our experiments, e.g. we measured values in the range [30,70].

By calibrating the NXT sensor, we instead allow the robot to run in different environments without changing the program code. The calibration procedure consists of the following. One of the two wheels is kept still while the other one moves with a very low constant speed. In this way the robot describes an arc, passing through the black line. During this trajectory the robot sees a completely white zone, a completely black zone and all the intermediate values. At every step, two memory areas are updated: one with the highest read value, and one with the lowest one. These values are used to evaluate the dynamic range of the sensor and its bias factor, and then are stored into two shared memories, called *BIAS* and *RANGE*.

3.3.2. Control scheme

The motion of the robot is controlled by the scheme shown in Fig. 6. At every step, the input data coming from the light sensor is subtracted from the set-point value, to generate the error signal $e(t)$.

Starting from $e(t)$, a control signal $u(t)$ is built by means of a PID regulator. As we can see in Fig. 6, without considering the $u(t)$ contribution, both the motors (right and left) are set to the maximum value of 100 (the motor speed can be set in the range [−100,100]). Two saturation blocks are included in the scheme to limit the speed range to [0,100], and to ensure that the motors move always in the same direction. The set-point value is equal to 50; it follows that $e(t)$ becomes zero only if the robot is exactly in the middle between the black line and the white space. This is because such position involves a light reading of 50%. When this situation occurs, also the signal $u(t)$ is zero, so that the robot moves forward at the maximum speed. Note that $u(t)$ is added to the right motor speed but is subtracted from the left one. For this reason, when the robot deviates from the correct position, and therefore the absolute value of $u(t)$ increases, the robot will turn to the right or left. In particular the robot will turn to right for $u(t)) > 0$, and to left if $u(t) < 0$.

For the sake of clarity, the PID regulator is a simple control strategy widely used in industrial plants, which takes into account three error contributions, weighted by three parameters, namely the proportional, integral and derivative gains. Intuitively, the three contributions can be interpreted in terms of time as follows:

1 a contribution depending on the current error;
2 a contribution depending on the accumulation of past errors;
3 a contribution depending on a prediction of future errors, based on the current rate of change.

The weighted sum of these three terms is used to control a process. The simplest (continuous) form of the PID is:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{d}{dt} e(t) \quad (1)$$

where K_p, K_i and K_d are, respectively, the aforementioned proportional, integral and derivative gain. The proportional term makes a change to the output that is proportional to the current error value. The proportional response can be adjusted by multiplying the error by K_p. A high proportional gain results in a large change in the output for a given change in the error. If the proportional gain is too high, the system can become unstable. In contrast, a small gain results in a small output response to a large input error, and a less responsive or less sensitive controller. If the proportional gain is too low, the control action may be too small when responding to system disturbances. The contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral in a PID controller is the sum of the instantaneous errors over time and gives the accumulated offset that should have been corrected previously. The integral term accelerates the movement of the process towards set-point and eliminates the residual steady-state error that occurs with a pure proportional controller. However, since the integral term responds to accumulated errors from the past, it can cause the present value to overshoot the set-point value. The derivative term slows the rate of change of the controller output. Derivative control is used to reduce the magnitude of the overshoot produced by the integral component and improve the combined controller-process stability. However, the derivative term slows the transient response of the controller. Also, differentiation of a signal amplifies noise and thus this term in the controller is highly sensitive to noise in the error term, and can cause a process to become unstable if the noise and the derivative gain are sufficiently large.

By tuning the three parameters in the PID algorithm, the controller can provide an at least sub-optimal control action designed for specific process requirements. There exist classical techniques to properly tune these parameters, starting from some knowledge or approximation of the plant to be controlled or from desired closed-loop properties. However, when this knowledge is not available, a trial and error process should be used, see e.g. the Ziegler–Nichols method [9]. Although these methods are proven, they often require some experience of the process and in some cases

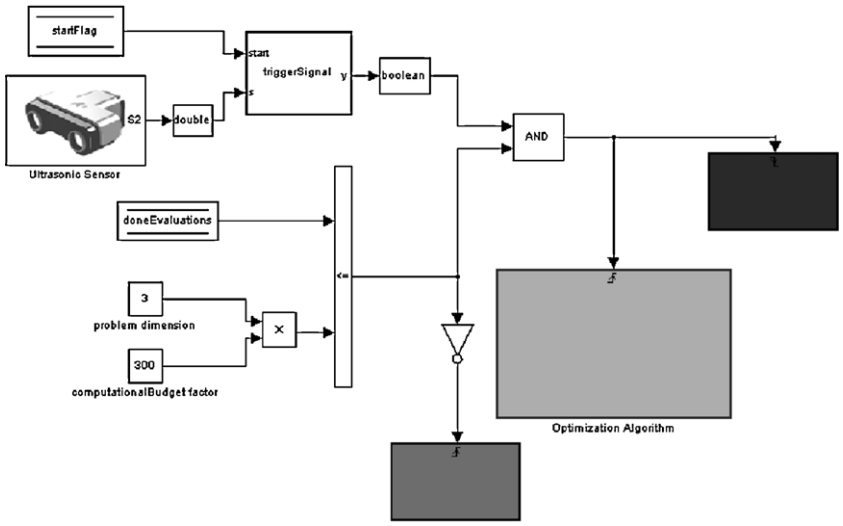


Fig. 7. The optimization task.

they are not robust enough. In the last three decades, automated PID tuning methods have become increasingly popular. These methods often involve the employment of an optimization algorithm which minimizes a given error measure. They are robust, versatile, and most of all they do not require any knowledge of the process. The present work falls in this class of methods.

With reference to the robot application, the PID controller has been implemented using an embedded Matlab function. The resulting algorithm is a discretized version of the formula (1). The three terms have been approximated as follows:

$$\begin{cases} e(t) = e[k]; \\ T_s \displaystyle\int_0^t e(\tau)\,d\tau \approx \sum_{k=1}^{n} e[k]; \\ \dfrac{d}{dt}e(t) \approx \dfrac{e[k-1]-e[k]}{T_s}; \end{cases} \tag{2}$$

where $t = k \cdot T_s$, being T_s the sample time of the system (1 ms), and $e[k]$ and $e[k-1]$ are, respectively, the error at the kth and $k-1$th step.

As mentioned before, the goal of the optimization is to find the optimal configuration of K_p, K_i and K_d which gives the minimum value for the integral absolute error (IAE), defined as follows:

$$\mathrm{IAE} = \int_0^{\infty} |e(t)|\,\mathrm{dt} \tag{3}$$

So, every T_s ms, a discrete version of IAE (i.e. with the sum operator instead of the integral) is updated and its value is stored into a shared memory. At the end of every path iteration, the optimization task (see the next subsection) reads the IAE from this memory and uses it as fitness value.

A final remark should be done on this application. From the scheme described above, it is clear that the error signal is proportional to the distance from the black line, measured in term of brightness percentage. For this reason, from a control perspective a P or a PI controller would be enough. Nevertheless, in order to test the general applicability of the proposed approach, we decided to let the algorithm tune a PID regulator.

3.4. Task 3: optimization

This task is responsible for the execution of the optimization algorithm. Every time the algorithm computes a new set of PID parameters, it writes their values on three additional shared memories, so that the control system task reads them and tries a new PID controller configuration. Fig. 7 shows that the optimization block is executed in a triggered mode. The trigger signal comes from the ultrasonic sensor, which signals the completion of a path iteration. Since the ultrasonic sensor has a fixed 20 ms refresh rate, the frequency of this task depends on this value. In order to obtain a fast and efficient control, we decided to separate the control system, which is executed at a higher frequency, from the optimization task, which is slower. The trigger signal is stored into the *startFlag* memory. Each time the scheduler activates Task 3, the value of *startFlag* is checked: if the system is ready and the ultrasonic sensor "sees" that the robot has returned to its original position (more realistically in a neighbourhood of the initial position), a step the optimization algorithm is executed and a new PID configuration is computed. Otherwise, the tasks are idle.

The search spaces for K_p, K_i and K_d have been set, respectively, to [2,20], [0,5], [0,0.5]. These bounds have been empirically chosen to guarantee the proper (i.e. stable) functioning of the system. It must be noted that the search space is asymmetric: the proportional term cannot be null (otherwise there would not be any control action), but its search space is larger. The other two terms however may be equal to zero, in order to obtain the PI or the PD configurations, but they cannot be very high, otherwise the system would become unstable.

To keep track of the elapsed iterations, a counter (*doneEvaluations*) is updated. As soon as the value of *doneEvaluations* reaches the computation budget, i.e. the number of fitness evaluations allotted to the optimization process (see Section 5), the optimization algorithm is stopped. At this point, the gray block in Fig. 7 is executed, which reinitializes the PID controller with the optimal set of parameters found so far, thus bypassing the optimization algorithm.

The rightmost block in Fig. 7, instead, is used to reinitialize all the shared memories that need a reset whenever a path repetition is completed, namely those ones used to store the IAE, the PID parameters, and all the integrators memories. In addition, after each step of the optimization algorithm, this block sets the *send* flag to zero. This flag is used by the data logging task (see next section), to trigger the sending of the data via Bluetooth. By means of this flag, it is possible to avoid unnecessary transmissions, which would result in an excessive consumption of the batteries of the robot. Both the optimization and reinitialization blocks are triggered by the same signal (Fig. 6). However, the first is performed

during the rising phase, the second during the falling phase of the trigger signal. Considering that *send* is set to one at the end of the optimization task, the Bluetooth transmission is enabled only for a short while.

3.5. Task 4: data logging

This task handles the Bluetooth communication used to log the data of the optimization process. Just like the optimization task, whose refresh frequency is limited by the ultrasonic sensor, the period of this task is 20 ms. Since the Lego Bluetooth device has a high energy consumption, it is necessary to handle it efficiently in order to increase the battery life of the robot. Thus, although the Bluetooth connection is always active during the optimization process, packets are transmitted to the logging PC only when new data are available, i.e. a new PID configuration is computed and evaluated. To do this, all the data to transmit are stored into a shared memory, and only at the end of every iteration of the path data are transmitted. Each time the data logging is executed, the *send* flag is checked to decide whether transmission should take place or not, and the data packet is created and sent accordingly. On the logging PC at the other end, a Matlab script is used to keep the Bluetooth communication with the robot active, receive data and log them in real-time.

4. Three-stage optimal memetic exploration

To solve the IAE minimization problem described in the previous section, the three stage optimal memetic exploration (3SOME) algorithm, introduced in [10], has been considered. 3SOME is a single-solution algorithm, based on the interaction of three memes. Despite its simplicity 3SOME proved to have a respectable performance on a large amount of test problems and be competitive with complex modern algorithms. Each of the three memes composing the 3SOME algorithm provides different search capabilities, and the coordination of the three of them guarantees a high chance of reaching optimal solutions. Although 3SOME exploits the concept of meme, it is not a proper MA, but rather it can be considered a memetic computing approach. MC is a "paradigm that uses the notion of meme(s) as units of information encoded in computational representations for the purpose of problem-solving", where meme is an abstract concept which can be, e.g. a strategy, an operator, or a search algorithm. In this sense MC is a much broader concept with respect to a MA. As just said before, the 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.

More specifically, 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-like operator (see Algorithm 1). 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.

Algorithm 1 *(Long distance exploration pseudo-code).*

```
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
```

Algorithm 2 *(Middle distance exploration pseudo-code).*

```
hypercube with side width δ centred 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
```

Algorithm 3 *(Short distance exploration pseudo-code).*

```
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
```

During the middle distance exploration, an 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 Algorithm 2). 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.

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 Algorithm 3. 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. In a nutshell the short distance exploration is a simple steepest descent deterministic local search algorithm, with an exploratory logic similar to that of Hooke–Jeeves algorithm [11]. 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 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 Algorithm 4.

Algorithm 4 *(Coordination of the exploration stages pseudo-code).*

```
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 Alg. 1
  end
  while x_e is updated do
    apply to x_e the middle distance exploration as in Alg. 2
  end
  apply to x_e the short distance exploration as in Alg. 3
  if x_e has been updated then
    apply middle distance exploration as in Alg. 2
  else
    apply long distance exploration as in Alg. 1
  end
end
```

5. Experimental results

In this section we present the experimental results we obtained on the IAE minimization problem described above. A first simulation campaign was conducted in order to test the applicability of the proposed approach and draw some preliminary conclusions about the performance of 3SOME on the specific optimization problem at hand. To further strengthen this initial analysis, results obtained with the 3SOME algorithm were compared with the results obtained with two classical optimization local search algorithms, four modern memory-saving global optimization algorithms. The simulation results were then validated executing the path-following application online on a real Lego Mindstorms robot. For the sake of efficiency, only the algorithms displaying the best performances in simulation were selected for real-world implementation.

5.1. Noise estimation

In order to make the simulations as close as possible to the real-world robot behaviour, we first of all tried to model the system noise. The presence of noise in sensors and the hardware precision (e.g. in actuators) heavily affect results. From the optimization perspective, this means that it is not possible to distinguish the average performances of two algorithms if their difference is smaller than the noise level, regardless of the choice of the PID parameters. Therefore an accurate estimation of the noise is needed. To estimate the noise level, we performed a set of 100 paths repetitions, maintaining the same set of parameters for the PID regulator (tuned manually). As shown in Fig. 8, we measured that the IAE nominal value is affected by a Gaussian noise with zero mean and standard deviation $\sigma = 2.8$.

5.2. Simulation results

In the first test phase, the control system described in Section 3 and depicted in Fig. 4 was simulated using Matlab Simulink and the ECRobot NXT toolbox. The noise model described in the previous subsection was used, in simulation, to inject artificially a noisy component into the fitness value. The 3SOME algorithm described before was executed with inheritance factor $\alpha_e = 0.05$, width of the hypercube for middle distance exploration $\delta = 20\%$ of the total decision space width, coefficient of generated points $k = 4$, and initial exploratory radius for short distance exploration $\rho = 40\%$ of the total decision space width. 3SOME was compared with the following classical local search algorithms, with their original parameter setting:

- Hooke–Jeeves algorithm [11], with tolerance $\varepsilon = 1e-6$ and step reduction coefficient $\alpha = 2$.
- Nelder–Mead algorithm [12], with reflection coefficient $\rho = 1$, expansion coefficient $\chi = 2$, contraction coefficient $\gamma = -1/2$ and shrink coefficient $\sigma = -1/2$.

In addition to these two methods, four memory-saving algorithms recently proposed in the literature were selected for comparison. More specifically, the algorithms hereinafter listed, with the parameter setting suggested in the original papers, have been considered:

- Simplified intelligence single particle optimization (ISPO) proposed in [13] with acceleration $A = 1$, acceleration power factor $P = 10$, learning coefficient $B = 2$, learning factor reduction ratio $S = 4$, minimum learning threshold $E = 1e-5$, and learning steps $PartLoop = 30$.
- Non-uniform simulated annealing (nuSA) proposed in [14] with mutation exponent $B = 5$, temperature reduction ratio $\alpha = 0.9$, temperature reduction period $L_k = 3$, and number of initial solutions to set the initial temperature $initialSolutions = 10$.
- Compact differential evolution with rand/1 mutation and exponential crossover, cDE/rand/1/exp, here simply indicated as cDE, proposed in [15]. The cDE algorithm has been run with virtual population size equal to 300, scale factor $F = 0.5$, and proportion of genes undergoing exponential crossover, see [16], $\alpha_m = 0.25$.
- Compact differential evolution light (cDELight), proposed in [17]. The virtual population size has been set equal to 300, scale factor $F = 0.5$, and proportion of genes undergoing crossover, see [16], $\alpha_m = 0.25$.

It should be noticed that all the algorithms listed above were implemented in Stateflow, so that the same Simulink scheme shown in Fig. 4 (except the data logging task, which was not used in simulation), could be used both for simulation and for real-world tests. After the simulation stage, the Simulink scheme (this time including the data logging task) was used to automatically generate the C code optimized for the ARM7 architecture. This code was then cross-complied and executed on the target micro-controller of the robot to perform the real-world experiments described in the next subsection.

For each algorithm, 30 simulated runs have been performed, each one consisting of 300 complete repetitions of the closed path to follow. This value of computational budget comes from theoretical and practical considerations. Firstly, it is related to the complexity of the problem, which has only three parameters (a higher dimensional problem would require a higher number of fitness evaluations). In addition to that, since in the real-world case (see below) each repetition of the path took, on average, about 15 s (using the 80% of the robot maximum velocity), the budget was bounded essentially by the battery duration. Especially due to the Bluetooth communication used for logging data, which is extremely energy consuming, in the real-world case the system had about a 3-h battery life.

As an additional remark, it should be noticed that the algorithms listed above represent a fairly comprehensive selection of classical optimization algorithms and state-of-the-art memory-saving

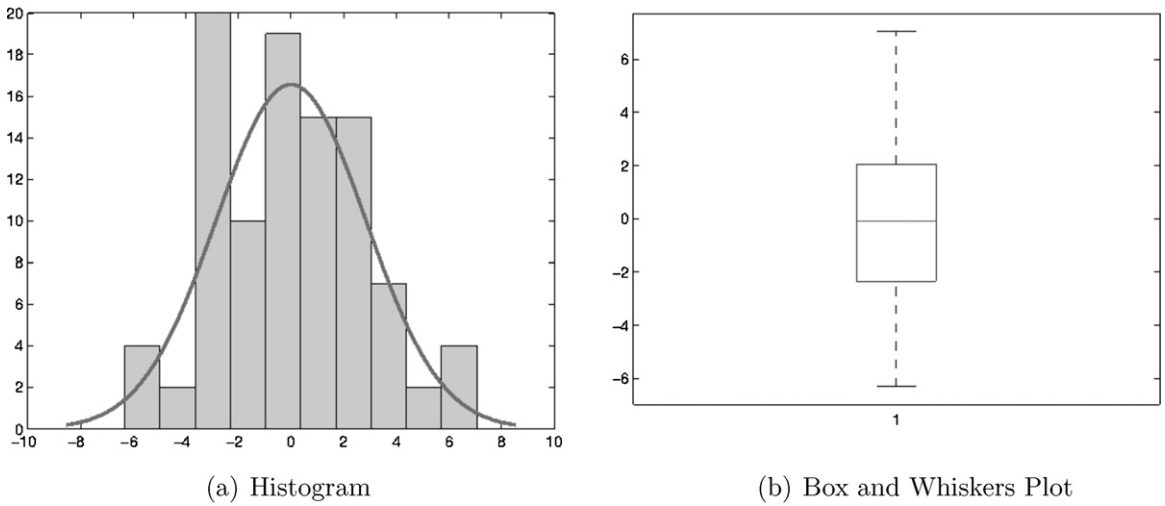


Fig. 8. Noise on the integral absolute error.

algorithms. Considering in particular the latter algorithms, it is important to understand that they use different nature-inspired metaphors. More specifically, ISPO is Particle Swarm [18,19] algorithm in which the "swarm" is reduced to a single particle; nuSA is an advanced version of the popular stochastic algorithm named simulated annealing [20]; cDE and cDELight are two compact (i.e. based on a probabilistic model of the population, see EDAs) implementations of the popular evolutionary algorithm named differential evolution [21–23]. In particular, cDELight maintains the same structural features of cDE, but with a reduced computational footprint. Finally, 3SOME is memetic framework which shares some ideas with Variable Neighbourhood Search [24,25] and Iterated Local Search [26] algorithms.

Fig. 9 shows the average fitness trend obtained in simulation with the seven algorithms considered. The final values of IAE ± standard deviation are reported in Table 1. Also the output of the Wilcoxon rank-sum test [27] is shown, applied with a confidence level of 0.95. The symbol "=" indicates a statistical equivalent performance, while "+" indicates a superior performance of 3SOME with respect to the algorithm on the selected row in the table. From the fitness trend plot, it can be seen that all the selected algorithms are able to improve upon the initial solution of the problem at hand. However, 3SOME emerges as the best of the algorithms considered in this test, since it outperforms particularly ISPO and the two classical local search algorithms (i.e. the Hooke–Jeeves and Nelder–Mead methods), while it shows similar performance to nuSA, cDE and cDELight. This result can be explained considering that, despite its limited dimensionality, the fitness function measuring the IAE is highly multi-modal; moreover, it is affected by noise. Due to the combination of these two landscape features, extremely exploitative algorithms (see ISPO) and local search methods (such as Hooke–Jeeves and Nelder–Mead) are not particularly efficient on this specific problem. Instead these algorithms produce very few fitness improvements (especially ISPO), clearly suffering from premature convergence. In addition to that, it is interesting to notice that the Nelder–Mead method requires more memory than all the other algorithms considered in this study, since it needs to store $n+1$ solutions (the simplex), where n is the problem dimensionality. While in this particular 3-dimensional problem this memory footprint is still acceptable, a higher dimensional problem (e.g. a more complex controller, with more parameters) would require a much higher memory capacity, thus making the Nelder–Mead method not memory-wise efficient. On the other hand, modern global memory-saving optimization algorithms (3SOME, nuSA, cDE and cDELight), despite their simplicity, are able to handle the multimodal noisy fitness landscape of the IAE more efficiently than ISPO and the two classical local search methods. Additionally, they have a very similar performance, as indicated by the Wilcoxon test in Table 1.

For the sake of completeness, the best PID parameters obtained with the aforementioned algorithms are reported in Table 2: it is interesting to notice how 3SOME and cDE, as well as Hooke–Jeeves, tend to a very high value of K_p (almost close to its upper bound), although the average performance of Hooke–Jeeves is worse. Similar considerations can be made about the pairs nuSA/ISPO and cDELight/Nelder–Mead, which converge to lower values of K_p. On the other hand, the best values of K_i and K_d differ a lot from one algorithm to another, also when the average performance is the same. In other words very small differences in the parameter space produce very different values of IAE, while very different sets of PID parameters produce PID regulators which guarantee an equally low level of IAE. This diversity in the best PID parameters can be seen as an indirect evidence of the multi-modality of fitness landscape at hand. Together with the noise which affects the IAE, this feature of the landscape justifies the need for a robust global optimizer which

Table 1
Average final IAE obtained ± standard deviation and Wilcoxon test outcome (reference: 3SOME) on the simulation experiments.

Algorithm	IAE	Std. Dev.	W
3SOME	80.7073	12.0787	
nuSA	81.2889	16.9048	=
cDE	77.6358	2.9000	=
cDELight	78.4746	4.0249	=
ISPO	143.1742	21.3450	+
Hooke–Jeeves	93.7439	30.4175	+
Nelder–Mead	101.8764	27.8688	+

Table 2
Best PID parameters obtained on the simulation experiments.

Algorithm	K_p	K_i	K_d
3SOME	17.5992	4.2104	0.0920
nuSA	2.3710	1.3213	0.4103
cDE	18.2708	0.3255	0.2481
cDELight	13.1273	0.1454	0.1434
ISPO	2.1263	0.5705	0.2970
Hooke–Jeeves	16.9844	2.5159	0.1724
Nelder–Mead	12.5015	2.5573	0.2127

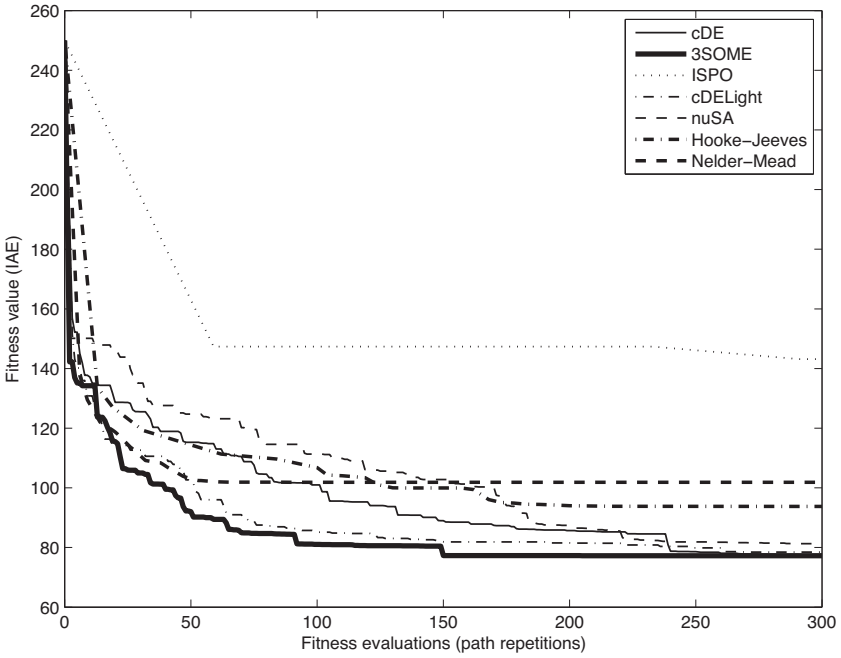


Fig. 9. Average performance trends of the algorithms considered in the simulation experiments.

is able to explore efficiently the landscape and conclude the search with a fine tuning of the parameters.

5.3. Real-world results

In the real-world case, due to long execution time and battery life, for each algorithm only eight complete runs have been performed, consisting again of 300 path iterations. As anticipated in the previous section, the same Simulink control scheme employed for simulations was used for generating the binary ARM7-specific file executed on the Lego Mindstorms robot. Only the algorithms displaying the best performance in simulation, namely the four modern memory-saving algorithms (3SOME, nuSA, cDE and cDELight), were selected for real-world implementation and tests. For each algorithm, the same parameter setting used in simulation was selected.

Fig. 10 shows the fitness trends, averaged on eight algorithm repetitions, obtained logging data coming from the Lego Mindstorms NXT 2.0. The final values of IAE ± standard deviation are reported in Table 3, together with the result of the Wilcoxon rank-sum test. Recalling the noise analysis presented in Section 5.1, from these experimental results we can conclude that the pairs 3SOME-nuSA and cDE-cDELight have a practically indistinguishable performance (as confirmed also by the Wilcoxon test), although 3SOME is able to provide an average lower value of IAE, noise-wise more robust than cDE and cDELight.

In order to give a practical interpretation of these results, let us remember that the robot requires, on average, 15 s to perform the entire path once. The sensor samples the error every millisecond. Thus, on average, totally 15,000 samples are performed. Each sample corresponds to a measure of error between −50 (sensor sees all white) and 50 (sensor sees all black). Due to the application of the absolute values, see formula (3), each contribution to the IAE can be between 0 and 50. Considering that the IAE is composed of 15,000 contributions, numerical results in Table 3 show that all the algorithms considered in this study, especially 3SOME, display a very good performance in terms of IAE minimization. In other words, the proposed self-optimized control system allows the execution of the path-following task very precisely, despite the very limited computational power and sensor precision of the toy robot used in this work.

Also in this case we report the best PID parameters obtained with selected algorithms (Table 4). Similarly to the simulated case, 3SOME and cDE tend to a very high value of K_p, while nuSA and cDE-light converge to lower values of K_p. On the other hand, only cDE converges to a value of K_i close to its upper bound, while 3SOME, nuSA and cDELight converge to quite different values ranging from 0.8412 to 4.0621. Similar considerations can be made about K_d. It should be noticed that, due to the stochasticity of the algorithms and to simulation approximations, the numerical results are slightly different from the experimental ones (both in terms of convergence and final values). However, the general trends are the same, and similar conclusions about the features of the landscape (see the previous subsection) can be drawn.

5.4. Memory consumption

In order to have an indication of the memory consumption, in the real-world case the memory footprint for the selected

Table 3
Average final IAE obtained ± standard deviation and Wilcoxon test outcome (reference: 3SOME) on the real-world experiments.

Algorithm	IAE	Std. Dev.	W
3SOME	74.1579	11.7389	
nuSA	83.0540	1.5326	=
cDE	90.0341	7.0072	+
cDELight	87.433	3.9071	+

Table 4
Best PID parameters obtained on the real-world experiments.

Algorithm	K_p	K_i	K_d
3SOME	19.5414	4.0621	0.2802
nuSA	14.2172	3.7887	0.2972
cDE	18.2438	4.9695	0.3677
cDELight	7.6512	0.8412	0.2483

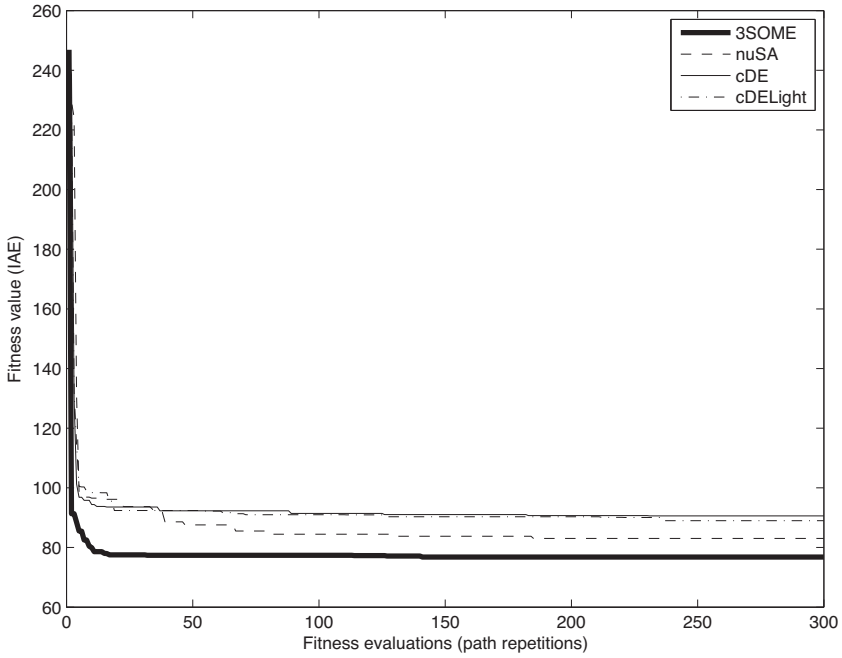


Fig. 10. Average performance trends of the algorithms considered in the real-world experiments.

algorithms was also measured. The following memory footprints where measured:

- 38.240 KB, with 3SOME:
- 37.400 KB, with nuSA;
- 43.664 KB, with cDE;
- 43.632 KB, with cDELight.

These measurements can be explained as follows. The entire control scheme application, without the optimization algorithm, occupies 28.528 KB of memory (part of which contains the BIOS, which is linked to the user application). When one of the four memory-saving algorithms is added, there is an additional memory overhead. With nuSA and 3SOME, this overhead is about 10 KB, while the cDE schemes require about 15 KB. These latter schemes are slightly more expensive, in terms of memory, due to the sampling mechanism which requires some mathematical libraries to be linked. In any case, it is important to notice that this limited memory footprint could not be achieved by any population-based algorithm.

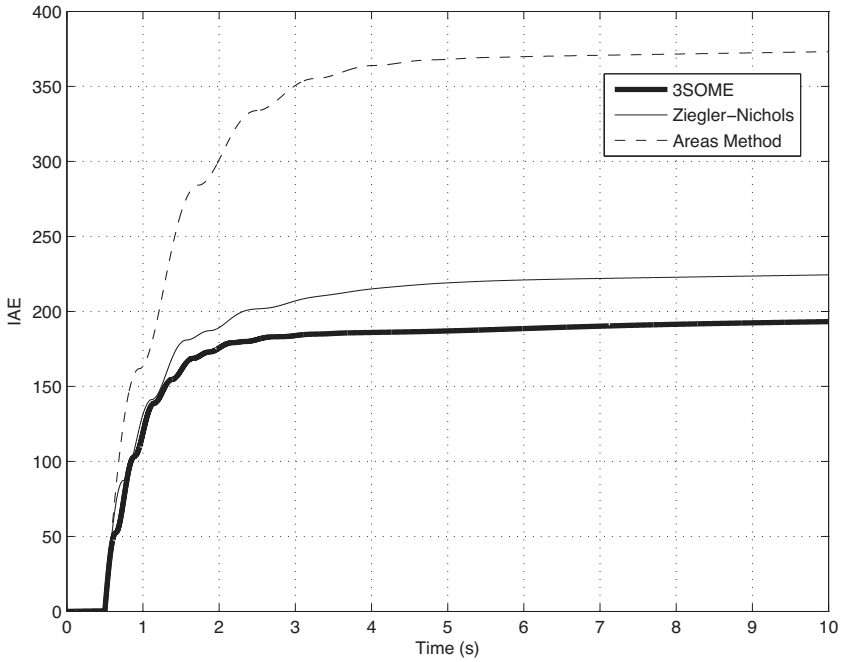


Fig. A.11. Comparison of the IAE trends obtained with the Ziegler–Nichols method, the areas method and the 3SOME algorithm (for the latter, the average trend over 30 runs is shown).

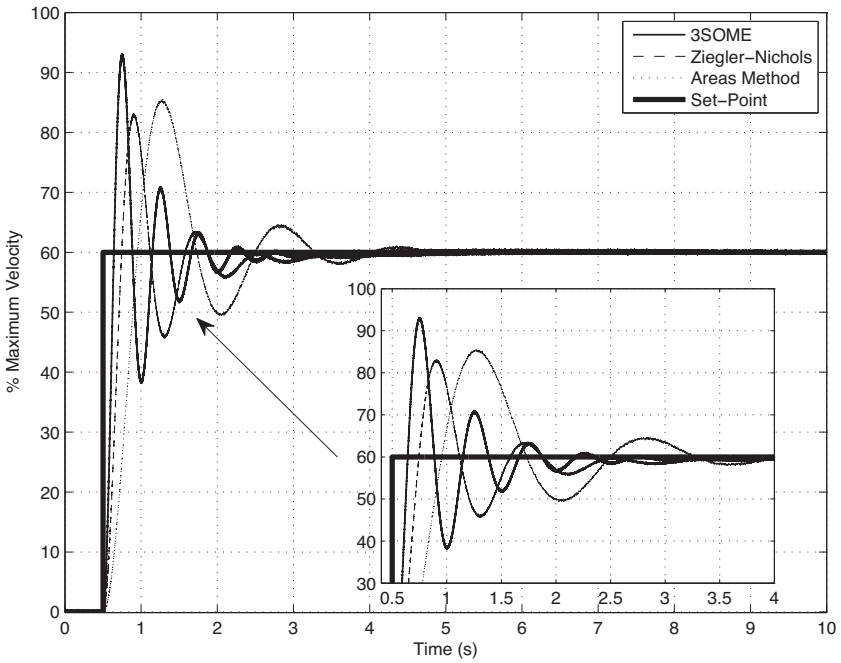


Fig. A.12. Comparison of the step response obtained with the Ziegler–Nichols method, the areas method and the 3SOME algorithm (for the latter, the best PID configuration found in 30 runs is considered).

6. Conclusions

In this paper we proposed an architecture for tuning on-line a PID controller on board of an embedded system, i.e. the ARM7 micro-controller of the Lego Mindstorms NXT 2.0 robot. A recently proposed memetic algorithm, namely 3SOME, has been used to minimize the IAE in a path-following robotic application. To validate the optimization results, two experiment campaigns have been performed: in the first one, a model of the system has been simulated with Simulink, and 3SOME was compared with two classical local search algorithms and four state-of-the-art memory saving algorithms employing different logics. In the second stage, the algorithms displaying the four best performances in simulation (including the 3SOME structure) have been implemented on a real Lego Mindstorms NXT 2.0 robot. Despite its simple structure and limited memory footprint, 3SOME proved extremely good at rapidly improving upon the initial solution and exploiting promising search directions. Among the selected algorithms, 3SOME was indeed able to find the minimum value of IAE.

Although it has been tested on popular commercial gadget, the proposed approach may be easily extended and applied in all those industrial contexts or engineering problems plagued by limited hardware. This study confirms our previous finding that in some specific applications characterized by poor hardware resources and real-time requirements, a very simple algorithm, if carefully designed, should be preferred to complex algorithms characterized by a large memory consumption and a high computational overhead. In other words, this work aims at showing that the future of artificial intelligence is in the developed of efficiently designed soft computing techniques and their integration in various every day life situations, even when the available hardware is very limited. The fact that, thanks to a highly performing soft computing technique, an embedded optimization over a toy allows the solution of a complex industrial problem at a professional level is an argument to support our intuition. Future works will likely go in this direction, further extending memetic structures composed of simple, memory-saving memes, whose topology and coordination strategy could be able to adapt to the specific optimization problem.

Acknowledgements

F. Caraffini is supported by TUCEP Tiber Umbria Comett Education Programme. F. Neri is supported by the Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing.

Appendix A. Comparison with empiric tuning methods

This section validates the control performance obtained with the proposed algorithmic approach by comparing it with traditional tuning methods. In order to carry out this comparison, we designed an additional experiment where the results obtained by means of the 3SOME algorithm are compared with two traditional tuning methods. It should be remarked that most of the tuning procedures known in the literature rely on a trial and error process and require the system under analysis to be tested in an open-loop configuration and/or excited with a step signal. However, both these conditions would turn out being rather impractical to be applied on the 2-wheels robotic system presented above. Recalling the control scheme presented in Fig. 6, it can be noticed that the system reference (and its feedback measure) is a percentage of black/white, while the input for the motors is a delta of velocity. Feeding this system with a "black/white step" would lead the robot to an improper functioning, since the path-following task, as it was implemented, would be impossible to be performed (this task needs, by definition, a constant input setting). Obviously, also an open-loop configuration might bring the robot to an uncontrolled condition. For this reason, and in order to draw some intuitive physical conclusions, we focused our analysis on a single NXT motor fed with a velocity step, rather than the rover robot described in the previous sections. A dynamic model of the NXT motor, obtained via experimental measures and available in [8,28], was used to tune a PID regulator. The model also includes an accurate model characterization. The motor was excited with a velocity step of amplitude $U = 60$ (% maximum velocity) starting at $\tau = 0.5$ s. All the experiments were performed simulating the system with Matlab Simulink.

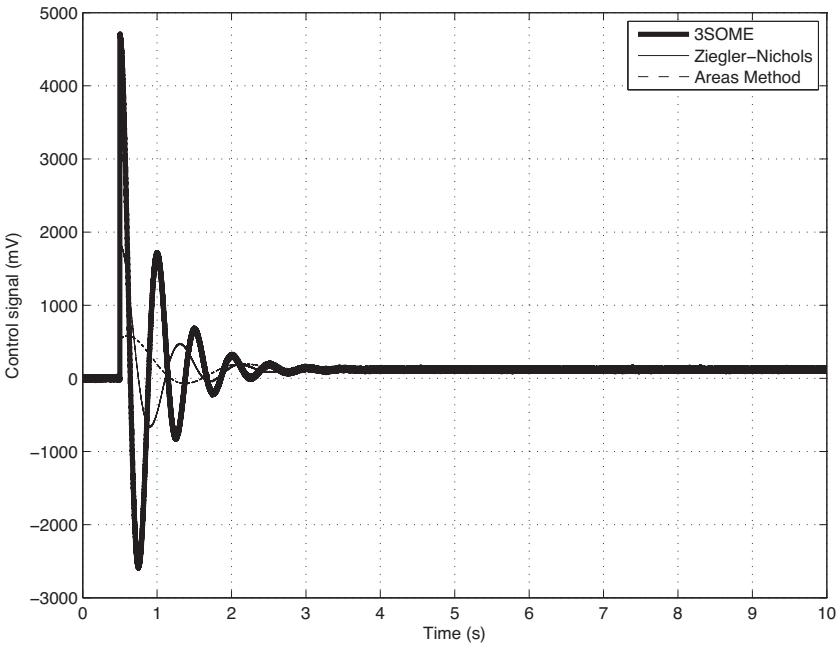


Fig. A.13. Comparison of the control signal generated with the Ziegler–Nichols method, the areas method and the 3SOME algorithm (for the latter, the best PID configuration found in 30 runs is considered).

Two empirical tuning methods were used and compared with the 3SOME-based self-tuning approach, namely the Ziegler–Nichols open-loop tuning method (also called process reaction method) [9] and the areas method [29]. Both the methods approximate the system under study with an asymptotically stable, strictly proper first order plus dead time (FOPDT) model:

$$G(s) = \frac{\mu}{1+sT} e^{-\tau \ s} \tag{A.1}$$

where μ, T and τ are respectively the static gain, the time constant and the delay of the system. The step response of the FOPDT system can be analytically expressed as follows:

$$y(t) = \mu \ \ U(1 - e^{(t-\tau/T)}) \cdot 1(t-\tau) \tag{A.2}$$

where $1(t-\tau)$ indicates a unit step starting at time τ. In both the Ziegler–Nichols and the areas methods, the open-loop step response (process reaction curve) of the real system is graphically analyzed to determine some physical characteristics of the system. The PID parameters are empirically determined by these features.

In the Ziegler–Nichols method, starting from the process reaction curve the following system parameters are determined:

- the dead time (delay) τ, defined as the time at which the system response starts;
- Y_∞, the asymptotic ultimate value that the system reaches at steady-state;
- the time constant T, obtained as difference between the time intercept of the tangent to the reaction curve in τ with the asymptotic line $y = Y_\infty$.

The PID constants are then empirically set as $K_p = 1.2U/\eta$, $K_i = K_p/(2.0T)$ and $K_d = 0.5K_pT$, where $\eta = \tau\, Y_\infty/T$. Applying this procedure to the NXT motor, we measured $Y_\infty = 30$, $\tau = 0.1$ and $T = 0.8823$. Hence, being $\eta = 3.4$, we obtained $K_p = 21.1765$, $K_i = 11.9994$ and $K_d = 9.3431$. The IAE, measured over a time period of 10 s, was 246.0139.

Similarly to the Ziegler–Nichols approach, the areas method determines from a graphical analysis of the open-loop step response of the system two "characteristic areas":

- $S_1 = \int_0^\infty [Y_\infty - y(t)]\, dt$, i.e. the area between the asymptotic line $y = Y_\infty$ and the process reaction curve;
- $S_2 = \int_0^{\tau+T} y(t) dt$, i.e. the area "below" the process reaction curve limited by $t = \tau + T$.

In this case the time constant T is computed as $S_2 e/Y_\infty$, while the delay τ is set equal to $(S_1 - S_2 e)/Y_\infty$. Defining the static gain $\mu = Y_\infty/U$, the values of K_p, K_i and K_d are then computed, respectively, as $K_p = 1.2T/(\mu\ \tau)$, $K_i = K_p/(2.0\tau)$. In our case, we measured $S_1 = 46.1235$ and $S_2 = 8.7900$. Thus it resulted $T = 0.7965$ and $\tau = 0.7409$. Subtracting from this value of τ the delay of the input signal (0.5 s) we obtained $\tau = 0.2409$. Finally, being $\mu = 0.5$, we calculated $K_p = 7.9352$, $K_i = 16.4699$ and $K_d = 0.9562$, with an IAE of 373.1705 measured over 10 s.

The results obtained with the two methods described above were then compared with the average results obtained by means of 3SOME. More specifically, 30 independent runs of the algorithm were performed, each one consisting of 300 fitness evaluations. The bounds for the three parameters K_p, K_i and K_d were kept purposely large [0,50], in order to test the robustness of the 3SOME algorithm and to explore a larger search space. A comparison of the IAE trend in a time frame of 10 s obtained by means of three methods is reported in Fig. A.11, where it can be clearly seen that 3SOME is able to converge to a much lower value of IAE compared to the two empiric methods. For the sake of completeness, we also report the step response (Figs. A.12 and A.13) and the control signal (Fig. A.13) measured with the three methods: in case of 3SOME, we considered the best PID configuration found in 30 runs. Although the control signal generated by the PID obtained with 3SOME is slightly higher, it can be seen intuitively how the proposed approach guarantees a more rapid response, with a shorter settling time. This result confirms that the proposed approach represents a valid alternative for self-tuning control systems.

References

[1] P.G. Norman, The new AP101S general-purpose computer (GPC) for the space shuttle, IEEE Proceedings 75 (1987) 308–319.
[2] J. Zhang, A.C. Sanderson, JADE: adaptive differential evolution with optional external archive, IEEE Transactions on Evolutionary Computation 13 (2009) 945–958.
[3] R. Hollier, Automated Guided Vehicle Systems, International Trends in Manufacturing Technology, IFS Publications, London, UK, 1987.
[4] T. Müller, Automated Guided Vehicles, IFS Publications, London, UK, 1983.
[5] A. Prügel-Bennett, Benefits of a population: five mechanisms that advantage population-based algorithms, IEEE Transactions on Evolutionary Computation 14 (2010) 500–517.
[6] D. Wolpert, W. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1997) 67–82.
[7] Lego, Lego Mindstorms, 2012 http://mindstorms.lego.com
[8] T. Chikamasa, nxtOSEK/JSP – ANSI C/C++with OSEK/μITRON RTOS for Lego Mindstorms NXT, 2012 http://lejos-osek.sourceforge.net
[9] J.G. Ziegler, N.B. Nichols, Optimum settings for automatic controllers, Transactions of ASME 64 (1942) 759–768.
[10] G. Iacca, F. Neri, E. Mininno, Y.-S. Ong, M.-H. Lim, Ockham's razor in memetic computing: three stage optimal memetic exploration, Information Sciences 188 (2012) 17–43.
[11] R. Hooke, T.A. Jeeves, Direct search solution of numerical and statistical problems, Journal of the ACM 8 (1961) 212–229.
[12] A. Nelder, R. Mead, A simplex method for function optimization, Computation Journal 7 (1965) 308–313.
[13] J. Zhou, Z. Ji, L. Shen, Simplified intelligence single particle optimization based neural network for digit recognition, in: Chinese Conference on Pattern Recognition, 2008 (CCPR'08), 2008, pp. 1–5.
[14] Z. Xinchao, Simulated annealing algorithm with adaptive neighborhood, Applied Soft Computing 11 (2011) 1827–1836.
[15] E. Mininno, F. Neri, F. Cupertino, D. Naso, Compact differential evolution, IEEE Transactions on Evolutionary Computation 15 (2011) 32–54.
[16] F. Neri, G. Iacca, E. Mininno, Disturbed exploitation compact differential evolution for limited memory optimization problems, Information Sciences 181 (2011) 2469–2487.
[17] G. Iacca, F. Caraffini, F. Neri, Compact differential evolution light, Journal of Computer Science and Technology 27 (2012) 1056–1076.
[18] R.C. Eberhart, J. Kennedy, A new optimizer using particle swarm theory, in: Proceedings of the Sixth International Symposium on Micromachine and Human Science, 1995, pp. 39–43.
[19] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.
[20] S. Kirkpatrick, C.D.J. Gelatt, M.P. Vecchi, Optimization by simulated annealing, Science (1983) 671–680.
[21] R. Storn, K. Price, Differential Evolution – A Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces, Technical Report TR-95-012, ICSI, 1995.
[22] K.V. Price, R. Storn, J. Lampinen, Differential Evolution: A Practical Approach to Global Optimization, Springer, Berlin Heidelberg, Germany, 2005.
[23] U.K. Chakraborty (Ed.), Advances in Differential Evolution, vol. 143 of Studies in Computational Intelligence, Springer, 2008.
[24] N. Mladenović, P. Hansen, Variable neighborhood search, Computers and Operations Research 24 (1997) 1097–1100.
[25] P. Hansen, N. Mladenović, Variable neighborhood search: principles and applications, European Journal of Operational Research 130 (2001) 449–467.
[26] H. Lourenço, O. Martin, T. Stützle, Iterated local search, in: F. Glover, G. Kochenberger (Eds.), Handbook of Metaheuristics, vol. 57 of International Series in Operations Research & Management Science, Springer, New York, 2003, pp. 320–353.
[27] F. Wilcoxon, Individual comparisons by ranking methods, Biometrics Bulletin 1 (1945) 80–83.
[28] P.E. Hurbain, NXT Motor Internals, 2012 http://www.philohome.com/nxtmotor/nxtmotor.htm
[29] Y. Nishikawa, N. Sannomiya, T. Ohta, H. Tanaka, A method for auto-tuning of PID control parameters, Automatica 20 (1984) 321–332.

PIV

MICRO-DIFFERENTIAL EVOLUTION WITH EXTRA MOVES ALONG THE AXES

by

Fabio Caraffini, Ferrante Neri and Ilpo Poikolainen 2013

IEEE Symposium on Differential Evolution (SDE), pages 46-53

Micro-Differential Evolution with Extra Moves Along the Axes

Fabio Caraffini and Ferrante Neri
Centre for Computational Intelligence,
School of Computer Science and Informatics,
De Montfort University,
The Gateway, Leicester LE1 9BH, United Kingdom
Email: fabio.caraffini@email.dmu.ac.uk and fneri@dmu.ac.uk
Department of Mathematical Information Technology,
P.O. Box 35 (Agora), 40014
University of Jyväskylä, Finland
Email: fabio.caraffini@jyu.fi and ferrante.neri@jyu.fi

Ilpo Poikolainen
Department of Mathematical Information Technology,
P.O. Box 35 (Agora), 40014
University of Jyväskylä, Finland
Email: ilpo.poikolainen@jyu.fi

***Abstract*—This paper proposes a novel implementation of micro-Differential Evolution (μDE) that incorporates within the DE scheme an extra search move that attempts to improve the best solution by perturbing it along the axes. These extra moves complement the DE search logic and allows the exploration of the decision space from an alternative perspective. In addition, these extra moves at subsequent activations tend to explore a progressively narrowing area. This mechanism increases the exploitation of the original scheme thus helping μDE to prevent from stagnation. An experimental set-up including various test problems and dimensionality values has been considered. Numerical results show that the proposed algorithm enhances upon the original μDE performance and, despite its simplicity, is competitive with modern complex DE based algorithms.**

I. Introduction

Population-size a crucially important, if not the most important, parameter in algorithms that process multiple solutions, such as Evolutionary and Swarm Intelligence Algorithms (EAs and SIAs, respectively), see e.g. [1]. The tuning of this parameter is hard since, the success of a given problem can heavily depend on it. Looking at this issue from a complementary perspective, a robust algorithmic design might have a variable population size. This variation can be deterministic as in [2] or self-adaptive as in [3], [4], and [5]. The topic whether a large, a small, or unitary population is preferable is a topic under discussion in computational intelligence community.

In this regard, an extensive study on population-based algorithms and their advantages over single solution algorithms has been reported in [6]. Five distinct mechanisms that would justify the superiority of population-based algorithms over schemes that perturb a single solution have been identified and studied. The first mechanism is that a population offers a diversified pool of building blocks whose combination might generate new promising solutions. The second mechanism is the result of focusing of the search caused by recombination operators. Since most recombination operators have the property that, if both parents share the same value of a variable, then the offspring also has the same value in correspondence of that variable, see [7], recombination has the power of exploring the part of the search space where individuals disagree. In contrast, mutation explores the entire search space. According to this analysis this mechanism of focusing of the search by crossover can dramatically enhance the speed of the algorithm to detect a good solution. The third mechanism is the capability of a population to act as a low-pass filter of the landscape, ignoring short-length scale features in the landscape (e.g. shallow basins of attractions). The fourth mechanism is the possibility to search different areas of the decision space. This mechanism can be seen also in a different way: since population-based algorithms naturally perform an initial multiple sampling, the chance that an unlucky initial sample jeopardizes the entire algorithmic functioning is significantly mitigated. The fifth mechanism is the opportunity of using the population to learn about good parameters of the algorithm, i.e. to find a proper balance between exploration and exploitation.

For the above-listed reasons, the employment of a population-based algorithm would, in principle, be preferable when possible. However, in counter-tendency with the analysis in [6], some algorithms recently proposed in literature, although based on a single solution, still display an excellent performance, even when compared with that of modern complex population-based algorithms, see e.g. [8].

Contradictory results in literature are not only about the advisability of using or not a population within an optimization framework, but also about the proper sizing of the population. Some studies clearly suggest the usage of large populations in order to ensure the success of the algorithm, see [9]. On the other hand, in [10] and [11], it is shown that, if properly designed, a population-based algorithm with a very small population size can efficiently solve large scale problems, see also [12] and [13].

The latter kind of algorithms, i.e. population-based algorithm that use a small population, are indicated as micro algorithms and indicated by the prefix μ. An early implementation

978-1-4673-5873-6/13/$31.00

of micro algorithm is the micro Genetic Algorithm (μGA), see e.g. [14] and [15]. Over the latest years, micro algorithms have been employed in various engineering applications as they are proven to be lighter in terms of hardware requirements and thus are prone to their use in embedded systems, see [16]. In addition, algorithms that make use of small populations are more exploitative than the large ones and thus quickly achieve improvements during the early stages of the optimization process. This feature makes micro-algorithms especially useful in real-time applications when a quick answer is needed, see e.g. [17]. The effect of small populations is obviously different when applied to various search strategies, see [18].

Amongst the various micro algorithms proposed in literature, micro-Differential Evolution (μDE) is a successfully applied scheme. For example, in [19] a μDE employing opposition-based mechanism has been proposed for image thresholding problems. In [20] a μDE apprach is proposed for evolving an indirect representation of the Bin Packing Problem.

This paper proposes a novel implementation of μDE, namely micro-Differential Evolution with Axis-moves (μDEA). The proposed μDEA is a DE/rand/1/exp scheme, see [21], that employs a very small population. In addition, μDEA makes use of an extra refinement operator that perturbs the solution of the micro-population characterized by the highest performance. This refinement operator attempts to improve upon the solution by means of an exploratory move in the direction of each variable.

The remainder of this paper is organized in the following way. Section II describes the working principles of the proposed μDEA. Section III displays the experimental results of this study. Section IV gives the conclusions of this work.

II. MICRO-DIFFERENTIAL EVOLUTION WITH AXIS MOVES

Without a loss of generality, in order to clarify the notation in this paper, we refer to the minimization problem of an objective function $f(x)$, where the candidate solution x is a vector of n design variables (or genes) in a decision space D. The i^{th} design variable of the vector x is indicated as $x[i]$. The proposed μDEA algorithm consists of a DE framework and the extra moves along the axes. Section II-A and II-B describe framework and extra moves, respectively. Section II-C analyzes the μDE behavior and gives a justification to the proposed algorithmic structure.

A. Micro-Differential Evolution framework

At the beginning of the optimization process, a sampling of S_{pop} individuals is performed randomly with a uniform distribution function within the decision space D. In our implementation, the μDE population size S_{pop} has been set equal to 5.

At each generation, for each individual x_j of the S_{pop}, three individuals x_r, x_s and x_t are randomly extracted from the population. According to the DE logic, a provisional offspring x'_{off} is generated by mutation:

$$x'_{off} = x_t + F(x_r - x_s) \quad (1)$$

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

Fig. 1. Pseudo code of the exponential crossover

where $F \in [0, 1+\epsilon[$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_j the offspring should be generated. With $F \in [0, 1+\epsilon[$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1 (i.e. ϵ is a small positive value), see [22]. While there is no theoretical upper limit for F, effective values are rarely greater than 1.0. The mutation scheme given in Equation (1) is also known as DE/rand/1. In literature many other mutation variants have been proposed, see [21] and [23].

When the provisional offspring has been generated by mutation, a popular crossover, namely exponential crossover is applied to the parent solution x_j and the provisional offspring x'_{off}, see [22]. In this crossover scheme, the number of variables x_j that are exchanged during one crossover follows a geometric distribution. A geometric distribution is the discrete counterpart of the exponential distribution (that gives the name to this operator).

In the exponential crossover, a design variable of the provisional offspring $x'_{off}\,(j)$ is randomly selected and copied into the j^{th} design variable of the solution x_i. This guarantees that parent and offspring have different genotypes. Subsequently, a set of random numbers between 0 and 1 are generated. As long as $\mathrm{rand}\,(0,1) \leq CR$, 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 parent x_i. The first time that $\mathrm{rand}\,(0,1) > CR$ the copy process is interrupted. Thus, all the remaining design variables of the offspring are copied from the parent. When this crossover is combined with the DE/rand/1 mutation, the algorithm is referred to as DE/rand/1/exp (in our case μDE/rand/1/exp). For the sake of clarity the pseudo-code of the exponential crossover is shown in Fig. 1.

As shown in [24], it can easily be observed that for a given value of Cr, the meaning of the exponential crossover would change with the dimensionality of the problem. For low dimensionality problems the trial solution would inherit most of the genes from the elite while for high dimensionality problems, only a small portion of x_e would be copied into x_t. In order to avoid this problem and make the crossover action independent on the dimensionality of the problem, the

following quantity is fixed:

$$\alpha_e \approx \frac{n_e}{n} \tag{2}$$

where n_e is the number of genes we expect to copy from parent to offspring in addition to that gene deterministically copied. The probability that n_e genes are copied is $Cr^{n_e} = Cr^{n\alpha_e}$. In order to control the approximate amount of copied genes and to achieve that about n_e genes are copied into the offspring with probability 0.5, we imposed that

$$Cr^{n\alpha_e} = 0.5. \tag{3}$$

It can easily be seen that, for a chosen α_e, the crossover rate can be set on the basis of the dimensionality as follows:

$$Cr = \frac{1}{\sqrt[n\alpha_e]{2}}. \tag{4}$$

By means of formula (4), the expected quantity of information to be transmitted from parent to offspring is controlled.

B. Extra moves along the axes

Let us indicate with x_p the *pivot* individual, i.e. the individual of the micro-population that displays the best performance. With a given probability η, the pivot individual undergoes the following operator that perturbs a single solution along its n axes, i.e. separately perturbs each design variable. This operator can be seen as a modification of a classical hill-descend algorithm and employs the perturbation logic proposed in [25].

The implementation of this operator requires an additional solution, which will here be referred to as x_s. The pivot individual x_p is perturbed by computing, for each variable i:

$$x_s[i] = x_p[i] - \rho, \tag{5}$$

where ρ is the exploratory radius. Subsequently, if x_s outperforms x_p, its values (the values of the vector elements) are saved and the pivot solution is updated), otherwise a half step in the opposite direction is taken:

$$x_s[i] = x_p[i] + \frac{\rho}{2}. \tag{6}$$

Again, x_s replaces x_p if it outperforms it. If there is no update, i.e. the exploration was unsuccessful, the radius ρ is halved. This operation is repeated a limited prefixed amount of times $Iter$, thus working as a shallow local search. The current value of ρ is saved and used as the initial radius for the subsequent activation of this operator.

The complete pseudo-code of the proposed μDEA algorithm is shown in Fig. 2.

C. Algorithmic functioning

The DE algorithm is a very versatile and efficient optimizer for continuous optimization problem. However, the original scheme has a wide margin of improvement. For this reason, part of the computer science community put an energetic effort in order to propose DE variants that can outperform the original DE scheme over various optimization problems. Some of these variants turned out to be very successful.

```
generate randomly S_pop individuals of the initial population
and compute their fitness values
while the computational budget is smaller than the prefixed
amount do
  for j = 1 : S_pop do
    select three individuals x_r, x_s, and x_t
    compute mutant individual x_off = x_t + F(x_r − x_s)
    compute exponential crossover in Fig. 1
  end for
  for j = 1 : S_pop do
    compute f(x_j)
  end for
  if rand(0,1) < η then
    extract the pivot individual x_p from the micro-
    population
    x_s = x_p
    for k = 1 : Iter do
      for i = 1 : n do
        compute x_s[i] = x_p[i] − ρ
        if f(x_s) ≤ f(x_p) then
          x_p = x_s
        else
          compute x_s[i] = x_p[i] + ρ/2
          if f(x_s) ≤ f(x_p) then
            x_p = x_s
          end if
        end if
      end for
    end for
  end if
end while
```

Fig. 2. Pseudo code of the μDEA algorithm

For example, the so called jDE [26] proposed a controlled randomization of the DE parameters. Another popular DE variant based on controlled randomization of the parameters has been proposed in [27]. The Self-Adaptive Differential Evolution (SADE) proposed in [28] employs multiple mutation strategies and a randomized coordination scheme based on an initial learning. Another efficient coordination strategy for a multiple mutation structure has been proposed in [29]. A modified selection strategy, based on the location within the population, for the individuals undergoing mutation has been proposed in [30]. Another example of efficient DE variant has been proposed in [31] where a novel DE mutation is combined with a randomized fitness based selection of the individuals undergoing mutation.

As highlighted in [23] and [32], the reasons behind the wide margin of improvements for the original DE scheme are mainly two. The first reason is that DE scheme has a limited amount of search moves. Thus, DE variants that include extra moves into the original framework, usually, lead to improved versions. The extra moves can be explicitly implemented within the DE framework, see e.g. [33] and [34], or can be implicitly contained in other perturbation mechanism. The randomization, as shown in [32] and [35], plays a very important role as it allows the generation of candidate solutions that would not be generated by standard mutation and crossover operations. The second reason is that

DE can be excessively exploratory. As shown in [36], a typical challenge in DE functioning is that the solutions in a population can be diverse and still unable to outperform the individual with the best performance (i.e. DE can easily suffers from stagnation). In order to prevent from this condition, the employment of exploitative component or the implementation of exploitative actions can be beneficial to DE performance.

The μDE schemes, due to the fact that use a small population-size, are intrinsically more exploitative than standard DE schemes and this would, in principle, make them less prone to stagnation issues. On the other hand, small populations could potentially lead to an excessively quick diversity loss and thus to a premature convergence. The undesired premature convergence effect would actually have a major impact on the performance on a micro-Evolutionary Algorithm (μEA), such as a μGA. Unlike μEAs, the DE search logic does not appear to lead too often to a diversity loss. In low dimensions and for simple fitness landscapes, a DE with a small population would obviously lose the diversity and converge to a solution. On the other hand, in complex multi-modal and multi-dimensional problems (already in 30 dimensions), even though only a few solutions (e.g. 5) compose the population of a DE scheme, the μDE population tends to keep the diversity high and its solutions could still be distant within the decision space D. Since the distance among solutions in a μDE scheme is correlated to the position of the potential offspring, after initial improvements, a μDE can be too exploratory and generate new points far away from the interesting areas. On the contrary, in order to continue achieving fitness improvements, the algorithm may require to enhance the exploitation and focus the search in the areas of interest.

The proposed μDEA aims at compensating this effect by including within the search moves an alternative exploration rule for the neighbourhood of the best solution. The extra moves along the axes are supposed to offer, in a simplistic way, a support to the μDE framework. These moves offer an alternative search logic with respect to the normal DE mutation and crossover and, most importantly, performs thorough exploration of the most interesting areas so far detected, i.e. the areas surrounding the solution characterized by the best performance. As a result, μDEA explicitly incorporates extra moves within a μDE framework and increases the exploitation of the original scheme. Finally, the fact that the exploratory radius of the moves along the axes is not re-initialized (but used for the subsequent activation) results in a natural increase in the exploitation action of this operator. In this way, the moves along the axes explore a progressively narrowing area around the pivot solution x_p.

III. Numerical Results

All the test problems included in the following four testbeds have been considered in this study.

- The CEC2005 benchmark described in [37] in 30 dimensions (25 test problems)
- The BBOB2010 benchmark described in [38] in 100 dimensions (24 test problems)
- The CEC2008 benchmark described in [39] in 1000 dimensions (7 test problems)
- The CEC2010 benchmark described in [40] in 1000 dimensions (20 test problems)

Thus, 76 test problems have been considered in this study. For each algorithm in this paper (see following subsections) 100 runs have been performed. Each run has been continued for $5000 \times n$ fitness evaluations, where n is the dimensionality of the problem. For each test problem and each algorithm, the average final fitness value $\pm$ standard deviation over the 100 available runs has been computed. In order to strengthen the statistical significance of the results, for each test problem the Wilcoxon Rank-Sum test [41] has been also applied, with a confidence level of 0.95.

The proposed μDEA has been run with $S_{pop} = 5$, $F = 0.7$, $\alpha_e = 0.5$, see eq. (2), $Iter = 20$, $\eta = 0.25$, and $\rho = 0.4$ of the width of the decision space D.

The following algorithms with respective parameter setting have been considered for comparison against μDEA.

- A μDE with the same parameter setting of μDEA
- Self-Adaptive Differential Evolution (SADE) proposed in [28] with population size equal to 50 individuals.
- Adaptive Differential Evolution (JADE) proposed in [27] with population size equal to 60 individuals, group size factor $p = 0.05$ and parameters adaptation rate factor $c = 0.1$.
- Modified Differential Evolution with p-Best Crossover (MDE-pBX) proposed in [31] with population size equal to 100 individuals and group size q equal to 15% of the population size.

Tables I, II, III, and IV show the comparison against μDE for the four benchmarks under consideration. Tables V, VI, VII, and VIII, show the comparison against SADE, JADE, and MDE-pBX. The tables in this study display the average final fitness value over the 100 available runs and the corresponding standard deviation value. The results of the Wilcoxon test are also reported in terms of pair-wise comparisons. The symbols"=" and "+" ("-") indicate, respectively, a statistically equivalent performance and a better (worse) performance of RIS compared with the algorithm in the column label.

The numerical comparison between μDEA and μDE shows that the extra moves along the axes tend to have a positive effect on the algorithmic performance in the majority of the considered cases. This fact confirms the validity of the analysis reported in [23] about the lack of moves in DE frameworks and that extra moves appear to be beneficial for DE. In addition, as shown Tables III and IV, the success of μDEA with respect to μDE in high dimensions demonstrates that an increase in the exploitation is beneficial also in DE schemes that employ a micro-population. In our opinion, this fact can be interpreted by considering that even in the case micro-populations, DE solutions tend to be scattered in the decision space, thus using a large exploration step whilst a neighbourhood search would

TABLE I
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =μDEA) FOR μDEA AGAINST μDE ON CEC2005[37] IN 30 DIMENSIONS.

	μDEA	μDE	
f_1	$\mathbf{-4.50e+02 \pm 2.18e-13}$	$-3.98e+02 \pm 5.14e+02$	+
f_2	$\mathbf{-4.50e+02 \pm 2.43e-12}$	$-4.29e+02 \pm 1.28e+02$	+
f_3	$\mathbf{1.83e+05 \pm 1.05e+05}$	$1.66e+07 \pm 5.61e+06$	+
f_4	$6.74e+04 \pm 1.60e+04$	$\mathbf{7.59e+02 \pm 1.22e+03}$	-
f_5	$\mathbf{7.20e+03 \pm 2.22e+03}$	$9.49e+03 \pm 2.07e+03$	+
f_6	$\mathbf{8.52e+02 \pm 1.03e+03}$	$2.79e+07 \pm 1.94e+08$	=
f_7	$\mathbf{-1.80e+02 \pm 1.35e-02}$	$2.99e+11 \pm 1.20e+12$	+
f_8	$\mathbf{-1.20e+02 \pm 4.75e-03}$	$-1.19e+02 \pm 5.98e-02$	+
f_9	$\mathbf{-1.17e+02 \pm 1.16e+00}$	$-1.16e+02 \pm 1.78e+00$	=
f_{10}	$2.62e+02 \pm 2.05e+01$	$\mathbf{2.56e+02 \pm 1.89e+01}$	-
f_{11}	$\mathbf{1.18e+02 \pm 3.55e+00}$	$1.21e+02 \pm 2.50e+00$	+
f_{12}	$\mathbf{1.49e+03 \pm 2.90e+03}$	$1.55e+04 \pm 6.55e+03$	+
f_{13}	$-1.22e+02 \pm 1.45e+00$	$\mathbf{-1.27e+02 \pm 1.20e+00}$	-
f_{14}	$-2.86e+02 \pm 2.78e-01$	$\mathbf{-2.87e+02 \pm 2.60e-01}$	-
f_{15}	$1.45e+03 \pm 2.89e+00$	$\mathbf{1.45e+03 \pm 4.25e+00}$	-
f_{16}	$1.59e+03 \pm 1.56e+01$	$\mathbf{1.58e+03 \pm 1.20e+01}$	=
f_{17}	$1.74e+03 \pm 1.81e+01$	$\mathbf{1.61e+03 \pm 1.13e+01}$	-
f_{18}	$\mathbf{9.10e+02 \pm 5.26e-12}$	$9.10e+02 \pm 5.41e-02$	+
f_{19}	$\mathbf{9.10e+02 \pm 5.82e-12}$	$9.10e+02 \pm 1.64e-01$	+
f_{20}	$\mathbf{9.10e+02 \pm 5.61e-12}$	$9.10e+02 \pm 4.18e-01$	+
f_{21}	$\mathbf{1.72e+03 \pm 1.09e+01}$	$1.72e+03 \pm 8.91e+00$	+
f_{22}	$2.60e+03 \pm 5.80e+01$	$\mathbf{2.54e+03 \pm 4.93e+01}$	-
f_{23}	$1.73e+03 \pm 9.39e+00$	$\mathbf{1.72e+03 \pm 8.43e+00}$	-
f_{24}	$1.71e+03 \pm 1.44e+01$	$\mathbf{1.71e+03 \pm 9.66e+00}$	=
f_{25}	$\mathbf{1.88e+03 \pm 3.40e+02}$	$1.91e+03 \pm 1.37e+02$	=

TABLE II
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE = μDEA) FOR μDEA AGAINST μDE ON BBOB2010[38] IN 100 DIMENSIONS.

	μDEA	μDE	
f_1	$\mathbf{7.95e+01 \pm 3.23e-03}$	$7.95e+01 \pm 2.08e-01$	+
f_2	$\mathbf{-1.50e+02 \pm 1.73e+02}$	$1.67e+03 \pm 1.13e+04$	+
f_3	$-2.20e+02 \pm 1.05e+02$	$\mathbf{-4.09e+02 \pm 2.79e+01}$	-
f_4	$-1.16e+02 \pm 1.12e+02$	$\mathbf{-3.81e+02 \pm 3.26e+01}$	-
f_5	$\mathbf{-8.26e+00 \pm 2.06e+00}$	$-6.49e+00 \pm 4.89e+00$	+
f_6	$\mathbf{8.73e+01 \pm 1.34e+02}$	$4.32e+02 \pm 1.03e+02$	+
f_7	$\mathbf{4.42e+02 \pm 1.71e+02}$	$6.35e+02 \pm 8.28e+01$	+
f_8	$\mathbf{2.74e+02 \pm 9.87e+01}$	$2.99e+02 \pm 9.18e+01$	+
f_9	$\mathbf{1.92e+02 \pm 5.80e+01}$	$2.26e+02 \pm 2.78e+01$	+
f_{10}	$\mathbf{5.65e+04 \pm 9.88e+04}$	$2.07e+05 \pm 2.87e+04$	+
f_{11}	$8.30e+02 \pm 1.30e+02$	$\mathbf{6.43e+02 \pm 6.68e+01}$	-
f_{12}	$\mathbf{6.52e+02 \pm 3.39e+03}$	$7.29e+04 \pm 3.11e+05$	+
f_{13}	$\mathbf{4.02e+01 \pm 1.04e+01}$	$6.89e+01 \pm 9.00e+01$	=
f_{14}	$\mathbf{-5.23e+01 \pm 1.73e-02}$	$-5.23e+01 \pm 2.41e-01$	+
f_{15}	$\mathbf{2.36e+03 \pm 3.43e+02}$	$2.87e+03 \pm 2.09e+02$	+
f_{16}	$\mathbf{9.04e+01 \pm 6.19e+00}$	$9.80e+01 \pm 3.31e+00$	+
f_{17}	$\mathbf{-7.56e+00 \pm 2.73e+00}$	$-3.41e+00 \pm 2.08e+00$	+
f_{18}	$\mathbf{1.75e+01 \pm 8.38e+00}$	$3.62e+01 \pm 7.70e+00$	+
f_{19}	$\mathbf{-9.29e+01 \pm 2.50e+00}$	$-9.08e+01 \pm 8.39e-01$	+
f_{20}	$-5.45e+02 \pm 2.07e-01$	$\mathbf{-5.46e+02 \pm 6.08e-02}$	-
f_{21}	$4.98e+01 \pm 6.24e+00$	$\mathbf{4.32e+01 \pm 2.40e+00}$	-
f_{22}	$-9.87e+02 \pm 9.58e+00$	$\mathbf{-9.95e+02 \pm 6.34e+00}$	-
f_{23}	$\mathbf{8.60e+00 \pm 8.04e-01}$	$9.85e+00 \pm 3.78e-01$	+
f_{24}	$\mathbf{1.71e+03 \pm 3.62e+02}$	$2.10e+03 \pm 1.88e+02$	+

be more beneficial. The extra moves along the axes, explore progressively narrowing neighbourhood and support the basic DE search moves to detect solutions characterized by a high quality.

Numerical results in 30 dimensions show that the proposed μDEA is, in general, slightly less promising that some of the other three modern DE versions but still is capable to display a respectable performance. More specifically, μDE out performs each of the other three algorithms in slightly less than half of the cases. In 100 dimensions, μDE is still slightly outperformed by SADE and MDE-pBX while is definitely competitive with JADE. The most interesting results of this study are reported in the large scale cases. In 1000 dimensions, μDE displays a surprisingly good performance with respect to the other modern DE based algorithms considered in this study. In high dimensions, μDEA displays the best performance (see Table VIII) by slightly outperforming SADE and clearly outperforming JADE and MDE-pBX. This result is especially interesting if we take into account that μDEA is a very simple and light (in terms of memory requirement and computational overhead) algorithm. It is important to remark that this study shows that small DE populations are more adequate than large ones to tackle large scale problems. Fig 3 shows the average performance in a case of successful application of the μDEA scheme.

TABLE III
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE = μDEA) FOR μDEA AGAINST μDE ON CEC2008[39] IN 1000 DIMENSIONS.

	μDEA	μDE	
f_1	$\mathbf{-4.50e+02 \pm 1.42e-09}$	$5.49e+02 \pm 2.25e+03$	+
f_2	$\mathbf{-4.50e+02 \pm 2.53e-02}$	$-3.59e+02 \pm 1.33e+01$	+
f_3	$\mathbf{1.52e+03 \pm 8.92e+01}$	$2.83e+08 \pm 1.52e+09$	+
f_4	$5.77e+03 \pm 4.56e+02$	$\mathbf{2.16e+02 \pm 5.30e+01}$	-
f_5	$\mathbf{-1.80e+02 \pm 1.89e-03}$	$-1.70e+02 \pm 2.25e+01$	+
f_6	$\mathbf{-1.40e+02 \pm 5.01e-07}$	$-1.37e+02 \pm 7.81e-01$	+
f_7	$-1.35e+04 \pm 6.61e+01$	$\mathbf{-1.44e+04 \pm 2.61e+01}$	-

TABLE IV
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE = μDEA) FOR μDEA AGAINST μDE ON CEC2010[40] IN 1000 DIMENSIONS.

	μDEA	μDE	
f_1	$\mathbf{4.44e-18 \pm 7.20e-19}$	$4.45e+07 \pm 1.87e+08$	+
f_2	$5.71e+03 \pm 3.66e+02$	$\mathbf{5.26e+02 \pm 4.45e+01}$	-
f_3	$\mathbf{2.47e-02 \pm 9.79e-02}$	$2.88e+00 \pm 8.15e-01$	+
f_4	$\mathbf{2.07e+13 \pm 3.99e+12}$	$3.12e+13 \pm 7.40e+12$	+
f_5	$\mathbf{4.49e+08 \pm 1.16e+08}$	$6.32e+08 \pm 8.94e+07$	+
f_6	$\mathbf{1.90e+07 \pm 3.01e+06}$	$2.04e+07 \pm 2.15e+05$	+
f_7	$1.92e+10 \pm 4.99e+09$	$\mathbf{1.83e+10 \pm 3.99e+09}$	=
f_8	$\mathbf{2.36e+10 \pm 1.22e+10}$	$2.70e+11 \pm 1.71e+12$	+
f_9	$\mathbf{1.71e+08 \pm 7.24e+06}$	$4.55e+08 \pm 2.53e+08$	+
f_{10}	$7.23e+03 \pm 2.88e+02$	$\mathbf{6.95e+03 \pm 2.85e+02}$	-
f_{11}	$\mathbf{1.48e+02 \pm 4.56e+01}$	$2.08e+02 \pm 2.18e+00$	+
f_{12}	$\mathbf{8.39e+04 \pm 1.48e+05}$	$3.79e+05 \pm 2.10e+04$	+
f_{13}	$\mathbf{2.26e+05 \pm 9.13e+04}$	$9.57e+07 \pm 4.98e+08$	=
f_{14}	$\mathbf{1.33e+08 \pm 2.53e+08}$	$9.38e+08 \pm 4.51e+07$	+
f_{15}	$\mathbf{7.31e+03 \pm 3.08e+02}$	$1.37e+04 \pm 3.75e+02$	+
f_{16}	$\mathbf{2.23e+02 \pm 1.08e+02}$	$4.11e+02 \pm 2.25e+00$	+
f_{17}	$\mathbf{1.14e+05 \pm 2.39e+05}$	$8.07e+05 \pm 2.43e+04$	+
f_{18}	$\mathbf{3.57e+04 \pm 1.21e+04}$	$3.71e+08 \pm 2.08e+09$	+
f_{19}	$5.47e+05 \pm 3.43e+04$	$\mathbf{2.82e+05 \pm 2.68e+04}$	-
f_{20}	$\mathbf{1.49e+04 \pm 1.10e+03}$	$1.99e+08 \pm 7.66e+08$	+

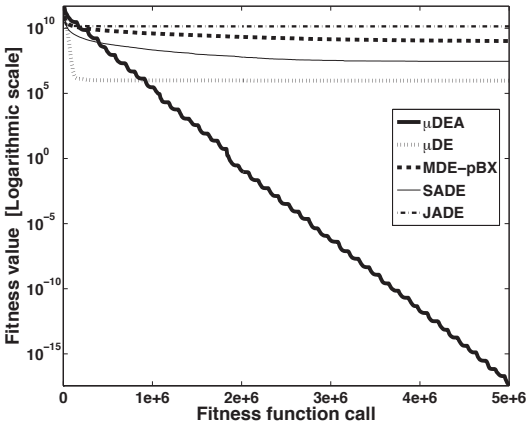


Fig. 3. Performance trend for f_1 of CEC2010 [40] in 1000 dimensions.

TABLE V
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =μDEA) FOR μDEA AGAINST SADE, JADE AND MDE-PBX ON CEC2005[37] IN 30 DIMENSIONS.

	μDEA	MDE-pBX		SADE		JADE	
f_1	$-4.50e+02 \pm 2.18e-13$	$-4.50e+02 \pm 1.62e-13$	=	$\mathbf{-4.50e+02 \pm 2.90e-14}$	=	$-3.51e+02 \pm 1.99e+02$	+
f_2	$\mathbf{-4.50e+02 \pm 2.43e-12}$	$-4.50e+02 \pm 2.54e-03$	+	$-4.39e+02 \pm 2.31e+01$	+	$3.08e+02 \pm 7.32e+02$	+
f_3	$\mathbf{1.83e+05 \pm 1.05e+05}$	$2.81e+05 \pm 1.99e+05$	+	$8.97e+05 \pm 4.27e+05$	+	$3.71e+06 \pm 1.77e+06$	+
f_4	$6.74e+04 \pm 1.60e+04$	$\mathbf{-1.29e+02 \pm 9.67e+02}$	-	$-8.60e+01 \pm 5.87e+02$	-	$2.27e+03 \pm 1.68e+03$	-
f_5	$7.20e+03 \pm 2.22e+03$	$\mathbf{2.74e+03 \pm 6.34e+02}$	-	$2.86e+03 \pm 5.36e+02$	-	$3.91e+03 \pm 9.18e+02$	-
f_6	$8.52e+02 \pm 1.03e+03$	$4.33e+02 \pm 4.81e+01$	-	$\mathbf{4.16e+02 \pm 3.62e+01}$	-	$5.07e+06 \pm 1.53e+07$	+
f_7	$\mathbf{-1.80e+02 \pm 1.35e-02}$	$2.03e+06 \pm 1.88e+07$	+	$4.39e+02 \pm 6.16e+03$	+	$1.69e+13 \pm 1.78e+13$	+
f_8	$\mathbf{-1.20e+02 \pm 4.75e-03}$	$-1.19e+02 \pm 4.23e-01$	+	$-1.19e+02 \pm 4.24e-01$	+	$-1.19e+02 \pm 5.75e-02$	+
f_9	$-1.17e+02 \pm 1.16e+00$	$-1.17e+02 \pm 1.14e+00$	-	$\mathbf{-1.17e+02 \pm 4.60e-01}$	-	$-1.17e+02 \pm 1.22e+00$	=
f_{10}	$2.62e+02 \pm 2.05e+01$	$2.23e+02 \pm 2.44e+01$	-	$2.27e+02 \pm 2.25e+01$	-	$\mathbf{2.02e+02 \pm 2.20e+01}$	-
f_{11}	$1.18e+02 \pm 3.55e+00$	$\mathbf{1.11e+02 \pm 4.59e+00}$	-	$1.16e+02 \pm 3.54e+00$	-	$1.16e+02 \pm 4.48e+00$	-
f_{12}	$\mathbf{1.49e+03 \pm 2.90e+03}$	$3.77e+03 \pm 3.87e+03$	+	$4.90e+03 \pm 5.23e+03$	+	$1.72e+04 \pm 1.54e+04$	+
f_{13}	$-1.22e+02 \pm 1.45e+00$	$-1.19e+02 \pm 2.28e+00$	+	$-1.24e+02 \pm 9.45e-01$	-	$\mathbf{-1.26e+02 \pm 9.64e-01}$	-
f_{14}	$-2.86e+02 \pm 2.78e-01$	$\mathbf{-2.87e+02 \pm 4.50e-01}$	-	$-2.87e+02 \pm 4.22e-01$	-	$-2.87e+02 \pm 2.02e-01$	-
f_{15}	$1.45e+03 \pm 2.89e+00$	$1.46e+03 \pm 6.43e+00$	+	$\mathbf{1.44e+03 \pm 1.67e+00}$	-	$1.45e+03 \pm 3.45e+00$	=
f_{16}	$1.59e+03 \pm 1.56e+01$	$1.58e+03 \pm 1.11e+01$	-	$1.56e+03 \pm 8.59e+00$	-	$\mathbf{1.56e+03 \pm 6.50e+00}$	-
f_{17}	$1.74e+03 \pm 1.81e+01$	$1.62e+03 \pm 9.04e+00$	-	$1.62e+03 \pm 9.84e+00$	-	$\mathbf{1.59e+03 \pm 7.53e+00}$	-
f_{18}	$\mathbf{9.10e+02 \pm 5.26e-12}$	$9.10e+02 \pm 8.31e-11$	+	$9.10e+02 \pm 4.66e-09$	+	$9.10e+02 \pm 2.62e-01$	+
f_{19}	$\mathbf{9.10e+02 \pm 5.82e-12}$	$9.10e+02 \pm 2.42e-10$	+	$9.10e+02 \pm 5.64e-09$	+	$9.10e+02 \pm 1.31e-01$	+
f_{20}	$\mathbf{9.10e+02 \pm 5.61e-12}$	$9.10e+02 \pm 3.41e-11$	+	$9.10e+02 \pm 1.36e-10$	+	$9.10e+02 \pm 1.40e-01$	+
f_{21}	$1.72e+03 \pm 1.09e+01$	$1.70e+03 \pm 5.51e+00$	-	$1.70e+03 \pm 7.05e+00$	-	$\mathbf{1.69e+03 \pm 4.32e+00}$	-
f_{22}	$2.60e+03 \pm 5.80e+01$	$2.41e+03 \pm 4.97e+01$	-	$2.34e+03 \pm 3.99e+01$	-	$\mathbf{2.29e+03 \pm 3.44e+01}$	-
f_{23}	$1.73e+03 \pm 9.39e+00$	$1.70e+03 \pm 5.28e+00$	-	$1.71e+03 \pm 6.04e+00$	-	$\mathbf{1.70e+03 \pm 4.48e+00}$	-
f_{24}	$1.71e+03 \pm 1.44e+01$	$1.67e+03 \pm 1.55e+01$	-	$1.67e+03 \pm 1.21e+01$	-	$\mathbf{1.66e+03 \pm 1.40e+01}$	-
f_{25}	$1.88e+03 \pm 3.40e+02$	$1.83e+03 \pm 1.55e+02$	=	$\mathbf{1.78e+03 \pm 2.05e+02}$	-	$1.86e+03 \pm 4.65e+01$	=

TABLE VI
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =μDEA) FOR μDEA AGAINST SADE, JADE AND MDE-PBX ON BBOB2010[38] IN 100 DIMENSIONS.

	μDEA	MDE-pBX		SADE		JADE	
f_1	$7.95e+01 \pm 3.23e-03$	$7.95e+01 \pm 7.60e-05$	=	$\mathbf{7.95e+01 \pm 9.67e-13}$	=	$8.77e+01 \pm 7.64e+00$	+
f_2	$-1.50e+02 \pm 1.73e+02$	$-2.10e+02 \pm 6.06e-03$	-	$\mathbf{-2.10e+02 \pm 9.83e-13}$	-	$5.73e+04 \pm 8.69e+04$	+
f_3	$-2.20e+02 \pm 1.05e+02$	$3.29e+01 \pm 7.94e+01$	+	$-2.94e+02 \pm 4.40e+01$	-	$\mathbf{-3.11e+02 \pm 4.95e+01}$	-
f_4	$-1.16e+02 \pm 1.12e+02$	$4.03e+02 \pm 1.31e+02$	+	$\mathbf{-1.61e+02 \pm 1.19e+02}$	-	$-1.43e+02 \pm 9.83e+01$	-
f_5	$-8.26e+00 \pm 2.06e+00$	$-3.09e-02 \pm 1.27e+01$	+	$\mathbf{-9.16e+00 \pm 4.59e-01}$	-	$1.24e+02 \pm 5.08e+01$	+
f_6	$8.73e+01 \pm 1.34e+02$	$\mathbf{8.03e+01 \pm 3.32e+01}$	-	$1.11e+02 \pm 3.87e+01$	+	$3.92e+02 \pm 1.18e+02$	+
f_7	$4.42e+02 \pm 1.71e+02$	$3.70e+02 \pm 7.43e+01$	-	$3.39e+02 \pm 6.69e+01$	-	$\mathbf{3.08e+02 \pm 6.50e+01}$	-
f_8	$\mathbf{2.74e+02 \pm 9.87e+01}$	$3.40e+02 \pm 6.77e+01$	+	$2.82e+02 \pm 6.22e+01$	+	$7.24e+03 \pm 6.15e+03$	+
f_9	$\mathbf{1.92e+02 \pm 5.80e+01}$	$2.52e+02 \pm 3.75e+01$	+	$2.28e+02 \pm 2.45e+01$	+	$1.78e+03 \pm 1.33e+03$	+
f_{10}	$5.65e+04 \pm 9.88e+04$	$\mathbf{1.64e+04 \pm 7.99e+03}$	-	$5.22e+04 \pm 2.07e+04$	-	$1.94e+05 \pm 8.87e+04$	+
f_{11}	$8.30e+02 \pm 1.30e+02$	$\mathbf{9.16e+01 \pm 7.45e+00}$	-	$1.83e+02 \pm 2.72e+01$	-	$2.11e+02 \pm 2.76e+01$	-
f_{12}	$6.52e+02 \pm 3.39e+03$	$-5.99e+02 \pm 7.07e+01$	-	$\mathbf{-6.14e+02 \pm 7.07e+00}$	=	$2.22e+07 \pm 2.13e+07$	+
f_{13}	$4.02e+01 \pm 1.04e+01$	$3.47e+01 \pm 6.70e+00$	-	$\mathbf{3.20e+01 \pm 2.81e+00}$	-	$7.69e+02 \pm 2.46e+02$	+
f_{14}	$-5.23e+01 \pm 1.73e-02$	$-5.23e+01 \pm 2.55e-03$	=	$\mathbf{-5.23e+01 \pm 1.86e-03}$	=	$-4.65e+01 \pm 3.89e+00$	+
f_{15}	$2.36e+03 \pm 3.43e+02$	$1.66e+03 \pm 1.10e+02$	-	$\mathbf{1.35e+03 \pm 6.05e+01}$	-	$1.59e+03 \pm 8.67e+01$	-
f_{16}	$9.04e+01 \pm 6.19e+00$	$\mathbf{8.85e+01 \pm 4.46e+00}$	=	$9.72e+01 \pm 4.30e+00$	+	$1.01e+02 \pm 3.46e+00$	+
f_{17}	$-7.56e+00 \pm 2.73e+00$	$-1.35e+01 \pm 4.83e-01$	-	$-1.38e+01 \pm 5.21e-01$	-	$\mathbf{-1.45e+01 \pm 5.96e-01}$	-
f_{18}	$1.75e+01 \pm 8.38e+00$	$-4.84e+00 \pm 1.68e+00$	-	$-5.07e+00 \pm 1.98e+00$	-	$\mathbf{-8.79e+00 \pm 2.11e+00}$	-
f_{19}	$-9.29e+01 \pm 2.50e+00$	$\mathbf{-1.00e+02 \pm 7.13e-01}$	-	$-9.98e+01 \pm 6.72e-01$	-	$-9.50e+01 \pm 2.26e-01$	-
f_{20}	$\mathbf{-5.45e+02 \pm 2.07e-01}$	$-5.44e+02 \pm 1.14e-01$	+	$-5.45e+02 \pm 1.61e-01$	+	$-5.12e+02 \pm 9.92e+01$	+
f_{21}	$4.98e+01 \pm 6.24e+00$	$\mathbf{4.49e+01 \pm 5.88e+00}$	-	$4.63e+01 \pm 5.76e+00$	-	$4.93e+01 \pm 6.25e+00$	=
f_{22}	$-9.87e+02 \pm 9.58e+00$	$-9.92e+02 \pm 9.11e+00$	-	$-9.93e+02 \pm 7.97e+00$	-	$\mathbf{-9.94e+02 \pm 6.66e+00}$	-
f_{23}	$\mathbf{8.60e+00 \pm 8.04e-01}$	$9.34e+00 \pm 7.99e-01$	+	$9.17e+00 \pm 6.78e-01$	+	$1.07e+01 \pm 3.78e-01$	+
f_{24}	$1.71e+03 \pm 3.62e+02$	$4.75e+02 \pm 4.72e+01$	-	$\mathbf{3.73e+02 \pm 3.17e+01}$	-	$1.03e+03 \pm 4.44e+01$	-

In addition to the results presented above, the ranking among all the algorithms considered in this article has been performed by means of the Holm-Bonferroni procedure, see [42] and [43], for the 5 algorithms under study and the 76 problems under consideration. The Holm-Bonferroni procedure consists of the following. Considering the results in the tables above, the 5 algorithms under analysis have been ranked on the basis of their average performance calculated over the 76 test problems. More specifically, a score R_i for $i = 1, \ldots, N_A$ (where N_A is the number of algorithms under analysis, $N_A = 5$ in our case) has been assigned. The score has been assigned in the following way: for each problem, a score of 5 is assigned to the algorithm displaying the best performance, 4 is assigned to the second best, 3 to the third and so on. The algorithm displaying the worst performance scores 1. For each algorithm, the scores obtained on each problem are summed up averaged over the amount of test problems (76 in our case). On the basis of these scores the algorithms are sorted (ranked). With the calculated R_i values, RIS has been taken as a reference algorithm. Indicating with R_0 the rank of RIS, and with R_j for $j = 1, \ldots, N_A - 1$ the rank of one of the remaining eleven algorithms, the values z_j have been calculated as

$$z_j = \frac{R_j - R_0}{\sqrt{\frac{N_A(N_A+1)}{6N_{TP}}}} \tag{7}$$

where N_{TP} is the number of test problems in consideration ($N_{TP} = 76$ in our case). By means of the z_j values, the corresponding cumulative normal distribution values p_j have been calculated. These p_j values have then been compared with the corresponding δ/j where δ is the level of confidence, set to 0.05 in our case. Table IX displays the ranks, z_j values, p_j values, and corresponding δ/j obtained in this way. The rank of μDEA is shown in parenthesis. Moreover, it is indicated whether the null-hypothesis (that the two algorithms have indistinguishable performances) is "Rejected", i.e. μDEA statistically outperforms the algorithm under consideration, or

TABLE VII
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =μDEA) FOR μDEA AGAINST SADE, JADE AND MDE-PBX ON CEC2008[39] IN 1000 DIMENSIONS.

	muDEA	MDEpBX		SADE		JADE	
f_1	$\mathbf{-4.50e+02 \pm 1.42e-09}$	$1.20e+05 \pm 4.41e+04$	+	$5.06e+03 \pm 6.00e+03$	+	$1.02e+06 \pm 3.49e+05$	+
f_2	$\mathbf{-4.50e+02 \pm 2.53e-02}$	$-3.33e+02 \pm 4.09e+00$	+	$-3.19e+02 \pm 5.11e+00$	+	$-3.20e+02 \pm 7.98e+00$	+
f_3	$\mathbf{1.52e+03 \pm 8.92e+01}$	$3.13e+10 \pm 1.65e+10$	+	$9.97e+08 \pm 1.66e+09$	+	$4.40e+11 \pm 2.18e+11$	+
f_4	$5.77e+03 \pm 4.56e+02$	$7.60e+03 \pm 2.55e+02$	+	$5.88e+03 \pm 3.95e+02$	=	$\mathbf{4.43e+03 \pm 8.64e+02}$	-
f_5	$\mathbf{-1.80e+02 \pm 1.89e-03}$	$1.08e+03 \pm 4.60e+02$	+	$-1.20e+02 \pm 6.46e+01$	+	$8.70e+03 \pm 2.95e+03$	+
f_6	$\mathbf{-1.40e+02 \pm 5.01e-07}$	$-1.21e+02 \pm 5.10e-02$	+	$-1.21e+02 \pm 1.77e-01$	+	$-1.22e+02 \pm 5.79e-01$	+
f_7	$\mathbf{-1.35e+04 \pm 6.61e+01}$	$-1.11e+04 \pm 1.63e+02$	+	$-1.11e+04 \pm 1.28e+02$	+	$-1.19e+04 \pm 4.24e+02$	+

TABLE VIII
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =μDEA) FOR μDEA AGAINST SADE, JADE AND MDE-PBX ON CEC2010[40] IN 1000 DIMENSIONS.

	μDEA	MDE-pBX		SADE		JADE	
f_1	$\mathbf{4.44e-18 \pm 7.20e-19}$	$1.05e+09 \pm 6.58e+08$	+	$2.89e+07 \pm 1.02e+08$	+	$1.40e+10 \pm 6.91e+09$	+
f_2	$5.71e+03 \pm 3.66e+02$	$7.02e+03 \pm 2.38e+02$	+	$5.55e+03 \pm 2.99e+02$	-	$\mathbf{4.56e+03 \pm 1.04e+03}$	-
f_3	$\mathbf{2.47e-02 \pm 9.79e-02}$	$1.93e+01 \pm 4.76e-02$	+	$1.89e+01 \pm 2.83e-01$	+	$1.76e+01 \pm 6.75e-01$	+
f_4	$2.07e+13 \pm 3.99e+12$	$3.21e+12 \pm 9.76e+11$	-	$\mathbf{1.95e+12 \pm 8.82e+11}$	-	$2.62e+12 \pm 1.03e+12$	-
f_5	$4.49e+08 \pm 1.16e+08$	$1.54e+08 \pm 2.77e+07$	-	$1.03e+08 \pm 1.83e+07$	-	$\mathbf{8.58e+07 \pm 1.77e+07}$	-
f_6	$1.90e+07 \pm 3.01e+06$	$3.65e+06 \pm 1.75e+06$	-	$\mathbf{9.16e+05 \pm 1.21e+06}$	-	$3.48e+06 \pm 1.40e+06$	-
f_7	$1.92e+10 \pm 4.99e+09$	$\mathbf{6.79e+06 \pm 1.01e+07}$	-	$1.01e+08 \pm 2.36e+08$	-	$3.37e+09 \pm 3.66e+09$	-
f_8	$2.36e+10 \pm 1.22e+10$	$2.03e+08 \pm 1.63e+08$	-	$\mathbf{7.08e+07 \pm 3.71e+07}$	-	$6.31e+13 \pm 1.80e+14$	+
f_9	$\mathbf{1.71e+08 \pm 7.24e+06}$	$1.68e+09 \pm 1.00e+09$	+	$2.11e+08 \pm 2.93e+08$	+	$1.67e+10 \pm 5.87e+09$	+
f_{10}	$7.23e+03 \pm 2.88e+02$	$7.33e+03 \pm 2.55e+02$	+	$\mathbf{6.22e+03 \pm 3.15e+02}$	-	$7.50e+03 \pm 1.07e+03$	+
f_{11}	$\mathbf{1.48e+02 \pm 4.56e+01}$	$2.06e+02 \pm 2.40e+00$	+	$2.05e+02 \pm 4.34e+00$	+	$1.94e+02 \pm 7.49e+00$	+
f_{12}	$\mathbf{8.39e+04 \pm 1.48e+05}$	$2.92e+05 \pm 6.60e+04$	+	$3.15e+05 \pm 1.36e+05$	+	$2.32e+06 \pm 4.55e+05$	+
f_{13}	$\mathbf{2.26e+05 \pm 9.13e+04}$	$2.88e+09 \pm 3.17e+09$	+	$5.67e+07 \pm 2.48e+08$	=	$8.02e+10 \pm 4.76e+10$	+
f_{14}	$\mathbf{1.33e+08 \pm 2.53e+08}$	$1.04e+09 \pm 1.97e+08$	+	$3.77e+08 \pm 1.13e+08$	+	$1.31e+10 \pm 4.64e+09$	+
f_{15}	$7.31e+03 \pm 3.08e+02$	$7.44e+03 \pm 2.80e+02$	+	$\mathbf{6.49e+03 \pm 2.38e+02}$	-	$8.51e+03 \pm 1.03e+03$	+
f_{16}	$\mathbf{2.23e+02 \pm 1.08e+02}$	$3.84e+02 \pm 1.22e+00$	+	$3.82e+02 \pm 2.00e+00$	+	$3.83e+02 \pm 1.19e+01$	+
f_{17}	$\mathbf{1.14e+05 \pm 2.39e+05}$	$4.35e+05 \pm 8.33e+04$	+	$6.37e+05 \pm 2.00e+05$	+	$2.63e+06 \pm 7.56e+05$	+
f_{18}	$\mathbf{3.57e+04 \pm 1.21e+04}$	$3.73e+10 \pm 1.95e+10$	+	$7.60e+08 \pm 1.14e+09$	+	$4.42e+11 \pm 1.91e+11$	+
f_{19}	$\mathbf{5.47e+05 \pm 3.43e+04}$	$9.22e+05 \pm 1.06e+05$	+	$2.11e+06 \pm 1.61e+05$	+	$3.59e+06 \pm 7.17e+05$	+
f_{20}	$\mathbf{1.49e+04 \pm 1.10e+03}$	$4.18e+10 \pm 2.02e+10$	+	$2.26e+09 \pm 3.42e+09$	+	$5.48e+11 \pm 2.10e+11$	+

TABLE IX
HOLM TEST ON THE FITNESS, REFERENCE ALGRITHM = μDEA (RANK = 3.24E+00)

j	Optimizer	$Rank$	z_j	p_j	δ/j	Hypothesis
1	SADE	3.68e+00	2.14e+00	9.84e-01	5.00e-02	Accepted
2	MDE-pBX	3.08e+00	-7.54e-01	2.25e-01	2.50e-02	Accepted
3	μDE	2.53e+00	-3.39e+00	3.46e-04	1.67e-02	Rejected
4	JADE	2.46e+00	-3.71e+00	1.05e-04	1.25e-02	Rejected

"Accepted" if the distribution of values can be considered the same (there is no out-performance).

As shown in Table IX, the proposed μDEA is ranked second after SADE over all the 76 problems included in this study, thus confirming that μDEA is a valuable algorithm that makes use of a micro-population.

IV. CONCLUSION

This paper proposes a micro-Differential Evolution scheme that includes, in a memetic fashion, a shallow local search that performs, a limited amount of times exploitation in the directions of each variable of the candidate solution displaying the best performance. The extra moves according to the axes complement the search carried out by the DE logic and support the external workshop to detect solutions with a high performance. More specifically, DE schemes, even when characterized by a small population, tend to keep the candidate solutions far from each other. This may result into an excessive exploration, especially in high dimensions, thus resulting into an undesired stagnation condition. The extra moves increase the exploitation of the algorithm and allow an overall better performance. The comparison with modern DE based algorithms show that the proposed algorithm, notwithstanding its simplicity, is nearly as good as them for low dimensional problem, thus displaying a respectable performance. The comparison in large scale domains show that the proposed algorithm outperforms all the other algorithms contained in this study. From this finding, we can conclude that small populations in DE schemes can lead to performance higher than that of DE schemes that use large populations.

The proposed micro-Differential Evolution implementation appears a good and robust alternative that can be promisingly applied in those application characterized by a limted hardware, such as embedded systems, and in those problems that impose a modest computational overhead, such as real-time optimization problems. Future work will consider randomized operators and mechanisms that impose a narrowing of the search in the late stage of the optimization.

ACKNOWLEDGMENT

This research is supported by the Academy of Finland, Akatemiatutkija 130600, "Algorithmic design issues in Memetic Computing". The numerical experiments have been carried out on the computer network of the De Montfort University by means of the software for distributed optimization Kimeme [44]. We thank Dr. Lorenzo Picinali, Dr. Nathan Jeffery and David Tunnicliffe for the technical support.

REFERENCES

[1] A. E. Eiben and S. K. Smit, "Parameter tuning for configuring and analyzing evolutionary algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 19–31, 2011.

[2] J. Brest and M. S. Maučec, "Population size reduction for the differential evolution algorithm," *Applied Intelligence*, vol. 29, no. 3, pp. 228–247, 2008.

[3] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, "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*, vol. 37, no. 1, pp. 28–41, 2007.

[4] F. Neri, J. I. Toivanen, G. L. Cascella, and Y. S. Ong, "An Adaptive Multimeme Algorithm for Designing HIV Multidrug Therapies," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 2, pp. 264–278, 2007.

[5] N. S. Teng, J. Teo, and M. H. A. Hijazi, "Self-adaptive population sizing for a tune-free differential evolution," *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, vol. 13, no. 7, pp. 709–724, 2009.

[6] A. Prügel-Bennett, "Benefits of a population: Five mechanisms that advantage population-based algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 4, pp. 500–517, 2010.

[7] N. J. Radcliffe, "Forma analysis and random respectful recombination," in *Proceedings of the 4th Int. Conf. Genet. Algorithms*. Morgan Kaufmann, 1991, pp. 222–229.

[8] G. Iacca, F. Neri, E. Mininno, Y. S. Ong, and M. H. Lim, "Ockham's Razor in Memetic Computing: Three Stage Optimal Memetic Exploration," *Information Sciences*, vol. 188, pp. 17–43, 2012.

[9] T. Chen, K. Tang, G. Chen, and X. Yao, "A large population size can be unhelpful in evolutionary algorithms," *Theoretical Computer Science*, vol. 436, pp. 54–70, 2012.

[10] K. E. Parsopoulos, "Cooperative micro-differential evolution for high-dimensional problems," in *Proceedings of the conference on Genetic and evolutionary computation*, 2009, pp. 531–538.

[11] ——, "Parallel cooperative micro-particle swarm optimization: A master-slave model," *Applied Soft Computing*, vol. 12, no. 11, pp. 3552–3579, Nov. 2012.

[12] S. Dasgupta, S. Das, A. Biswas, and A. Abraham, "On stability and convergence of the population-dynamics in differential evolution," *AI Communications - The European Journal on Artificial Intelligence*, vol. 22, no. 1, pp. 1–20, 2009.

[13] A. Rajasekhar, S. Das, and S. Das, "Abc: a micro artificial bee colony algorithm for large scale global optimization," in *GECCO (Companion)*, 2012, pp. 1399–1400.

[14] K. Krishnakumar, "Micro-genetic algorithms for stationary and non-stationary function optimization."

[15] C. A. Coello Coello and G. Toscano Pulido, "A micro-genetic algorithm for multiobjective optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO2001*. Springer-Verlag, 2001, pp. 126–140.

[16] A. Rajasekhar, S. Das, and P. N. Suganthan, "Design of fractional order controller for a servohydraulic positioning system with micro artificial bee colony algorithm," in *IEEE Congress on Evolutionary Computation*, 2012, pp. 1–8.

[17] V. Tam, K.-Y. Cheng, and K.-S. Lui, "Using micro-genetic algorithms to improve localization in wireless sensor networks," *Journal of Communications*, vol. 1, no. 4, pp. 137–141, 2006.

[18] F. Viveros-Jimènez, E. Mezura-Montes, and A. Gelbukh, "Empirical analysis of a micro-evolutionary algorithm for numerical optimization," *International Journal of Physical Sciences*, vol. 7, no. 8, pp. 1235–1258.

[19] S. Rahnamayan and H. R. Tizhoosh, "Image thresholding using micro opposition-based differential evolution (micro-ode)," in *IEEE Congress on Evolutionary Computation*, 2008, pp. 1409–1416.

[20] M. A. Sotelo-Figueroa, H. J. P. Soberanes, J. M. Carpio, H. J. F. Huacuja, L. C. Reyes, and J. A. S. Alcaraz, "Evolving bin packing heuristic using micro-differential evolution with indirect representation," in *Recent Advances on Hybrid Intelligent Systems*, ser. Studies in Computational Intelligence, 2013, vol. 451, pp. 349–359.

[21] S. Das and P. Suganthan, "Differential evolution: A survey of the state-of-the-art," *Evolutionary Computation, IEEE Transactions on*, vol. 15, no. 1, pp. 4–31, feb. 2011.

[22] K. V. Price, R. Storn, and J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005.

[23] F. Neri and V. Tirronen, "Recent Advances in Differential Evolution: A Review and Experimental Analysis," *Artificial Intelligence Review*, vol. 33, no. 1–2, pp. 61–106, 2010.

[24] F. Neri, G. Iacca, and E. Mininno, "Disturbed Exploitation compact Differential Evolution for Limited Memory Optimization Problems," *Information Sciences*, vol. 181, no. 12, pp. 2469–2487, 2011.

[25] L.-Y. Tseng and C. Chen, "Multiple trajectory search for Large Scale Global Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2008, pp. 3052–3059.

[26] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[27] J. Zhang and A. C. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive," vol. 13, no. 5, 2009, pp. 945–958.

[28] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[29] R. Mallipeddi, P. N. Suganthan, Q. K. Pan, and M. F. Tasgetiren, "Differential evolution algorithm with ensemble of parameters and mutation strategies," *Applied Soft Computing*, vol. 11, no. 2, pp. 1679–1696, 2011, the Impact of Soft Computing for the Progress of Artificial Intelligence.

[30] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential Evolution with a Neighborhood-based Mutation Operator," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 526–553, 2009.

[31] S. Islam, S. Das, S. Ghosh, S. Roy, and P. Suganthan, "An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 42, no. 2, pp. 482–500, april 2012.

[32] E. Mininno, F. Neri, F. Cupertino, and D. Naso, "Compact Differential Evolution," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 32–54, 2011.

[33] N. Noman and H. Iba, "Accelerating Differential Evolution Using an Adaptive Local Search," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 107–125, 2008.

[34] F. Neri and V. Tirronen, "Scale Factor Local Search in Differential Evolution," *Memetic Computing Journal*, vol. 1, no. 2, pp. 153–171, 2009.

[35] M. Weber, V. Tirronen, and F. Neri, "Scale Factor Inheritance Mechanism in Distributed Differential Evolution," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 14, no. 11, pp. 1187–1207, 2010.

[36] J. Lampinen and I. Zelinka, "On Stagnation of the Differential Evolution Algorithm," in *Proceedings of 6th International Mendel Conference on Soft Computing*, P. Oŝmera, Ed., 2000, pp. 76–83.

[37] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem Definitions and Evaluation Criteria for the CEC 2005 Special Session on Real-Parameter Optimization," Nanyang Technological University and KanGAL, Singapore and IIT Kanpur, India, Tech. Rep. 2005005, 2005.

[38] N. Hansen, A. Auger, S. Finck, R. Ros *et al.*, "Real-Parameter Black-Box Optimization Benchmarking 2010: Noiseless Functions Definitions," INRIA, Tech. Rep. RR-6829, 2010.

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

[40] K. Tang, X. Li, P. N. Suganthan, Z. Yang, and T. Weise, "Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization," University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL): Hefei, Anhui, China, Tech. Rep., 2010.

[41] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[42] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

[43] S. Garcia, A. Fernandez, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability," *Soft Computing*, vol. 13, no. 10, pp. 959–977, 2008.

[44] Cyber Dyne Srl Home Page, "Kimeme," 2012, http://cyberdynesoft.it/.

PV

MULTICRITERIA ADAPTIVE DIFFERENTIAL EVOLUTION FOR GLOBAL NUMERICAL OPTIMIZATION

by

Jixiang Cheng, Gexiang Zhang, Fabio Caraffini and Ferrante Neri 2015

Integrated Computer-Aided Engineering, pre-press, DOI 10.3233/ICA-150481

PVI

SUPER-FIT MULTICRITERIA ADAPTIVE DIFFERENTIAL EVOLUTION

by

Fabio Caraffini, Ferrante Neri, Jixiang Cheng, Gexiang Zhang, Lorenzo Picinali, Giovanni Iacca and Ernesto Mininno 2013

IEEE Congress on Evolutionary Computation (CEC), pages 1678-1685

2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

Super-fit Multicriteria Adaptive Differential Evolution

Fabio Caraffini * †, Ferrante Neri * †, Jixiang Cheng ‡, Gexiang Zhang ‡,
Lorenzo Picinali*, Giovanni Iacca§, Ernesto Mininno†

* Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University,
The Gateway, Leicester LE1 9BH, United Kingdom
Email: fabio.caraffini@email.dmu.ac.uk, fneri@dmu.ac.uk, and lpicinali@dmu.ac.uk

† Department of Mathematical Information Technology,
P.O. Box 35 (Agora), 40014 University of Jyväskylä, Finland
Email: fabio.caraffini@jyu.fi, ferrante.neri@jyu.fi, and ernesto.mininno@jyu.fi

§School of Electrical Engineering, Southwest Jiaotong University, Chengdu, P.R. China 610031
Email: chengjixiang0106@126.com and zhgxdylan@126.com

§INCAS^3 - Innovation Centre for Advanced Sensors and Sensor Systems
P.O. Box 797, 9400 AT Assen, The Netherlands
Email:giovanniiacca@incas3.eu

***Abstract*—This paper proposes an algorithm to solve the CEC2013 benchmark. The algorithm, namely Super-fit Multicriteria Adaptive Differential Evolution (SMADE), is a Memetic Computing approach based on the hybridization of two algorithmic schemes according to a super-fit memetic logic. More specifically, the Covariance Matrix Adaptive Evolution Strategy (CMAES), run at the beginning of the optimization process, is used to generate a solution with a high quality. This solution is then injected into the population of a modified Differential Evolution, namely Multicriteria Adaptive Differential Evolution (MADE). The improved solution is super-fit as it supposedly exhibits a performance a way higher than the other population individuals. The super-fit individual then leads the search of the MADE scheme towards the optimum. Unimodal or mildly multimodal problems, even when non-separable and ill-conditioned, tend to be solved during the early stages of the optimization by the CMAES. Highly multi-modal optimization problems are efficiently tackled by SMADE since the MADE algorithm (as well as other Differential Evolution schemes) appears to work very well when the search is led by a super-fit individual.**

I. Introduction

Differential Evolution (DE) is a simple, efficient, and versatile algorithm that has been successfully applied to many problems in engineering and applied sciences, see [1]. For example, in [2] a DE is applied to a sensor fusion problem. In [3] and [4] a modified DE is used to address a filter design in the context of signal processing. In [5] a DE based hybrid algorithm with compact structure has successfully applied in the field of industrial robotics. Other applications, in mobile and space robotics, of DE schemes with compact structures are reported in [6] and [7].

As reported in [8], although DE is a very good algorithmic framework, it is characterised by a wide margin of improvement. During the latest years, computer scientists have proposed many enhanced versions of the original DE schemes. Paper [8] highlights that DE schemes suffer from a limited amount of search moves and an excessive determinism in the search logic. These facts can lead to a stagnation condition. On the other hand, DE schemes can be be easily improved by integrating new search moves in the offspring generations. The pool of search moves can be easily broaden by including a certain degree of randomization in the generation of new solutions, see [9] and [10]. The amount of search moves in DE schemes has been also increased by explicitly using multiple search logics. For example, in [11] the use of multiple mutation strategies is proposed. In [12] and [13] a local search is integrated within the DE framework to support the offspring generation. In [4] multiple local search assist the offspring generation of DE.

A special way to perform the hybridisation between a DE framework and an explicit extra search logic, is by means of the super-fit memetic logic, see [14]. It has been observed that DE, unlike Evolutionary Algorithms EAs, performs well when one individual is characterised by a higher fitness with respect to the other population individuals. This effect is due to the fact that, in DE frameworks, the relationship between diversity and algorithmic performance is different with respect to other EAs, see [15], [16], [17], [18], and [18]. If we consider for example the Evolution Strategies (ES), a proper algorithmic functioning occurs when to the exploration necessity corresponds a high diversity condition and to the exploitation necessity corresponds a low diversity condition. In high diversity conditions, the algorithm is capable of exploring the entire decision space while in low diversity conditions the algorithm finalizes the search. DE schemes, unlike an ES, tend not to lose the diversity even when the algorithm is no longer producing enhancements.

978-1-4799-0454-9/13/$31.00 ©2013 IEEE

When one individual has a much better performance than the others, the pivot individual (or super-fit individual) guides the search and is then outperformed by the other individuals of the population. Thus, in super-fit schemes an exploitative search is applied to a solution that is then inserted into a DE population. In order to generate this desirable condition, specifically in [14] an exploitative search at the beginning of the DE search is performed by applying the Particle Swarm Optimization to one individual of the population that is then injected in the DE population. The success of the super-fit logic within DE schemes has been confirmed in [19], where a super-fit individual generated by means of the Rosenbrock Algorithm [20] is applied to a DE having compact structure.

The present paper, in order to obtain a competitive algorithm for solving the testbed introduced in [21], proposes a super-fit hybridization of a DE algorithm. The initial improvement is achieved by means of the Covariance Matrix Adaptation Evolution Strategy (CMAES), see [22] and [23]. The DE scheme employed in this case is the recently developed Multicriteria Adaptive Differential Evolution (MADE), [24]. The resulting algorithm is here termed Super-fit Multicriteria Adaptive Differential Evolution (SMADE).

The remainder of this paper is organized in the following way. Section II describes the operators composing the proposed algorithm and their combination in a memetic fashion. Section III displays the numerical results obtained in this study on the testbed defined in [21] and, finally, Section IV presents the conclusion of this work.

II. Super-fit Multicriteria Adaptive Differential Evolution

Without a loss of generality, in order to clarify the notation in this paper, we refer to the minimization problem of an objective function $f(\mathbf{x})$, where the candidate solution $\mathbf{x}$ is a vector of n design variables (or genes) in a decision space D.

A. Covariance Matrix Adaptation Evolution Strategy

The CMAES algorithm is a popular and efficient optimizer for continuous optimization problems. The main powerful advantage of CMAES is that it adapts the search by sampling solutions in a region of the decision space that takes the shape of the basin of attraction. This feature makes CMAES very efficient for non-separable and ill-conditioned problems, especially when the fitness landscape is not highly multimodal. More specifically, CMAES samples a set of candidate solutions from a multi-variate normal distribution. The equivalent of the variance in a multi-variate distribution is the covariance matrix whose dimensions are $n \times n$ and that gives the name to this algorithm. The fitness values of the sampled candidate solutions are calculated and the solution are then sorted. The performance of the sampled solutions are then used to update the state variables including the covariance matrix, i.e. modifying the shape of the multi-variate distribution for the new set of samples. Implementations details of CMAES are in [23] and [25]. It is worthwhile mentioning that several CMAES variants have been proposed in the literature, see e.g. [26], [27].

In the CMAES framework, at each step, λ new solutions are sampled from a multivariate normal distribution whose mean and covariance matrix are adaptively updated from a weighted sum of the best μ solutions in the population. The loop goes on until a stop condition is met. It is interesting to note that, compared to classic evolution strategies employing the (μ, λ) or $(\mu + \lambda)$ survivor selection schemes, CMAES is said to employ a $(\mu/\mu_w, \lambda)$ strategy, where μ_w represent the weights on the best μ individuals used to update the distribution model. For the sake of completeness, a simplified pseudo-code illustrating the main steps of CMAES, see Algorithm 1.

Algorithm 1 Pseudo-code of CMAES

```
initialize covariance matrix C = I, and step-size σ
initialize the mean vector m with a random sample in D
while CMAES stop criterion is not met do
    (activate Covariance Matrix Repairing method, if needed)
    sample λ new individuals from distribution N(m, σ²C)
    evaluate individuals and sort them based on their fitness
    update m based on a weighted sum of the best μ individuals
    update covariance matrix C and step-size σ
end while
return best individual x_e
```

B. Multicriteria Adaptive Differential Evolution

A population of NP individuals is sampled within the decision space D. As in DE schemes, each element is taken into consideration. Similar to several successful DE versions proposed in the literature, such as [11], [28], and [29], MADE makes use of multiple mutation/crossover strategies. More specifically, for each individual of the population, in order to generate the offspring, the four mutation/crossover strategies in Fig. 1 are considered. In accordance with the analysis reported in [8], the employment of multiple search strategies is here proposed to increase the pool of search moves and thus enhance the DE performance.

Each strategy is associated to a probability p_k for $k = \{1, 2, 3, 4\}$. These probability values dynamically vary during the evolution. At the beginning of the optimization process, the probability values are set to be the same, i.e. $p_k = 0.25$, $\forall k$. The probability values are then updated on the basis of a feedback on the evolution. More specifically, at the generation g the Normalized Relative Fitness Improvement (NRFI) and Normalized Distance to the Best Individual (NDBI), measuring respectively the fitness impact (η_i^g) and the diversity impact (τ_i^g) related to the individual i at the generation g, are calculated. These two metrics are formulated as follows:

1) Normalized relative fitness improvement (NRFI):

$$\eta_i^g = \frac{\tilde{\eta}_i^g}{\max\{\tilde{\eta}_i^g | i = 1, 2, \ldots, NP\}} \tag{1}$$

where

$$\tilde{\eta}_i^g = \frac{f(\mathbf{x}_i^g) - f(\mathbf{x}_i^{g+1})}{f(\mathbf{x}_i^g) - f(\mathbf{x}_{best}^g)} \tag{2}$$

2) Normalized distance to the best individual (NDBI):

$$\tau_i^g = \frac{\tilde{\tau}_i^g}{\max\{\tilde{\tau}_i^g | i = 1, 2, \ldots, NP\}} \tag{3}$$

1) rand/1/bin:

$$\mathbf{u}_{i,j}^{g} = \begin{cases} \mathbf{x}_{r_1,j}^{g} + F(i) \cdot (\mathbf{x}_{r_2,j}^{g} - \mathbf{x}_{r_3,j}^{g}) & \text{if } \text{rand}_j(0,1) \leq Cr(i) \text{ or } j = j_r \\ \mathbf{x}_{i,j}^{g} & \text{otherwise} \end{cases}$$

2) rand/2/bin:

$$\mathbf{u}_{i,j}^{g} = \begin{cases} \mathbf{x}_{r_1,j}^{g} + F(i) \cdot (\mathbf{x}_{r_2,j}^{g} - \mathbf{x}_{r_3,j}^{g}) + F(i) \cdot (\mathbf{x}_{r_4,j}^{g} - \mathbf{x}_{r_5,j}^{g}) & \text{if } \text{rand}_j(0,1) \leq Cr(i) \text{ or } j = j_r \\ \mathbf{x}_{i,j}^{g} & \text{otherwise} \end{cases}$$

3) rand-to-best/2/bin:

$$\mathbf{u}_{i,j}^{g} = \begin{cases} \mathbf{x}_{r_1,j}^{g} + F(i) \cdot (\mathbf{x}_{best,j}^{g} - \mathbf{x}_{i,j}^{g}) + F(i) \cdot (\mathbf{x}_{r_1,j}^{g} - \mathbf{x}_{r_2,j}^{g}) + F(i) \cdot (\mathbf{x}_{r_3,j}^{g} - \mathbf{x}_{r_4,j}^{g}) & \text{if } \text{rand}_j(0,1) \leq Cr(i) \text{ or } j = j_r \\ \mathbf{x}_{i,j}^{g} & \text{otherwise} \end{cases}$$

4) current-to-rand/1:

$$\mathbf{u}_{i}^{g} = \mathbf{x}_{i}^{g} + \text{rand}(0,1) \cdot (\mathbf{x}_{r_1}^{g} - \mathbf{x}_{i}^{g}) + F(i) \cdot (\mathbf{x}_{r_2}^{g} - \mathbf{x}_{r_3}^{g})$$

Fig. 1. Recombination strategies used by MADE

where

$$\tilde{\tau}_i^g = \sqrt{\sum_{j=1}^{D} \left(\mathbf{x}_{i,j}^g - \mathbf{x}_{best,j}^g\right)^2} \tag{4}$$

Here, NRFI measures the degree of fitness improvement and NDBI weights the potential of escaping from a local optimum. The two indexes are then combined to generate the impact γ_i^g related to each individual i at the generation g by means of the so called *number-of-dominating-impacts* method, consisting of the following:

$$\gamma_i^g = |\{(\eta_j^g, \tau_j^g) | (\eta_i^g, \tau_i^g) \succeq (\eta_j^g, \tau_j^g), j \in h_i^g\}| \tag{5}$$

where $|\cdot|$ denotes the cardinality of a set; $\mathbf{a} \succeq \mathbf{b}$ means that vector $\mathbf{a} = (a_1, \ldots, a_k)$ dominates vector $\mathbf{b} = (b_1, \ldots, b_k)$, i.e., $\forall i \in \{1, \ldots, k\}, a_i \geq b_i \wedge \exists i \in \{1, \ldots, k\}, a_i > b_i$; $h_i^g = \{\bigcup_{k=1}^{K} s_k^g\} \backslash s_{n_i}^g$ denotes the set of indexes of the target vectors which use the operator differing from the operator n_i used by $\mathbf{x}_i^g$.

On the basis of the calculated impact of the operator, the credit of each operator in the current generation is calculated by

$$r_k^g = \frac{\sum_{n_i \in s_k^g} \gamma_i^g}{|s_k^g|} \tag{6}$$

where $k = 1, \ldots, K$ (in our case $K = 4$ as we have four recombination strategies). Then the operator quality is updated by an additive relaxation mechanism, i.e.,

$$q_k^g = (1 - \beta) \cdot q_k^g + \beta \cdot r_k^g \tag{7}$$

where β ($\beta \in [0, 1]$) is the adaptation rate; q_k^0 is initialized with 1. Finally, the probability of each operator to be selected at the next generation is updated by

$$p_k^{g+1} = p_{min} + (1 - K \cdot p_{min}) \frac{q_k^g}{\sum_{k=1}^{K} q_k^g}. \tag{8}$$

Apart from the trial vector generation strategy, the control parameters F and Cr are also adaptively adjusted by introducing a degree of randomization and rewarding the successful strategies. Let sF_k and sCr_k denote the sets containing the LP (stands for Learning Period) most recent values of F and Cr associated with the kth strategy which generates trial vectors entering the next generation. At each generation g, for each target vector $\mathbf{x}_i^g$ that selects the kth trial vector generation strategy, the scale factor $F(i)$ is sampled according to a Cauchy distribution with a local parameter μF_k and a scale parameter 0.1 by:

$$F(i) = \text{randc}(\mu F_k, 0.1) \tag{9}$$

and re-sampled if $F(i) > 1$ or $F(i) \leq 0$; the crossover rate $Cr(i)$ is generated following a normal distribution with mean μCr_k and standard deviation 0.1 by

$$Cr(i) = \text{randn}(\mu Cr_k, 0.1). \tag{10}$$

and regenerated if $Cr(i) > 1$ or $Cr(i) < 0$.

In (9) and (10), the local parameter μF_k and the mean μCr_k are initialized as 0.5. At the end of each generation,for LP generations if $|sF_k| < LP$, the values of μF_k and μCr_k remain unchanged, otherwise old values of sF_k and sCr_k are first removed to guarantee $|sF_k| = |sCr_k| = LP$. Then they are updated by

$$\mu F_k = \frac{\sum_{F \in sF_k} F^2}{\sum_{F \in sF_k} F} \tag{11}$$

and

$$\mu Cr_k = \frac{\sum_{Cr \in sCr_k} Cr}{LP}. \tag{12}$$

By means of these procedure an offspring individual is eventually generated. As for all the DE based algorithms, if the offspring solution outperforms the parent that generated it, then the parent solution is replaced by its offspring otherwise no replacement occurs. For the sake of clarity, the pseudo-code describing the working principles of MADE is reported in Algorithm 2.

C. Super-fit Memetic Hybridization and parameter setting

A the beginning of the optimization process, a population of NP individuals is sampled within the decision space D. By following the principles explained in [14] and [19], in the proposed SMADE, CMAES is run for 30% of the total budget. At the end of this procedure, the best solution detected by CMAES is injected into the MADE initial population and

Algorithm 2 Pseudo-code of MADE algorithm

Require: NP, LP, β, p_{min}
$g = 0$, $K = 4$
Generate initial population $P^0 = \{\mathbf{x}_1^0, \mathbf{x}_2^0, \ldots, \mathbf{x}_{NP}^0\}$ and evaluate P^0
Set $q_k^0 = 1$, $p_k^0 = 0.25$, $\mu F_k = \mu Cr_k = 0.5$, $sF_k = sCr_k = \varnothing$, $k = 1, \ldots, K$
while termination criterion is not satisfied **do**
  for $i = 1$ to NP **do**
    Select a strategy k via a roulette-wheel selection scheme for $\mathbf{x}_i^g$
    Generate $F(i)$ by (9) and regenerate it if $F(i) > 1$ or $F(i) \leq 0$
    Generate $Cr(i)$ by (10) and regenerate it if $Cr(i) > 1$ or $Cr(i) < 0$
    Generate $\mathbf{u}_i^g$ using kth strategy and parameter values $F(i)$ and $Cr(i)$
    Evaluate $\mathbf{u}_i^g$ as $f(\mathbf{u}_i^g)$
    if $f(\mathbf{u}_i^g) \leq f(\mathbf{x}_i^g)$ **then**
      $\mathbf{x}_i^{g+1} = \mathbf{u}_i^g$; $sF_k = sF_k \cup \{F(i)\}$; $sCr_k = sCr_k \cup \{Cr(i)\}$
    else
      $\mathbf{x}_i^{g+1} = \mathbf{x}_i^g$
    end if
  end for
  for $k = 1$ to NP **do**
    Calculate fitness impact η_i^g by (1)
    Calculate diversity impact τ_i^g by (3)
    Calculate aggregated impact γ_i^g by (5)
  end for
  for $k = 1$ to K **do**
    Calculate reward r_k^g by (6)
    Update quality q_k^g by (7)
    Update probability p_k^g by (8)
    if $|sF_k| \geq LP$ **then**
      Remove old values from sF_k and sCr_k so that $|sF_k| = |sCr_k| = LP$
      Update μF_k and μCr_k by (11) and (12)
    end if
  end for
  $g = g + 1$
end while
Ensure: the best solution

replaces the worst solution of the population. The MADE algorithm continues the optimization until the budget reaches its limit. As for the CMAES part of the algorithm, we used the default parameter setting of the original Java implementation [25]. In particular, the evolution strategy was configured with the standard values $\lambda = \lfloor 4 + 3\ln(D) \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, and initial step-size $\sigma = 0.2$, which are broadly accepted in the literature. The parameters of MADE, with reference to Fig. 2, are chosen in the following way: $NP = 50$, $\beta = 0.7$, $p_{min} = 0.02$. As mentioned in the algorithmic description, we set as initial values $\mu F_k = \mu Cr_k = 0.5$ and $q_k = 1$. While the setting $p_{min} = 0.02$ appears to be good for all the problems, we suggest that NP and β are selected from the intervals $[20, 60]$ and $[0.4, 0.8]$. The initial values of μF_k and μCr_k are selected at the centre of their interval and let evolve according to formulas (11) and (12). Thus the parameters to be selected are essentially β and NP.

By means of this hybridization, we expect that uni-modal or mildly multi-modal problems are solved (or nearly solved) during the first 30% of the budget. Due to the CMAES features, non-separable and ill-conditioned problems, stated that they are not highly multi-modal, should quickly be solved. For highly multi-modal landscapes, although CMAES is not likely to detect solutions with an extraordinary high performance, its application can be beneficial to achieve some improvements. The improved solution will then act as a super-fit individual to guide the search of MADE towards the optimum. The choice of 30% has been carried out on the basis of a tuning and general considerations. Given the total budget for this competition, we observed that $1500 \times n$ fitness evaluations are enough, for CMAES, to achieve a reasonably good result and stop improving upon the solution. It has been observed that a shorter budget can be not enough to reach high quality solutions and a too long budget would not lead to any further improvement. Usually, uni-modal problems are very easily/quickly solved by CMAES while highly multi-modal problems are likely to never been solved by it. We noticed that if CMAES has not solved yet the problem within $1500 \times n$ fitness evaluation, the problem will unlikely be solved within a longer budget. On the other hand, MADE appears to efficiently explore the decision space and make use of the super-fit individual. Although sometimes MADE improvements can be slow, the action of this algorithmic component appears to be efficient when a long budget is assigned. On the basis of these consideration, we recommend that most of the budget is assigned to MADE and only a moderate portion of the budget is initially given to CMAES.

III. Numerical Results

The proposed SMADE has been run over the 28 test problems contained in the testbed CEC2013 described in [21]. The problems of the testbed have been considered in 10, 30, and 50 dimensions. The SMADE algorithm, for each test problem and dimensionality value, has been run 51 times. Each run has been continued for $10000 \times n$ fitness evaluations. Tables I, II, and III report best, worst, median, and mean fitness values (in terms of fitness error at the end of the budget) as well the corresponding standard deviation in 10, 30, and 50 dimensions.

The proposed SMADE succeeded at solving, at least once, eight problems in 10D, seven problems in 30D, and five problems in 50D.

Fig.s 2, 3, and 4 show the average performance trends for some of the test problems listed in Tables I, II, and III. More specifically, in each figure, one example where CMAES solve the problems in the early stages of the optimization is depicted. This is the case of unimodal and moderately multi-modal optimization problems with or without ill-conditioning, such as **f1**, **f2**, **f3**, and **f4**. In addition, in each figure, the average trends of some cases where the initial improvement and thus the super-fit individual assists the DE based evolution is also shown. This would be the case of highly multi-modal and complex problems, such as **f17**, **f20**, and **f25**.

TABLE I. RESULTS OF SMADE IN 10D.

Func.	Best	Worst	Median	Mean	Std
1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
2	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
3	$0.00e+00$	$6.32e+00$	$0.00e+00$	$2.48e-01$	$1.23e+00$
4	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
6	$0.00e+00$	$9.81e+00$	$9.81e+00$	$5.41e+00$	$4.76e+00$
7	$7.03e-03$	$2.32e+01$	$3.05e-01$	$2.27e+00$	$4.45e+00$
8	$2.00e+01$	$2.05e+01$	$2.04e+01$	$2.03e+01$	$1.03e-01$
9	$8.33e-01$	$3.84e+00$	$2.36e+00$	$2.29e+00$	$7.19e-01$
10	$0.00e+00$	$3.95e-02$	$1.23e-02$	$1.42e-02$	$9.58e-03$
11	$0.00e+00$	$9.95e-01$	$0.00e+00$	$9.75e-02$	$2.96e-01$
12	$1.99e+00$	$1.79e+01$	$6.96e+00$	$7.80e+00$	$4.10e+00$
13	$1.99e+00$	$2.73e+01$	$1.18e+01$	$1.21e+01$	$6.40e+00$
14	$1.87e-01$	$1.87e+01$	$3.48e+00$	$3.64e+00$	$4.39e+00$
15	$1.60e+02$	$1.14e+03$	$7.93e+02$	$7.36e+02$	$2.60e+02$
16	$3.00e-02$	$1.45e+00$	$2.90e-01$	$4.04e-01$	$3.14e-01$
17	$1.01e+01$	$1.08e+01$	$1.02e+01$	$1.03e+01$	$1.55e-01$
18	$1.42e+01$	$3.57e+01$	$2.50e+01$	$2.46e+01$	$4.68e+00$
19	$1.64e-01$	$6.79e-01$	$3.84e-01$	$3.95e-01$	$1.25e-01$
20	$1.60e+00$	$3.48e+00$	$2.67e+00$	$2.65e+00$	$4.48e-01$
21	$2.00e+02$	$4.00e+02$	$4.00e+02$	$3.83e+02$	$5.50e+01$
22	$8.29e+00$	$2.12e+02$	$2.60e+01$	$4.93e+01$	$5.33e+01$
23	$1.07e+01$	$1.38e+03$	$5.17e+02$	$5.78e+02$	$3.16e+02$
24	$1.17e+02$	$2.18e+02$	$2.05e+02$	$2.02e+02$	$1.76e+01$
25	$2.00e+02$	$2.08e+02$	$2.01e+02$	$2.02e+02$	$1.91e+00$
26	$1.02e+02$	$2.00e+02$	$1.08e+02$	$1.26e+02$	$3.69e+01$
27	$1.80e+02$	$4.00e+02$	$3.03e+02$	$3.37e+02$	$5.23e+01$
28	$3.00e+02$	$6.00e+02$	$3.00e+02$	$3.17e+02$	$6.87e+01$

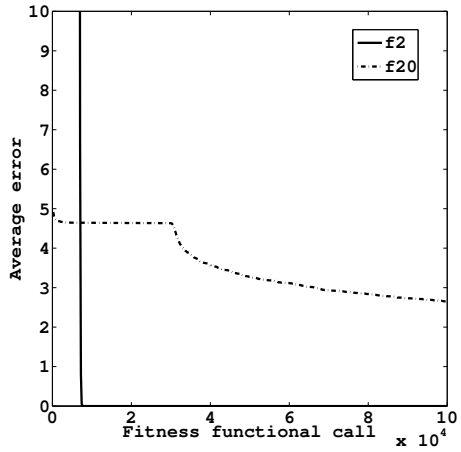


Fig. 2. Average fitness (error) trend of SMADE for **f2** and **f20** in 10D

TABLE II. RESULTS OF SMADE IN 30D.

Func.	Best	Worst	Median	Mean	Std
1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
2	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
3	$0.00e+00$	$3.52e+05$	$2.25e+00$	$9.82e+03$	$4.94e+04$
4	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
6	$0.00e+00$	$2.64e+01$	$0.00e+00$	$2.67e+00$	$7.85e+00$
7	$3.66e+00$	$6.53e+01$	$3.38e+01$	$3.25e+01$	$1.61e+01$
8	$2.08e+01$	$2.10e+01$	$2.10e+01$	$2.10e+01$	$4.80e-02$
9	$1.24e+01$	$3.15e+01$	$2.21e+01$	$2.23e+01$	$3.57e+00$
10	$0.00e+00$	$5.42e-02$	$1.23e-02$	$1.84e-02$	$1.34e-02$
11	$3.98e+00$	$1.99e+01$	$1.09e+01$	$1.09e+01$	$4.18e+00$
12	$2.89e+01$	$1.10e+02$	$5.47e+01$	$5.72e+01$	$1.70e+01$
13	$4.95e+01$	$2.24e+02$	$1.31e+02$	$1.28e+02$	$3.50e+01$
14	$1.31e+01$	$5.47e+02$	$7.29e+01$	$1.33e+02$	$1.27e+02$
15	$2.10e+03$	$6.12e+03$	$4.07e+03$	$4.10e+03$	$8.47e+02$
16	$3.46e-02$	$3.72e-01$	$1.06e-01$	$1.31e-01$	$7.57e-02$
17	$3.14e+01$	$4.08e+01$	$3.47e+01$	$3.48e+01$	$1.52e+00$
18	$5.28e+01$	$1.89e+02$	$8.12e+01$	$8.33e+01$	$2.06e+01$
19	$1.24e+00$	$3.90e+00$	$2.52e+00$	$2.55e+00$	$5.18e-01$
20	$8.54e+00$	$1.23e+01$	$1.05e+01$	$1.05e+01$	$8.07e-01$
21	$2.00e+02$	$4.44e+02$	$3.00e+02$	$3.27e+02$	$8.65e+01$
22	$1.27e+02$	$2.95e+02$	$1.66e+02$	$1.79e+02$	$4.50e+01$
23	$2.72e+03$	$6.41e+03$	$4.18e+03$	$4.22e+03$	$8.74e+02$
24	$1.40e+02$	$2.60e+02$	$2.40e+02$	$2.32e+02$	$2.57e+01$
25	$2.43e+02$	$2.94e+02$	$2.79e+02$	$2.78e+02$	$9.90e+00$
26	$1.37e+02$	$3.48e+02$	$2.00e+02$	$2.15e+02$	$5.25e+01$
27	$4.00e+02$	$8.56e+02$	$6.73e+02$	$6.47e+02$	$1.37e+02$
28	$1.00e+02$	$1.38e+03$	$3.00e+02$	$3.88e+02$	$3.23e+02$

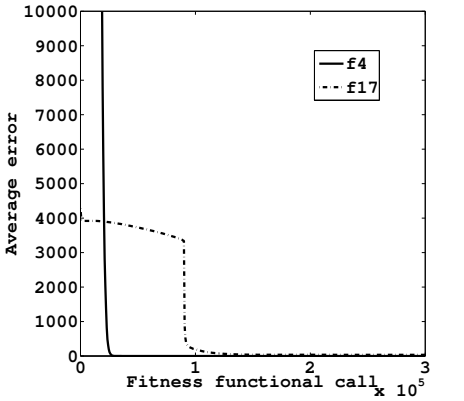


Fig. 3. Average fitness (error) trend of SMADE for **f4** and **f17** in 30D

TABLE III. RESULTS OF SMADE IN 50D.

Func.	Best	Worst	Median	Mean	Std
1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
2	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
3	$5.52e-06$	$9.30e+06$	$1.38e+04$	$3.81e+05$	$1.35e+06$
4	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
6	$0.00e+00$	$4.91e+01$	$4.34e+01$	$4.30e+01$	$6.28e+00$
7	$1.89e+01$	$1.02e+02$	$4.22e+01$	$4.32e+01$	$1.66e+01$
8	$2.10e+01$	$2.12e+01$	$2.11e+01$	$2.11e+01$	$3.85e-02$
9	$3.64e+01$	$5.25e+01$	$4.26e+01$	$4.36e+01$	$4.06e+00$
10	$0.00e+00$	$7.14e-02$	$2.46e-02$	$2.47e-02$	$1.48e-02$
11	$2.39e+01$	$8.76e+01$	$4.68e+01$	$4.81e+01$	$1.49e+01$
12	$7.06e+01$	$2.77e+02$	$1.55e+02$	$1.57e+02$	$4.52e+01$
13	$2.07e+02$	$4.57e+02$	$3.40e+02$	$3.35e+02$	$5.63e+01$
14	$5.41e+01$	$7.90e+02$	$3.14e+02$	$3.41e+02$	$2.05e+02$
15	$6.27e+03$	$1.05e+04$	$8.74e+03$	$8.54e+03$	$9.77e+02$
16	$1.95e-02$	$2.38e-01$	$7.91e-02$	$8.96e-02$	$4.24e-02$
17	$5.46e+01$	$8.45e+01$	$6.46e+01$	$6.57e+01$	$5.27e+00$
18	$1.19e+02$	$2.67e+02$	$1.91e+02$	$1.93e+02$	$3.46e+01$
19	$3.11e+00$	$7.63e+00$	$5.57e+00$	$5.43e+00$	$1.07e+00$
20	$1.72e+01$	$2.12e+01$	$1.92e+01$	$1.92e+01$	$8.86e-01$
21	$2.00e+02$	$1.12e+03$	$8.36e+02$	$8.46e+02$	$3.43e+02$
22	$8.69e+01$	$1.18e+03$	$2.64e+02$	$3.39e+02$	$2.24e+02$
23	$6.34e+03$	$1.30e+04$	$1.06e+04$	$9.89e+03$	$1.90e+03$
24	$2.72e+02$	$3.26e+02$	$3.02e+02$	$3.00e+02$	$1.20e+01$
25	$3.29e+02$	$3.92e+02$	$3.69e+02$	$3.68e+02$	$1.36e+01$
26	$1.82e+02$	$4.15e+02$	$2.01e+02$	$2.91e+02$	$9.70e+01$
27	$8.10e+02$	$1.57e+03$	$1.17e+03$	$1.18e+03$	$1.67e+02$
28	$4.00e+02$	$4.26e+03$	$4.00e+02$	$1.07e+03$	$1.27e+03$

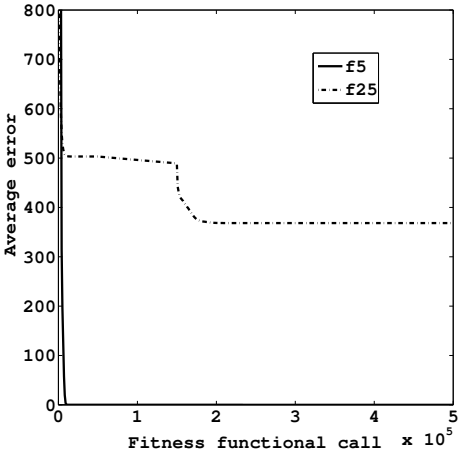


Fig. 4. Average fitness (error) trend of SMADE for **f5** and **f25** in 50D

TABLE IV. COMPUTATIONAL COMPLEXITY.

	$\mathbf{T0}$	$\mathbf{T1}$	$\widehat{\mathbf{T2}}$	$(\widehat{\mathbf{T2}} - \mathbf{T1})/\mathbf{T0}$
$\mathbf{D = 10}$	21.0	824.0	1847.0	48.7
$\mathbf{D = 30}$		2005.0	5764.2	179
$\mathbf{D = 50}$		3417.0	13280.8	469.7

TABLE V. HOLM TEST ON THE FITNESS, REFERENCE ALGORITHM = SMADE (RANK = 3.18E+00)

j	Optimizer	$Rank$	z_j	p_j	δ/j	Hypothesis
1	MDE-pBX	2.49e+00	-4.47e+00	3.82e-06	5.00e-02	Rejected
2	CCPSO2	2.23e+00	-6.17e+00	3.37e-10	2.50e-02	Rejected
3	CMAES	1.87e+00	-8.49e+00	1.06e-17	1.67e-02	Rejected

A. Computational Complexity

Table IV reports four measures of the computational complexity of SMADE by following the procedure reported in [21]. More specifically, $\mathbf{T0}$ is the execution time of a benchmark program performing some simple mathematical operations in double precision, $\mathbf{T1}$ is the time needed to compute 200000 evaluations of the benchmark function $\mathbf{f14}$, and $\widehat{\mathbf{T2}}$ it the average execution time (over 5 repetitions) of a run of SMADE while performing 200000 evaluations on the benchmark function $\mathbf{f3}$. Parameters $\mathbf{T1}$, $\mathbf{T1}$ and $\widehat{\mathbf{T2}}$ are here expressed in ms. A fourth, adimensional metric, is also defined as $(\widehat{\mathbf{T2}} - \mathbf{T1})/\mathbf{T0}$. In order to show how the algorithmic complexity varies with the problem dimension, $\mathbf{T1}$, $\widehat{\mathbf{T2}}$, $(\widehat{\mathbf{T2}} - \mathbf{T1})/\mathbf{T0}$ are reported for the three dimensionality values considered in this study (10, 30 and 50).

As in our previous experiments, the algorithm, as well as the program to compute its complexity, was coded in Java, wrapping the C/C++ implementation of the benchmark functions through JNI. The program was executed, in single thread, on a Linux Ubuntu system, version 12.04.2, with 2.66 GHz Intel Core 2 Quad processor, and 4 GB RAM. The timestamps were measured through the Java system call `System.currentTimeMillis()`.

The proposed SMADE contains two fairly complex components. The use of a covariance matrix has an important impact on the computational times as it introduces a quadratic complexity in the early stages of the optimization. On the other hand, the presence of CMAES at the beginning of the optimization is of great help to solve some problems in the benchmark and assists the DE framework to achieve solutions with a high quality. Although the MADE framework requires some non-trivial calculations to select the mutation strategy, its complexity is much lower than that of CMAES. In other words, the impact of MADE on complexity after the first 30% of the budget partially mitigates that of CMAES usage to generate the super-fit individuals. The computational complexity of the entire SMADE is thus super-linear but still sub-quadratic. This property makes the SMADE algorithm as an attractive option for optimization problem with a medium-relatively high number of dimensions.

B. Comparison against state-of-the-art algorithms

In order to provide a clear understanding of the SMADE performance, we compared the results shown in the previous section - on the same CEC 2013 benchmark and under the same conditions of budget and repetitions - with three different state-of-the-art optimization algorithms, namely the original CMAES [23], the Self-Adaptive Differential Evolution with pBX crossover (MDE-pBX) [29], and the Cooperatively Coevolving Particle Swarms Optimizer (CCPSO2) [30]. As suggested in their original papers, MDE-pBX was run with population size equal to 100 individuals and group size q equal to 15% of the population size, while CCPSO2 was executed with population size equal to 30 individuals, Cauchy/Gaussian sampling selection probability $p = 0.5$ and set of potential group sizes $S = \{2, 5\}$, $S = \{2, 5, 10\}$, $S = \{2, 5, 10, 25\}$, for experiments in 10, 30 and 50 dimensions, respectively. To provide a fair comparison, CMAES and SMADE were configured with the same values of λ, μ, σ.

Tables VI, VII and VIII, show the average fitness error (with respect to the target fitness) and its standard deviation, over 51 repetitions, obtained by the four algorithms under comparison after $T = 10000 \times D$ fitness evaluations. We report also the result of the Wilcoxon test [31] applied, with confidence level 0.95, to each pair-wise comparison on the final values obtained by SMADE against CMAES, MDE-pBX and CCPSO2 ("+", "-" and "=" indicate, respectively, a better, worse or equivalent performance of SMADE against the other algorithms). It can been seen that SMADE is extremely competitive: indeed, it outperforms MDE-pBX in 12 cases in 10D, 16 cases in 30D, and 18 cases in 50D, thus showing an excellent scalability. Most impressively, SMADE systematically outperforms (with few exceptions) both CMAES and CCPSO2 in all the three dimensionality values. This result is confirmed by the Holm-Bonferroni prodedure [32], which we performed as described in [33], with confidence level 0.05. As shown in Table V, SMADE is ranked first among the four algorithms, with the null-hypothesis (that SMADE and one of the other algorithms have indistinguishable performances) rejected in all the pair-wise comparisons.

IV. CONCLUSION

This paper proposes a novel algorithm for tackling the twenty-eight optimization problems listed in the CEC2013 benchmark in 10D, 30D, and 50D. The proposed algorithm applies the super-fit memetic logic to enhance upon a modified DE scheme. The super-fit logic consists of enhancing upon the initial performance of one solution by means of an exploitative component and injecting the improved solution into a starting population of DE. The solution has been here improved by means of the popular CMAES, thus allowing that unimodal and mildly multi-modal fitness landscapes are nearly solved in the initial stages of the optimization. The DE scheme here used is modified by the presence of multiple mutation strategies and an adaptive mechanism that tends to reward the most successful ones. This structure should then be able to handle high multi-modalities and thus continue the search performed by CMAES towards the optimum.

Numerical results show that the proposed SMADE is capable to solve several problems in the benchmark and can consistently achieve good results.

ACKNOWLEDGMENT

INCAS[3] is co-funded by the Province of Drenthe, the Municipality of Assen, the European Fund for Regional De-

TABLE VI. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =SMADE) FOR SMADE AGAINST MDE-PBX, CMAES, AND CCPSO2, ON CEC2013[21] IN 10 DIMENSIONS.

	SMADE	MDE-pBX		CMAES		CCPSO2	
f_1	$\mathbf{0.00e+00 \pm 0.00e+00}$	$0.00e+00 \pm 0.00e+00$	boh	$0.00e+00 \pm 0.00e+00$	=	$1.92e-04 \pm 1.16e-03$	+
f_2	$\mathbf{0.00e+00 \pm 0.00e+00}$	$4.15e+02 \pm 9.60e+02$	+	$0.00e+00 \pm 0.00e+00$	=	$9.93e+05 \pm 7.58e+05$	+
f_3	$\mathbf{2.48e-01 \pm 1.23e+00}$	$4.96e+03 \pm 4.66e+04$	+	$5.69e-01 \pm 1.81e+00$	=	$2.13e+07 \pm 3.13e+07$	+
f_4	$\mathbf{0.00e+00 \pm 0.00e+00}$	$6.50e-02 \pm 6.38e-01$	+	$0.00e+00 \pm 0.00e+00$	=	$8.80e+03 \pm 2.50e+03$	+
f_5	$\mathbf{0.00e+00 \pm 0.00e+00}$	$0.00e+00 \pm 7.54e-14$	=	$0.00e+00 \pm 0.00e+00$	=	$2.94e-03 \pm 8.06e-03$	+
f_6	$5.41e+00 \pm 4.76e+00$	$6.18e+00 \pm 4.73e+00$	+	$6.74e+00 \pm 6.74e+00$	=	$\mathbf{1.52e+00 \pm 2.91e+00}$	=
f_7	$\mathbf{2.27e+00 \pm 4.45e+00}$	$5.63e+00 \pm 7.94e+00$	+	$5.45e+08 \pm 5.38e+09$	+	$3.27e+01 \pm 8.31e+00$	+
f_8	$2.03e+01 \pm 1.03e-01$	$2.05e+01 \pm 1.06e-01$	+	$\mathbf{2.03e+01 \pm 1.32e-01}$	=	$2.04e+01 \pm 7.66e-02$	=
f_9	$\mathbf{2.29e+00 \pm 7.19e-01}$	$2.37e+00 \pm 1.41e+00$	=	$1.50e+01 \pm 3.65e+00$	+	$4.98e+00 \pm 9.13e-01$	+
f_{10}	$1.42e-02 \pm 9.58e-03$	$1.25e-01 \pm 9.20e-02$	+	$\mathbf{1.33e-02 \pm 1.39e-02}$	=	$1.47e+00 \pm 6.44e-01$	+
f_{11}	$\mathbf{9.75e-02 \pm 2.96e-01}$	$2.48e+00 \pm 1.61e+00$	+	$2.31e+02 \pm 2.67e+02$	+	$1.97e+00 \pm 1.18e+00$	+
f_{12}	$\mathbf{7.80e+00 \pm 4.10e+00}$	$1.06e+01 \pm 4.76e+00$	+	$3.49e+02 \pm 3.81e+02$	+	$2.64e+01 \pm 7.93e+00$	+
f_{13}	$\mathbf{1.21e+01 \pm 6.40e+00}$	$2.01e+01 \pm 7.94e+00$	+	$2.94e+02 \pm 3.99e+02$	+	$3.56e+01 \pm 8.30e+00$	+
f_{14}	$\mathbf{3.64e+00 \pm 4.39e+00}$	$1.25e+02 \pm 1.12e+02$	+	$1.88e+03 \pm 4.25e+02$	+	$5.27e+01 \pm 3.86e+01$	+
f_{15}	$7.36e+02 \pm 2.60e+02$	$\mathbf{7.26e+02 \pm 2.61e+02}$	=	$1.80e+03 \pm 3.92e+02$	+	$8.92e+02 \pm 2.18e+02$	+
f_{16}	$\mathbf{4.04e-01 \pm 3.14e-01}$	$5.43e-01 \pm 4.49e-01$	=	$4.51e-01 \pm 5.06e-01$	=	$1.18e+00 \pm 2.19e-01$	+
f_{17}	$\mathbf{1.03e+01 \pm 1.55e-01}$	$1.32e+01 \pm 1.85e+00$	+	$9.55e+02 \pm 3.42e+02$	+	$1.60e+01 \pm 2.07e+00$	+
f_{18}	$2.46e+01 \pm 4.68e+00$	$\mathbf{1.93e+01 \pm 4.67e+00}$	-	$9.01e+02 \pm 3.10e+02$	+	$5.41e+01 \pm 6.33e+00$	+
f_{19}	$\mathbf{3.95e-01 \pm 1.25e-01}$	$6.44e-01 \pm 2.09e-01$	+	$1.19e+00 \pm 5.00e-01$	+	$7.65e-01 \pm 2.52e-01$	+
f_{20}	$\mathbf{2.65e+00 \pm 4.48e-01}$	$2.87e+00 \pm 5.25e-01$	+	$4.68e+00 \pm 3.79e-01$	+	$3.50e+00 \pm 1.96e-01$	+
f_{21}	$3.83e+02 \pm 5.50e+01$	$3.98e+02 \pm 1.99e+01$	+	$\mathbf{3.68e+02 \pm 8.71e+01}$	=	$3.73e+02 \pm 6.75e+01$	-
f_{22}	$\mathbf{4.93e+01 \pm 5.33e+01}$	$1.33e+02 \pm 1.04e+02$	+	$2.32e+03 \pm 4.25e+02$	+	$7.27e+01 \pm 4.98e+01$	+
f_{23}	$\mathbf{5.78e+02 \pm 3.16e+02}$	$8.82e+02 \pm 3.08e+02$	+	$2.25e+03 \pm 4.48e+02$	+	$1.15e+03 \pm 2.58e+02$	+
f_{24}	$2.02e+02 \pm 1.76e+01$	$2.02e+02 \pm 1.56e+01$	=	$3.83e+02 \pm 1.57e+02$	+	$\mathbf{2.01e+02 \pm 2.54e+01}$	-
f_{25}	$2.02e+02 \pm 1.91e+00$	$\mathbf{2.00e+02 \pm 1.23e+01}$	-	$2.62e+02 \pm 4.92e+01$	+	$2.12e+02 \pm 1.09e+01$	+
f_{26}	$\mathbf{1.26e+02 \pm 3.69e+01}$	$1.47e+02 \pm 4.36e+01$	+	$2.57e+02 \pm 1.13e+02$	+	$1.58e+02 \pm 2.40e+01$	+
f_{27}	$3.37e+02 \pm 5.23e+01$	$\mathbf{3.06e+02 \pm 2.76e+01}$	-	$4.23e+02 \pm 1.35e+02$	+	$4.27e+02 \pm 4.91e+01$	+
f_{28}	$3.17e+02 \pm 6.87e+01$	$3.07e+02 \pm 5.78e+01$	=	$1.21e+03 \pm 1.21e+03$	+	$\mathbf{3.01e+02 \pm 1.28e+02}$	-

TABLE VII. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =SMADE) FOR SMADE AGAINST MDE-PBX, CMAES, AND CCPSO2, ON CEC2013[21] IN 30 DIMENSIONS.

	SMADE	MDE-pBX		CMAES		CCPSO2	
f_1	$0.00e+00 \pm 1.27e-13$	$0.00e+00 \pm 4.09e-13$	=	$\mathbf{0.00e+00 \pm 5.57e-14}$	=	$2.27e-13 \pm 1.21e-12$	+
f_2	$0.00e+00 \pm 1.74e-13$	$9.56e+04 \pm 6.16e+04$	+	$\mathbf{0.00e+00 \pm 7.88e-14}$	=	$9.95e+05 \pm 5.24e+05$	+
f_3	$9.82e+03 \pm 4.94e+04$	$1.80e+07 \pm 3.12e+07$	+	$\mathbf{1.53e+01 \pm 1.23e+02}$	-	$5.59e+08 \pm 5.57e+08$	+
f_4	$0.00e+00 \pm 1.86e-13$	$1.32e+01 \pm 5.46e+01$	+	$\mathbf{0.00e+00 \pm 5.08e-14}$	=	$5.57e+04 \pm 2.06e+04$	+
f_5	$1.14e-13 \pm 2.75e-13$	$1.14e-13 \pm 8.35e-13$	=	$\mathbf{1.14e-13 \pm 1.14e-14}$	=	$2.62e-08 \pm 6.05e-08$	+
f_6	$\mathbf{2.67e+00 \pm 7.85e+00}$	$1.99e+01 \pm 2.22e+01$	+	$3.05e+00 \pm 8.33e+00$	+	$2.19e+01 \pm 2.27e+01$	+
f_7	$\mathbf{3.25e+01 \pm 1.61e+01}$	$5.70e+01 \pm 1.77e+01$	+	$9.83e+03 \pm 6.44e+04$	=	$1.15e+02 \pm 3.11e+01$	+
f_8	$2.10e+01 \pm 4.80e-02$	$2.11e+01 \pm 5.94e-02$	+	$\mathbf{2.09e+01 \pm 6.66e-02}$	-	$2.10e+01 \pm 4.60e-02$	=
f_9	$2.23e+01 \pm 3.57e+00$	$\mathbf{2.22e+01 \pm 4.80e+00}$	=	$4.45e+01 \pm 6.99e+00$	+	$2.84e+01 \pm 2.08e+00$	+
f_{10}	$1.84e-02 \pm 1.34e-02$	$1.64e-01 \pm 1.20e-01$	+	$\mathbf{1.73e-02 \pm 1.25e-02}$	=	$1.48e-01 \pm 6.90e-02$	+
f_{11}	$1.09e+01 \pm 4.18e+00$	$4.62e+01 \pm 1.44e+01$	+	$9.47e+01 \pm 2.36e+02$	+	$\mathbf{1.19e-01 \pm 2.90e-01}$	-
f_{12}	$\mathbf{5.72e+01 \pm 1.70e+01}$	$6.94e+01 \pm 2.00e+01$	+	$7.11e+02 \pm 9.66e+02$	+	$2.12e+02 \pm 5.24e+01$	+
f_{13}	$\mathbf{1.28e+02 \pm 3.50e+01}$	$1.49e+02 \pm 3.66e+01$	+	$1.64e+03 \pm 1.66e+03$	+	$2.44e+02 \pm 3.44e+01$	+
f_{14}	$1.33e+02 \pm 1.27e+02$	$1.17e+03 \pm 3.95e+02$	+	$5.21e+03 \pm 7.37e+02$	+	$\mathbf{4.48e+00 \pm 2.87e+00}$	-
f_{15}	$4.10e+03 \pm 8.47e+02$	$3.95e+03 \pm 6.57e+02$	=	$5.37e+03 \pm 6.73e+02$	+	$\mathbf{3.85e+03 \pm 4.52e+02}$	=
f_{16}	$1.31e-01 \pm 7.57e-02$	$1.25e+00 \pm 6.19e-01$	+	$\mathbf{1.26e-01 \pm 8.87e-02}$	=	$2.16e+00 \pm 3.76e-01$	+
f_{17}	$3.48e+01 \pm 1.52e+00$	$7.05e+01 \pm 1.24e+01$	+	$3.95e+03 \pm 7.89e+02$	+	$\mathbf{3.07e+01 \pm 3.03e+00}$	-
f_{18}	$8.33e+01 \pm 2.06e+01$	$\mathbf{8.26e+01 \pm 1.89e+01}$	=	$4.23e+03 \pm 8.31e+02$	+	$2.31e+02 \pm 5.43e+01$	+
f_{19}	$2.55e+00 \pm 5.18e-01$	$9.54e+00 \pm 5.54e+00$	+	$3.66e+00 \pm 9.52e-01$	+	$\mathbf{7.77e-01 \pm 1.58e-01}$	-
f_{20}	$\mathbf{1.05e+01 \pm 8.07e-01}$	$1.07e+01 \pm 7.75e-01$	+	$1.50e+01 \pm 6.45e-02$	+	$1.35e+01 \pm 5.50e-01$	+
f_{21}	$3.27e+02 \pm 8.65e+01$	$3.40e+02 \pm 7.62e+01$	+	$3.05e+02 \pm 9.01e+01$	=	$\mathbf{2.37e+02 \pm 6.71e+01}$	-
f_{22}	$1.79e+02 \pm 4.50e+01$	$1.17e+03 \pm 4.92e+02$	+	$6.97e+03 \pm 1.06e+03$	+	$\mathbf{9.87e+01 \pm 6.70e+01}$	-
f_{23}	$\mathbf{4.22e+03 \pm 8.74e+02}$	$4.70e+03 \pm 7.70e+02$	+	$6.76e+03 \pm 6.82e+02$	+	$4.99e+03 \pm 6.31e+02$	+
f_{24}	$2.32e+02 \pm 2.57e+01$	$\mathbf{2.31e+02 \pm 8.60e+00}$	-	$8.19e+02 \pm 6.15e+02$	+	$2.80e+02 \pm 6.34e+00$	+
f_{25}	$\mathbf{2.78e+02 \pm 9.90e+00}$	$2.79e+02 \pm 1.38e+01$	=	$3.46e+02 \pm 1.45e+02$	+	$2.98e+02 \pm 6.94e+00$	+
f_{26}	$2.15e+02 \pm 5.25e+01$	$2.26e+02 \pm 5.15e+01$	=	$5.51e+02 \pm 5.14e+02$	+	$\mathbf{2.00e+02 \pm 6.76e-01}$	-
f_{27}	$\mathbf{6.47e+02 \pm 1.37e+02}$	$6.50e+02 \pm 1.04e+02$	=	$8.53e+02 \pm 2.42e+02$	+	$1.04e+03 \pm 8.09e+01$	+
f_{28}	$3.88e+02 \pm 3.23e+02$	$\mathbf{3.09e+02 \pm 1.50e+02}$	-	$1.96e+03 \pm 3.40e+03$	+	$4.35e+02 \pm 5.10e+02$	+

TABLE VIII. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE =SMADE) FOR SMADE AGAINST MDE-PBX, CMAES, AND CCPSO2, ON CEC2013[21] IN 50 DIMENSIONS.

	SMADE	MDE-pBX		CMAES		CCPSO2	
f_1	$\mathbf{2.27e-13 \pm 0.00e+00}$	$1.11e-11 \pm 7.68e-11$	+	$2.27e-13 \pm 0.00e+00$	=	$6.82e-13 \pm 1.31e-12$	+
f_2	$\mathbf{2.27e-13 \pm 0.00e+00}$	$4.37e+05 \pm 1.64e+05$	+	$2.27e-13 \pm 0.00e+00$	=	$1.85e+06 \pm 9.33e+05$	+
f_3	$3.81e+05 \pm 1.35e+06$	$8.45e+07 \pm 1.46e+08$	+	$\mathbf{9.64e+02 \pm 4.49e+03}$	-	$1.98e+09 \pm 2.09e+09$	+
f_4	$\mathbf{2.27e-13 \pm 0.00e+00}$	$3.05e+01 \pm 6.62e+01$	+	$2.27e-13 \pm 0.00e+00$	=	$1.00e+05 \pm 3.57e+04$	+
f_5	$6.82e-13 \pm 2.38e-12$	$1.93e-11 \pm 9.76e-11$	+	$\mathbf{6.82e-13 \pm 1.57e-12}$	=	$2.70e-10 \pm 8.03e-10$	+
f_6	$\mathbf{4.30e+01 \pm 6.28e+00}$	$5.48e+01 \pm 2.12e+01$	+	$4.32e+01 \pm 7.10e+00$	+	$4.35e+01 \pm 1.36e+01$	+
f_7	$4.32e+01 \pm 1.66e+01$	$6.59e+01 \pm 1.06e+01$	+	$\mathbf{4.19e+01 \pm 1.70e+01}$	=	$1.37e+02 \pm 2.31e+01$	+
f_8	$2.11e+01 \pm 3.85e-02$	$2.12e+01 \pm 4.44e-02$	+	$\mathbf{2.11e+01 \pm 9.75e-02}$	-	$2.11e+01 \pm 4.49e-02$	=
f_9	$4.36e+01 \pm 4.06e+00$	$\mathbf{4.32e+01 \pm 7.71e+00}$	=	$7.66e+01 \pm 7.77e+00$	+	$5.79e+01 \pm 4.39e+00$	+
f_{10}	$2.47e-02 \pm 1.48e-02$	$1.34e-01 \pm 1.23e-01$	+	$\mathbf{2.24e-02 \pm 1.49e-02}$	=	$1.24e-01 \pm 4.62e-02$	+
f_{11}	$4.81e+01 \pm 1.49e+01$	$1.24e+02 \pm 2.87e+01$	+	$2.19e+02 \pm 4.56e+02$	+	$\mathbf{4.31e-01 \pm 5.74e-01}$	-
f_{12}	$\mathbf{1.57e+02 \pm 4.52e+01}$	$1.58e+02 \pm 3.25e+01$	=	$2.25e+03 \pm 1.37e+03$	+	$4.46e+02 \pm 7.92e+01$	+
f_{13}	$3.35e+02 \pm 5.63e+01$	$\mathbf{3.24e+02 \pm 4.74e+01}$	=	$3.36e+03 \pm 1.09e+03$	+	$5.49e+02 \pm 6.67e+01$	+
f_{14}	$3.41e+02 \pm 2.05e+02$	$2.65e+03 \pm 8.86e+02$	+	$8.82e+03 \pm 1.04e+03$	+	$\mathbf{6.45e+00 \pm 3.20e+00}$	-
f_{15}	$8.54e+03 \pm 9.77e+02$	$\mathbf{7.46e+03 \pm 7.95e+02}$	-	$9.09e+03 \pm 9.43e+02$	+	$7.95e+03 \pm 7.11e+02$	-
f_{16}	$8.96e-02 \pm 4.24e-02$	$1.75e+00 \pm 7.40e-01$	+	$\mathbf{8.01e-02 \pm 4.72e-02}$	=	$2.39e+00 \pm 5.90e-01$	+
f_{17}	$6.57e+01 \pm 5.27e+00$	$1.75e+02 \pm 3.72e+01$	+	$6.97e+03 \pm 1.07e+03$	+	$\mathbf{5.14e+01 \pm 2.84e-01}$	-
f_{18}	$1.93e+02 \pm 3.46e+01$	$\mathbf{1.85e+02 \pm 3.40e+01}$	=	$7.08e+03 \pm 9.14e+02$	+	$4.94e+02 \pm 1.08e+02$	+
f_{19}	$5.43e+00 \pm 1.07e+00$	$4.25e+01 \pm 2.66e+01$	+	$6.32e+00 \pm 1.18e+00$	+	$\mathbf{1.40e+00 \pm 2.19e-01}$	-
f_{20}	$\mathbf{1.92e+01 \pm 8.86e-01}$	$2.00e+01 \pm 9.04e-01$	+	$2.50e+01 \pm 9.73e-02$	+	$2.28e+01 \pm 7.85e-01$	+
f_{21}	$8.46e+02 \pm 3.43e+02$	$9.22e+02 \pm 3.06e+02$	+	$8.12e+02 \pm 3.73e+02$	-	$\mathbf{3.27e+02 \pm 2.64e+02}$	-
f_{22}	$3.39e+02 \pm 2.24e+02$	$3.09e+03 \pm 9.98e+02$	+	$1.19e+04 \pm 1.26e+03$	+	$\mathbf{7.58e+01 \pm 8.58e+01}$	-
f_{23}	$9.89e+03 \pm 1.90e+03$	$\mathbf{8.88e+03 \pm 1.20e+03}$	-	$1.18e+04 \pm 8.52e+02$	+	$1.05e+04 \pm 1.11e+03$	=
f_{24}	$3.00e+02 \pm 1.20e+01$	$\mathbf{2.87e+02 \pm 1.47e+01}$	-	$1.64e+03 \pm 1.05e+03$	+	$3.56e+02 \pm 9.89e+00$	+
f_{25}	$\mathbf{3.68e+02 \pm 1.36e+01}$	$3.69e+02 \pm 1.78e+01$	=	$4.94e+02 \pm 1.88e+02$	+	$3.96e+02 \pm 1.19e+01$	+
f_{26}	$2.91e+02 \pm 9.70e+01$	$3.50e+02 \pm 7.93e+01$	+	$6.04e+02 \pm 7.11e+02$	=	$\mathbf{2.09e+02 \pm 3.92e+01}$	-
f_{27}	$\mathbf{1.18e+03 \pm 1.67e+02}$	$1.24e+03 \pm 1.56e+02$	=	$1.28e+03 \pm 2.51e+02$	+	$1.79e+03 \pm 8.78e+01$	+
f_{28}	$1.07e+03 \pm 1.27e+03$	$\mathbf{4.33e+02 \pm 3.26e+02}$	-	$3.27e+03 \pm 5.60e+03$	+	$6.33e+02 \pm 8.95e+02$	-

velopment and the Ministry of Economic Affairs, Peaks in the Delta. The work of FN is supported by the Academy of Finland, Akatemiatutkija 130600, "Algorithmic design issues in Memetic Computing" and Tutkijatohtori 140487, "Algorithmic Design and Software Implementation: a Novel Optimization Platform". The numerical experiments have been carried out on the computer network of the De Montfort University by means of the software for distributed optimization Kimeme [34]. The work of GZ and JC is supported by the National Natural Science Foundation of China (61170016), the Program for New Century Excellent Talents in University (NCET-11-0715) and SWJTU supported project (SWJTU12CX008)

REFERENCES

[1] K. Price and R. Storn, "Differential evolution: A simple evolution strategy for fast optimization," *Dr. Dobb's J. Software Tools*, vol. 22, no. 4, pp. 18–24, 1997.

[2] R. Joshi and A. C. Sanderson, "Minimal representation multisensor fusion using differential evolution," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 29, no. 1, pp. 63–76, 1999.

[3] N. Karaboga, "Digital IIR Filter Design Using Differential Evolution Algorithm," *EURASIP Journal on Applied Signal Processing*, vol. 8, pp. 1269–1276, 2005.

[4] V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi, "An enhanced memetic differential evolution in filter design for defect detection in paper production," *Evolutionary Computation*, vol. 16, pp. 529–555, 2008.

[5] F. Neri and E. Mininno, "Memetic Compact Differential Evolution for Cartesian Robot Control," *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, pp. 54–65, 2010.

[6] G. Iacca, F. Caraffini, and F. Neri, "Compact differential evolution light," *Journal of Computer Science and Technology*, vol. 27, no. 5, pp. 1056–1076, 2012.

[7] G. Iacca, F. Caraffini, E. Mininno, and F. Neri, "Robot base disturbance optimization with compact differential evolution light," in *EvoApplications*, ser. Lecture Notes in Computer Science. Springer, 2012, pp. 285–294.

[8] F. Neri and V. Tirronen, "Recent Advances in Differential Evolution: A Review and Experimental Analysis," *Artificial Intelligence Review*, vol. 33, no. 1–2, pp. 61–106, 2010.

[9] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, "Self-Adapting Control Parameters in Differential Evolution: A Comparative Study on Numerical Benchmark Problems," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[10] E. Mininno, F. Neri, F. Cupertino, and D. Naso, "Compact Differential Evolution," *IEEE Transactions on Evolutionary Computation*, vol. 15, no. 1, pp. 32–54, 2011.

[11] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[12] N. Noman and H. Iba, "Accelerating Differential Evolution Using an Adaptive Local Search," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 107–125, 2008.

[13] F. Neri and V. Tirronen, "Scale Factor Local Search in Differential Evolution," *Memetic Computing Journal*, vol. 1, no. 2, pp. 153–171, 2009.

[14] A. Caponio, F. Neri, and V. Tirronen, "Super-fit control adaptation in memetic differential evolution frameworks," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 13, pp. 811–831, 2009.

[15] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, "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*, vol. 37, no. 1, pp. 28–41, 2007.

[16] F. Neri, J. I. Toivanen, G. L. Cascella, and Y. S. Ong, "An Adaptive Multimeme Algorithm for Designing HIV Multidrug Therapies," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 2, pp. 264–278, 2007.

[17] F. Neri, G. L. Cascella, N. Salvatore, A. V. Kononova, and G. Acciani, "Prudent-Daring vs Tolerant Survivor Selection Schemes in Control Design of Electric Drives," in *Applications of Evolutionary Computing*, ser. Lecture Notes in Computer Science, Rothlauf, F. et al., Ed., vol. LNCS 3907. Springer, 2006, pp. 805–809.

[18] V. Tirronen and F. Neri, "Differential Evolution with Fitness Diversity Self-Adaptation," in *Nature-Inspired Algorithms for Optimisation, Studies in Computational Intelligence*, ser. Studies in Computational Intelligence, R. Chiong, Ed. Springer, 2009, vol. 193, pp. 199–234.

[19] G. Iacca, R. Mallipeddi, E. Mininno, F. Neri, and P. N. Suganthan, "Super-fit and Population Size Reduction Mechanisms in Compact Differential Evolution," in *Proceedings of IEEE Symposium on Memetic Computing*, 2011, pp. 21–28.

[20] H. H. Rosenbrock, "An automatic Method for finding the greatest or least Value of a Function," *The Computer Journal*, vol. 3, no. 3, pp. 175–184, 1960.

[21] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernndez-Daz, "Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization," Zhengzhou University and Nanyang Technological University, Zhengzhou China and Singapore, Tech. Rep. 201212, 2013.

[22] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[23] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

[24] J. Cheng, G. Zhang, F. Caraffini, and F. Neri, "Multicriteria adaptive differential evolution for global numerical optimization," *Applied Soft Computing*, submitted.

[25] N. Hansen, "The CMA Evolution Strategy," 2012, http://www.lri.fr/ hansen/cmaesintro.html.

[26] A. Auger and N. Hansen, "A Restart CMA Evolution Strategy With Increasing Population Size," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005, pp. 1769–1776.

[27] C. Igel, T. Suttorp, and N. Hansen, "A Computational Efficient Covariance Matrix Update and a (1+1)-CMA for Evolution Strategies," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM Press, 2006, pp. 453–460.

[28] J. Zhang and A. C. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive," vol. 13, no. 5, 2009, pp. 945–958.

[29] S. Islam, S. Das, S. Ghosh, S. Roy, and P. Suganthan, "An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 42, no. 2, pp. 482–500, april 2012.

[30] X. Li and X. Yao, "Cooperatively Coevolving Particle Swarms for Large Scale Optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 2, pp. 210–224, april 2012.

[31] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[32] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

[33] S. Garcia, A. Fernandez, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability," *Soft Computing*, vol. 13, no. 10, pp. 959–977, 2008.

[34] Cyber Dyne Srl Home Page, "Kimeme," 2012, http://cyberdynesoft.it/.

PVII

A CMA-ES SUPER-FIT SCHEME FOR THE RE-SAMPLED INHERITANCE SEARCH

by

Caraffini, F. and Iacca, G. and Neri, F. and Picinali, L. and Mininno, E. 2013

IEEE Congress on Evolutionary Computation (CEC), pages 1123-1130

2013 IEEE Congress on Evolutionary Computation
June 20-23, Cancún, México

A CMA-ES Super-fit Scheme for the Re-sampled Inheritance Search

Fabio Caraffini * †, Giovanni Iacca‡, Ferrante Neri * †, Lorenzo Picinali*, Ernesto Mininno†

* Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University,
The Gateway, Leicester LE1 9BH, United Kingdom
Email: fabio.caraffini@email.dmu.ac.uk, fneri@dmu.ac.uk, and lpicinali@dmu.ac.uk

† Department of Mathematical Information Technology,
P.O. Box 35 (Agora), 40014 University of Jyväskylä, Finland
Email: fabio.caraffini@jyu.fi, ferrante.neri@jyu.fi, and ernesto.mininno@jyu.fi

‡INCAS^3 - Innovation Centre for Advanced Sensors and Sensor Systems
P.O. Box 797, 9400 AT Assen, The Netherlands
Email:giovanniiacca@incas3.eu

***Abstract*—The super-fit scheme, consisting of injecting an individual with high fitness into the initial population of an algorithm, has shown to be a simple and effective way to enhance the algorithmic performance of the population-based algorithm. Whether the super-fit individual is based on some prior knowledge on the optimization problem or is derived from an initial step of pre-processing, e.g. a local search, this mechanism has been applied successfully in various examples of evolutionary and swarm intelligence algorithms. This paper presents an unconventional application of this super-fit scheme, where the super-fit individual is obtained by means of the Covariance Adaptation Matrix Evolution Strategy (CMA-ES), and fed to a single solution local search which perturbs iteratively each variable. Thus, compared to other super-fit schemes, the roles of super-fit individual generator and global optimizer are switched. To prevent premature convergence, the local search employs a re-sampling mechanism which inherits parts of the best individual while randomly sampling the remaining variables. We refer to such local search as Re-sampled Inheritance Search (RIS). Tested on the CEC 2013 optimization benchmark, the proposed algorithm, named CMA-ES-RIS, displays a respectable performance and a good balance between exploration and exploitation, resulting into a versatile and robust optimization tool.**

I. Introduction

Despite the important theoretical framework of the no free lunch theorem [1], which proved that a "universal" optimizer able to find the optimum on all possible fitness functions does not exist, or, in other words, that the average performance of any two algorithms on all possible fitness functions is equivalent, researchers all over the world are still putting a strong effort in designing robust and versatile algorithms for tackling diverse problems. The idea is that it is worthwhile, from both a conceptual and practical point of view, to study novel algorithmic structures that, if properly tuned, can guarantee a robust and reliable performance at least on on a broad range of optimization problems. With this aim in mind, the research community developed, in recent years, increasingly complex algorithms made up, in the Memetic Computing fashion, of multiple components (memes) and/or using sophisticated learning techniques able to adapt the algorithmic behaviour to the features of the fitness landscape, e.g. multi-modality, ill-conditioning, non-separability. Some examples of modern algorithms of this kind can be found in [2]-[6].

Such modern algorithmic structures, on the other hand, are often characterized by an excessive computational overhead, both in terms of memory and CPU operations, or by a high number of parameters which require a fine-tuning on the problem at hand. Thus, their applicability is sometimes limited in practical contexts. In addition to that, as shown in [7] and [8], minimalistic memetic approaches featuring two or three simple components, if properly coordinated, show in many cases at least the same performance as overwhelmingly complex algorithmic structures, yet requiring a much lower computational overhead. In [7], this finding was explained in the light of the Ockham's Razor, i.e. applying the principle of parsimony to Memetic Computing, and, in general, to the algorithmic design process: minimal structures composed of few simple memes should be preferred to complex ones.

In this paper, we propose to combine the law of parsimony with the super-fit logic by enhanching upon the performance of a multi-start local search. The latter, namely Re-sampled Inheritance Search (RIS), is a simple single-solution algorithm composed of two memes, exploitative local search and a re-sampling mechanism, respectively. The idea of super-fit individual, introduced in the context of Differential Evolution in [9] and [10], consists of "drugging" the initial population by injecting one individual (the so called super-fit) resulting from an exploitative search. The investment of an initial budget for performing an exploitative search pays off by a much high performance of the population-based framework, see [9] and [10]. In Differential Evolution frameworks, the super-fit individual appears to lead the search and attract the other candidate individuals towards the most promising area of the search space, similarly to the global best particle in swarm intelligence. This injection of a super-fit individual into the initial population has the clear effect of promoting exploitation

978-1-4799-0454-9/13/$31.00 ©2013 IEEE

over exploration. However, in [9] it is shown that, since Differential Evolution naturally maintains a certain degree of population diversity during the optimization process, the super-fit scheme allows a good balance between exploitation and exploration without excessively biasing the search but rather gently guiding it.

In the approach we propose here, instead of using a local search for super-fit individual generation as in [9] and [10], we use a population-based algorithm, namely the Covariance Adaptation Matrix Evolution Strategy (CMA-ES). Different from a local search, CMA-ES has a lower chance of being trapped into a local optimum, providing a super-fit individual which is likely close to the global optimum. After a limited number of fitness evaluations, the super-fit individual found by CMA-ES is then passed to RIS. The latter is supposed to improve upon it and, in case of premature convergence, restart the search inheriting some information from the best individual. The resulting algorithm, named CMA-ES-RIS, is thus characterized by a proper combination of exploration and exploitation. Most of all, its simple structure allows to distinguish clearly the contribution of each component to the overall algorithmic performance.

The remainder of this paper is organized as follows. Section II describes the CMA-ES-RIS structure and its components. Section III presents the numerical results obtained with the CMA-ES-RIS algorithm on the CEC 2013 real-parameter optimization benchmark in three different dimensionalities, namely 10, 30 and 50 and some considerations on its computational complexity. Finally, Section IV concludes this work and suggests a possible follow-up of this research.

II. The proposed approach

Without loss of generality, in the following we refer to the problem of minimizing an objective function $f(\mathbf{x})$, where the candidate solution $\mathbf{x}$ is a vector of n design variables (or genes) in a decision space $\mathbf{D}$. We indicate with $x[i]$ the i^{th} element of the vector $\mathbf{x}$. The proposed CMA-ES-RIS algorithm attempts to solve this problem at first generating a super-fit individual, and then refining the search to improve upon it. The two stages of the algorithm are executed sequentially by two different components, respectively CMA-ES and RIS, described in the next subsections. The general structure of CMA-ES-RIS is illustrated at the end of this section.

A. Covariance Adaptation Matrix Evolution Strategy (CMA-ES)

The Covariance Matrix Adaptation Evolution Strategy (CMA-ES) [11], [12], [13] is a powerful variant of classic Evolution Strategies (ES), see [14] and [15], which makes use of distribution model of the population. More specifically, at each step of the algorithm λ new solutions are sampled from a multivariate normal distribution whose mean and covariance matrix are adaptively updated from a weighted sum of the best μ solutions in the population. The loop goes on until a stop condition is met. It is interesting to note that, compared to classic evolution strategies employing the (μ, λ) or $(\mu + \lambda)$ survivor selection schemes, CMA-ES is said to employ a $(\mu/\mu_w, \lambda)$ strategy, where μ_w represent the weights on the best μ individuals used to update the distribution model. A detailed description of CMA-ES is out of the scope of this paper, while interested readers can find all relevant information in the vast literature on this algorithm. We report instead, for the sake of completeness, a simplified pseudo-code illustrating the main steps of CMA-ES, see Fig. 1.

```
initialize covariance matrix C = I, and step-size σ
initialize the mean vector m with a random sample in D
while CMA-ES stop criterion is not met do
    (activate Covariance Matrix Repairing method, if needed)
    sample λ new individuals from distribution N(m, σ²C)
    evaluate individuals and sort them based on their fitness
    update m based on a weighted sum of the best μ individuals
    update covariance matrix C and step-size σ
end while
return best individual x_e
```

Fig. 1. Pseudo-code of CMA-ES

Due to its solid theoretical background, its simple parameter setting, and its invariance to several transformations of the fitness function, CMA-ES is considered one of the state-of-the-art stochastic real-parameter optimization algorithms. In the last decade, it has been used as building block in a number of complex modern algorithms, either extending its original structure, introducing e.g. a restart scheme and an increasing population size as in [16]; hybridising it with other meta-heuristics, such as DE, see the DCMA-EA algorithm proposed in [17]; using it as an intense local search in a memetic structure, as proposed in [2]; or even simplifying its scheme as in [18], where a computationally efficient (1+1)-CMA-ES with Cholesky update for the covariance matrix is presented. Here we follow this research trend, using CMA-ES as super-fit individual generator for the Re-sampling Inheritance Search described in the next subsection.

As shown in Fig. 4, we here refer to the super-fit individual generated by CMA-ES as to $\mathbf{x_e}$ ("elite"). As a final remark, it is important to note that in this work we introduced into the original scheme of CMA-ES, the Covariance Matrix Repairing (CMR) method devised in [19] to fix ill-posed covariance matrices. This mechanism, activated whenever the distribution model has to be used to sample new solutions, works as follows: at the beginning, a correction factor $K = 1$ is initialized. Then, for a fixed number of iterations, first the eigenvalues of $\mathbf{C}$ are calculated: if all of them are positive, the loop is stopped as this condition ensures that $\mathbf{C}$ is well-posed; otherwise, if at least one of the eigenvalues is negative, i.e. $\mathbf{C}$ is ill-posed, the CMR update rule $\mathbf{C} = \mathbf{C} + |\gamma| \cdot K \cdot \mathbf{I}$ is applied, where γ is the minimum (negative) eigenvalue of $\mathbf{C}$. The factor K is subsequently multiplied by a factor $\Delta = 1.5$ and the loop goes on until a maximum number of iterations is reached. In [19], it is shown that 5 iterations are generally enough to guarantee the numerical stability of $\mathbf{C}$, as we also confirmed during our numerical experiments.

B. Re-sampling Inheritance Search (RIS)

The Re-sampling Inheritance Search (RIS) is an extremely simple single-solution algorithm for numerical optimization designed on the basis of the Ockham Razor's principle in MC, see [7]. The RIS algorithm makes use of only two operators which perturb, alternatively, a single solution. In the CMA-ES-RIS scheme, the initial solution is obviously the super-fit individual $\mathbf{x_e}$ generated by the CMA-ES during the previous

```
input: x_e
generate a random solution x_t within D
generate i = round(n · rand(0, 1))
compute Cr according to eq. (1)
x_t[i] = x_e[i]
k = 1
while rand(0, 1) ≤ Cr and k < n do
    x_t[i] = x_e[i]
    i = i + 1
    if i == n then
        i = 1
    end if
    k = k + 1
end while
if f(x_t) ≤ f(x_e) then
    x_e = x_t
end if
return x_t, x_e
```

Fig. 2. Pseudo-code of re-sampling with inheritance

```
input: x_t, x_e
ρ = δ · (ub − lb)
while ρ < ε do
    x_s = x_t
    for i = 1 : n do
        x_s[i] = x_t[i] − ρ
        if f(x_s) ≤ f(x_t) then
            x_t = x_s
        else
            x_s[i] = x_t[i] + ρ/2
            if f(x_s) ≤ f(x_t) then
                x_t = x_s
            end if
        end if
    end for
    if f(x_t) ≤ f(x_e) then
        x_e = x_t
    else
        ρ = ρ/2
    end if
end while
return x_e
```

Fig. 3. Pseudo-code of the exploitative local search

stage. Then, the following two operators are applied to perturb $\mathbf{x_e}$.

1) Re-sampling with inheritance: Similar to the exponential crossover used in Differential Evolution [20], this operator modifies the current solution $\mathbf{x_e}$ in order to generate a perturbed solution $\mathbf{x_t}$ ("trial"). With reference to Fig. 2, the operator works as follows: at the beginning, the trial solution is randomly sampled within the decision space $\mathbf{D}$. Then, one gene of $\mathbf{x_e}$ is randomly selected and copied into the corresponding gene of $\mathbf{x_t}$, to ensure that at least one gene of the current solution is inherited by the new individual. Starting from this gene, a copy process of the genes of $\mathbf{x_e}$ into the corresponding genes of $\mathbf{x_t}$ is continued until $r \leq Cr$, where r is a random number in $\mathcal{U}(0,1)$ – generated for each gene – and Cr (crossover rate) is a parameter affecting the number of transferred genes. As shown in [21], instead of setting Cr explicitly, we rather set the inheritance factor $alpha_e \approx n_e/n$, where n_e is the expectation of the number of genes copied from $\mathbf{x_e}$ into $\mathbf{x_t}$ in addition to the first gene, which is deterministically copied. Given that the probability that n_e genes are copied is $Cr^{n_e} = Cr^{n\alpha_e}$, it can be easily shown that, in order to achieve that approximately n_e genes are copied into the offspring with probability 0.5, the crossover rate can be set as:

$$Cr = \frac{1}{\sqrt[n\alpha_e]{2}}. \tag{1}$$

Thus Cr can be easily derived from the problem dimension n and the inheritance factor α_e, making its effect independent on problem dimension. As soon as $r > Cr$, the process is interrupted and the remaining genes of $\mathbf{x_t}$ remained unchanged. The copy process is applied as for a cyclic buffer, i.e. when the n^{th} gene is reached, the next to be copied is the first one.

Once the trial solution $\mathbf{x_t}$ has been generated, its fitness is computed and compared with that of $\mathbf{x_e}$. In case of improvement, $\mathbf{x_t}$ replaces $\mathbf{x_e}$; otherwise, the elite $\mathbf{x_e}$ is not modified. In both cases, the trial solution $\mathbf{x_t}$ is passed to the next operator, i.e. the exploitative local search, which will use it as a starting point. Thus, this mechanism can be seen as a "smart" restart for the exploitative local search, i.e. a non-destructive re-sampling which preserves parts of the genes of the current best individual in order to reuse the achievements obtained so far by the algorithm.

2) Exploitative local search: The exploitative local search algorithm belongs to the family of hill-descend algorithms; more specifically, it employs the search logic of one searcher proposed in [22], which attempts to improve upon a given solution perturbing separately, on both directions, each of its design variables. With reference to Fig. 3, this operator performs, iteratively, the following operations. At the beginning, the trial solution $\mathbf{x_t}$ generated by the first operator is copied into a temporary solution $\mathbf{x_s}$. The, for each i^{th} variable, the solution $\mathbf{x_s}$ is perturbed by computing:

$$x_s[i] = x_t[i] - \rho, \tag{2}$$

where ρ is the search radius, initialized as fraction δ of the search space $\mathbf{D}$. I.e., $\rho = \delta \cdot (ub - lb)$, where ub and lb are respectively the upper and lower bound of $\mathbf{D}$. If the newly perturbed solution $\mathbf{x_s}$ outperforms the initial trial solution $\mathbf{x_t}$, the trial solution $\mathbf{x_t}$ is replaced by $\mathbf{x_s}$, otherwise a perturbation in the opposite direction is performed, of width $\rho/2$:

$$x_s[i] = x_t[i] + \frac{\rho}{2} \tag{3}$$

and again $\mathbf{x_t}$ is replaced by $\mathbf{x_s}$ in case of improvement. After all the variables have been perturbed in both directions, the elite $\mathbf{x_e}$ is replaced by $\mathbf{x_t}$ if it is outperformed by it. If the elite is not updated, the radius ρ is halved further. The search is then repeated again, for all the design variables, within the new search radius. This process is continued until the radius ρ becomes smaller than a fixed threshold ε.

C. General structure of CMA-ES-RIS

Fig. 4 displays the global structure of CMA-ES-RIS, as serial activation of the components described before. The coordination of the components is rather straightforward. At the beginning of the algorithm, for a given fraction $r_{CMA-ES} \times T$ of the total computational budget T, CMA-ES is executed to generate a super-fit individual $\mathbf{x_e}$. This will be the best individual in the final population processed by CMA-ES. The super-fit individual is then passed to the RIS part of the algorithm, which is continued for the remaining budget $(1 - r_{CMA-ES}) \times T$. During each step of this second stage, first the re-sampling with inheritance processes the current elite

$\mathbf{x_e}$ and generates a trial solution $\mathbf{x_t}$, which is then processed by the exploitative local search. At the end of the local search, the (possibly improved) elite $\mathbf{x_e}$ is passed to the next iteration of RIS, and the loop goes on until the budget T is consumed.

Each component has a clear role in the algorithmic structure and gives a different contribution to the overall performance. As said before, the CMA-ES plays the role of super-fit individual generator: compared to other algorithms, CMA-ES efficiently exploits the parameter linkages, thanks to the covariance matrix, thus converging quickly – and reliably – towards a promising area of the search space. This area represents the starting point for the second stage, i.e. the RIS algorithm, a simple multi-start local search which, in the fashion of Memetic Computing, combines an explorative component (the re-sampling mechanism) with an exploitative local search. A crucial element of RIS is the exchange of information between the two components, in the form of a partial transmission of genes from the elite to the perturbed solution, which balances exploration and exploitation in a simple but apparently effective way.

While super-fit individual generated by CMA-ES is processed by the RIS a double action is taken. The exploitative local search attempts to quickly improve upon the algorithmic performance while the re-sampling mechanism allows an enhancement of the global search features and thus that the solution is trapped within a local optimum. The inheritance mechanism integrated within the re-sampling helps to make the search excessively exploratory and thus randomly search throughout the decision space. Finally, it must be observed that the exploitative local search has a very different search rule with respect to that of CMA-ES. While the first perturbs the solution by moving along the axes, the second follows the shape of the basin of attraction thus performing diagonal moves. The proposed CMA-ES-RIS is supposed to harmonically combine these features and offer a robust performance over a diverse array of optimization problems, including separable and non-separable problems.

From a complexity point of view, it has been shown [13] that, due to the covariance matrix update, the time complexity of the CMA-ES is quadratic. On the other hand, the re-sampling with inheritance performs, on average, $0.5 \cdot n + 1$ copies from the elite to the best, being n the problem dimensionality. In the worst case (which is however rather improbable), it performs n copies. Thus its complexity is $\mathcal{O}(n)$. As for the exploitative local search, it performs, at each step, $2 \cdot n$ perturbations. Therefore also its asymptotic complexity is linear. We can conclude then that the complexity of RIS is $\mathcal{O}(n)$, while the global complexity of CMA-ES-RIS (see below) is between $\mathcal{O}(n)$ and $\mathcal{O}(n^2)$, depending on the value of r_{CMA-ES}.

As a final remark, we note that CMA-ES-RIS makes use of a toroidal handling of the search space bounds: in other words, if $x[i]$ exceeds its upper bound $ub[i]$ by a quantity ζ ($x[i] = ub[i] + \zeta$), the replacement $x[i] = lb[i] + \zeta$ occurs. A specular mechanism is applied on the lower bound.

III. Numerical results

Following the rules of the CEC 2013 Special Session on Real-Parameter Optimization, we tested the proposed approach

```
CMA-ES stop criterion: r_CMA-ES × T
execute CMA-ES as in Fig. 1
RIS stop criterion: (1 − r_CMA-ES) × T
while RIS stop criterion is not met do
    execute re-sampling with inheritance as in Fig. 2
    execute exploitative local search as in Fig. 3
end while
return best individual x_e
```

Fig. 4. Pseudo-code of CMA-ES-RIS

on the 28 minimization problems defined in [23]. The whole benchmark was tested in $D = 10$, $D = 30$ and $D = 50$ dimensions, and for each test function and problem size the CMA-ES-RIS algorithm was executed 51 times, as dictated by the competition rules. Each run was continued for $T = 10000 \times D$ fitness evaluations. All the experiments were coded in Java, although we used the C/C++ original implementation of the CEC 2013 benchmark which was wrapped through the Java Native Interface (JNI) [24]. The 28×51 experiments were run, in parallel, on a heterogeneous network of Linux/Mac computers, through the distributed platform Kimeme [25].

A. Detailed results of CMA-ES-RIS

Tables III, IV and V show, respectively, the numerical results obtained with $D = 10$, $D = 30$ and $D = 50$ dimensions. For each test function, we report the best, worst, median and mean fitness value obtained at the end of the computational budget T, together with its standard deviation over the 51 runs available.

It can be seen that, apart from **f3** and **f5** (only in $50D$), which however show a small deviation from the global optimum, all the unimodal functions ($\mathbf{f1} - \mathbf{f5}$) are solved in all the three dimensioned considered. As for the multimodal functions ($\mathbf{f6} - \mathbf{f20}$), in 10 dimensions the optimum is detected in 4 cases, while the mean error ranges from $1.24e-02$ to $6.17e+02$ and the standard deviation ranges from $7.49e-02$ to $1.72e+02$, depending on the benchmark function. In 30 dimensions, the optimum is detected in 2 cases, while the mean error ranges from $6.94e-04$ to $3.13e+03$ and the standard deviation ranges from $5.41e-03$ to $4.53e+02$. In 50 dimensions, the optimum is detected in 1 case, while the mean error ranges from $8.60e-03$ to $6.30e+03$ and the standard deviation ranges from $5.86e-03$ to $6.74e+02$. Finally, regarding the composition functions ($\mathbf{f21} - \mathbf{f28}$), the mean error is in the order of $1e+02$ textdiv $1e+03$ for all the three dimensionalities, while the standard deviation ranges from $2.27e+01$ to $1.89e+02$, from $8.41e+00$ to $1.32e+03$, and from $3.25e-02$ to $1.53e+03$, respectively for 10, 30 and 50 dimensions.

It can be seen that, in case of the unimodal function **f1**, CMA-ES-RIS is able to quickly converge to the optimum for all the problem dimensionalities considered. As for the multimodal function **f16**, CMA-ES-RIS obtains, fast and reliably, a rather small average error. On the other hand, on the composition function **f24** the error is relatively large.

B. Comparison against state-of-the-art algorithms

To provide a deeper understanding of the performance of CMA-ES-RIS, we compared the results shown in the previous

TABLE I. HOLM-BONFERRONI TEST, REFERENCE = CMA-ES-RIS (RANK = 3.13E+00)

j	Optimizer	$Rank$	z_j	p_j	δ/j	Hypothesis
1	MDE-pBX	2.64e+00	-3.16e+00	7.80e-04	5.00e-02	Rejected
2	CCPSO2	2.24e+00	-5.79e+00	3.60e-09	2.50e-02	Rejected
3	CMA-ES	1.79e+00	-8.72e+00	1.41e-18	1.67e-02	Rejected

section - on the same CEC 2013 benchmark and under the same conditions of budget and repetitions - with three different state-of-the-art optimization algorithms, namely the original CMA-ES [13], the Self-Adaptive Differential Evolution with pBX crossover (MDE-pBX) [26], and the Cooperatively Coevolving Particle Swarms Optimizer (CCPSO2) [27]. As suggested in their original papers, MDE-pBX was run with population size equal to 100 individuals and group size q equal to 15% of the population size, while CCPSO2 was executed with population size equal to 30 individuals, Cauchy/Gaussian sampling selection probability $p = 0.5$ and set of potential group sizes $S = \{2, 5\}$, $S = \{2, 5, 10\}$, $S = \{2, 5, 10, 25\}$, for experiments in 10, 30 and 50 dimensions, respectively. To provide a fair comparison, CMA-ES and CMA-ES-RIS were configured with the same values of λ, μ, σ (see next section).

Tables VI, VII and VIII, show the average fitness error (with respect to the target fitness) and its standard deviation, over 51 repetitions, obtained by the four algorithms under comparison after $T = 10000 \times D$ fitness evaluations. We report also the result of the Wilcoxon test [28] applied, with confidence level 0.95, to each pair-wise comparison on the final values obtained by CMA-ES-RIS against CMA-ES, MDE-pBX and CCPSO2 ("+", "-" and "=" indicate, respectively, a better, worse or equivalent performance of CMA-ES-RIS against the other algorithms). It can been seen that CMA-ES-RIS is extremely competitive: indeed, it outperforms MDE-pBX in 12 cases in 10D, 16 cases in 30D, and 18 cases in 50D, thus showing an excellent scalability. Most impressively, CMA-ES-RIS systematically outperforms (with few exceptions) both CMA-ES and CCPSO2 in all the three dimensionalities. This result is confirmed by the Holm-Bonferroni prodedure [29], which we performed as described in [30], with confidence level 0.05. As shown in Table I, CMA-ES-RIS ranks first among the four algorithms, with the null-hypothesis (that CMA-ES-RIS and one of the other algorithms have indistinguishable performances) rejected in all the pair-wise comparisons.

C. Parameters

The CMA-ES-RIS algorithm was executed with the following parameters. As for the CMA-ES part of the algorithm, we used the default parameter setting of the original Java implementation [31]. In particular, the evolution strategy was configured with the standard values $\lambda = \lfloor 4 + 3\ln(D) \rfloor$, $\mu = \lfloor \lambda/2 \rfloor$, and initial step-size $\sigma = 0.2$, which are broadly accepted in the literature.

The RIS algorithm has instead three main parameters which affect its performance, namely the inheritance factor α_e, the initial search radius ρ and the stop-threshold ε. As said, the inheritance factor represents the approximate percentage of genes transferred during re-sampling, with probability 0.5, from the elite to the trial solution. Its value ranges in $(0, 1]$: the smaller α_e, the lower is the number of genes copied from the best individual, and vice versa. If one needs to improve the exploration capabilities of the algorithm, a smaller value of α_e should be used: this might be needed, for example, to tackle highly multi-modal landscapes. Otherwise, a larger value would increase the exploitation pressure.

Similarly, possible values for the fraction δ of the search space (used to initialize the search radius ρ in the exploitative local search) are $(0, 0.5]$, so that smaller values make the local search explore in a narrow neighbourhood of the current solution $\mathbf{x_s}$, and vice versa. This parameter should be related to the width of the basin(s) of attraction present in the fitness landscape, so that, if one has some prior knowledge about the fitness, the initial search radius should be as large as the smaller basin.

As for the stop-threshold ε, its value obviously affects the exploitation pressure of the local search: smaller values let RIS refine the search deeper, larger ones instead produce a coarser exploitation stage in favor of exploration. This parameter depends on the precision one wants to obtain on the fitness, and is related to the steepness (or, on the contrary, flatness) of the fitness landscape. Therefore ε may take arbitrarily small values, depending on the required precision.

Finally, the most crucial – and sensitive – parameter in the CMA-ES-RIS structure is the budget ratio between r_{CMA-ES} which determines the amount of budget assigned to both CMA-ES and RIS. This parameter ranges, theoretically, in $(0, 1)$. However, according to our empirical observation in order to have a well-balanced combination of the two algorithms this parameter should be selected within the range $[0.2, 0.5]$. A value larger than 0.5 would obviously assign more budget to the super-fit individual generation. As shown in the comparative experiments, the RIS application has an important impact on the performance with respect to the sole CMA-ES action, see Section III-B. This parameter should be set based both on theoretical and practical considerations: on one hand, allotting more budget to CMA-ES means that more effort is put in the first stage to learn the variable linkages and determine a super-fit individual; on the other, due to the higher complexity of CMA-ES with respect to RIS (see below), a larger value of r_{CMA-ES} increases the overall complexity of CMA-ES-RIS.

The numerical results shown in the previous section were obtained with $\alpha_e = 0.5$, $\delta = 0.2$, $\varepsilon = 10^{-6}$, and $r_{CMA-ES} = 0.3$. These values were selected tuning each parameter keeping the others at a fixed value chosen from a predefined set. More specifically, we tested the following sets of values: $\alpha_e = \{0.2, 0.5, 0.8\}$, $\delta = \{0.2, 0.3, 0.5\}$, $\varepsilon = \{10^{-10}, 10^{-6}, 10^{-4}\}$ and $r_{CMA-ES} = \{0.2, 0.3, 0.5\}$. We recorded the average fitness error (over 51 runs) obtained with CMA-ES-RIS on the whole CEC 2013 benchmark at the three different dimensionalities, and then selected, for each parameter, the value showing the best global mean fitness error. The performance obtained with the best parameter setting can be explained as follows: $\alpha_e = 0.5$ forces the trial solution to inherit at least half genome from the elite (with probability 0.5), thus guaranteeing a well-balanced inheritance ratio. Moreover, configuring the local search so that it explores a neighbourhood equal to $1/5$ of the whole search space provides a sufficiently large space for fitness improvements, still being enough focused on a restricted area of the search space. Similarly, the value $\varepsilon = 10^{-6}$ guarantees a good trade-off between the re-sampling with inheritance and the exploitation local search, as well as

$r_{CMA-ES} = 0.3$ guarantees the best trade-off between CMA-ES and RIS, allowing an optimal usage of the budget.

D. Algorithm complexity

Table II reports four measures of the computational complexity of the proposed CMA-ES-RIS, as defined in [23]. More specifically, $\mathbf{T0}$ is the execution time of a benchmark program performing some simple mathematical operations in double precision, $\mathbf{T1}$ is the time needed to compute 200000 evaluations of the benchmark function $\mathbf{f14}$, and $\widehat{\mathbf{T2}}$ it the average execution time (over 5 repetitions) of a run of CMA-ES-RIS performing 200000 evaluations of the benchmark function $\mathbf{f3}$. $\mathbf{T1}$, $\mathbf{T1}$ and $\widehat{\mathbf{T2}}$ are here expressed in ms. A fourth, adimensional metric, is also defined as $(\widehat{\mathbf{T2}} - \mathbf{T1})/\mathbf{T0}$. To show how the algorithm complexity changes with the problem dimension, $\mathbf{T1}$, $\widehat{\mathbf{T2}}$, $(\widehat{\mathbf{T2}} - \mathbf{T1})/\mathbf{T0}$ are reported for the three dimensionalities considered in this study (10, 30 and 50).

TABLE II. COMPUTATIONAL COMPLEXITY

	$\mathbf{T0}$	$\mathbf{T1}$	$\widehat{\mathbf{T2}}$	$(\widehat{\mathbf{T2}} - \mathbf{T1})/\mathbf{T0}$
$\mathbf{D = 10}$	27.0	544.0	496.0	-1.8
$\mathbf{D = 30}$		914.0	2451.6	56.9
$\mathbf{D = 50}$		1520.0	6390.2	180.4

As in our previous experiments, the algorithm, as well as the program to compute its complexity, was coded in Java, wrapping the C/C++ implementation of the benchmark functions through JNI. The program was executed, in single thread, on a Mac OS X system, version 10.7.5, with 2.2 GHz Intel Core i7 processor, and 4 GB RAM. The timestamps were measured through the Java system call `System.currentTimeMillis()`.

It can be seen that the time complexity of CMA-ES-RIS grows super-linearly (but sub-quadratically) with the problem dimensionality, showing a trade-off between the complexity of CMA-ES, which is quadratic, and that of RIS, which is linear, as proven previously. For the sake of completeness, it is interesting to note the also the space complexity is quadratic for the CMA-ES part (because of the covariance matrix, which requires n^2 elements), and linear for RIS (which instead requires only 3 n-dimensional vectors, i.e. $\mathbf{x_e}$, $\mathbf{x_s}$ and $\mathbf{x_t}$). Thus the proposed approach is mostly indicated for low-medium scale problems, while for large-scale problems the first part of the algorithm, based on CMA-ES, becomes naturally more time consuming.

IV. CONCLUSION

In this paper we introduced CMA-ES-RIS, a simple yet powerful optimization algorithm composed of two sequential elements: an initial CMA-ES step which is used to generate a super-fit individual, and a local search with inheritance which improves upon it. In the first stage, thanks to the covariance matrix adaptation mechanism, the features of the fitness landscape are implicitly analysed and exploited to rapidly converge to a promising area of the search space. Once this area is detected, the best individual in the final population of CMA-ES is selected as a starting point for the local search algorithm, which refines the search further. During this second stage, based on a fitness threshold, the algorithm is however allowed to perform a re-sampling with partial inheritance of the current best individual, so to prevent premature convergence and explore other areas of the search space. We presented the numerical results obtained on the CEC 2013 real-parameter optimization benchmark and discussed the parameter setting and time complexity of the proposed approach, showing that CMA-ES-RIS is able to obtain good solutions fast and reliably on a broad range of optimization problems. In the future, the CMA-ES-RIS structure will be further investigated testing different super-fit mechanisms and local search methods.

TABLE III. RESULTS FOR 10D

Func.	Best	Worst	Median	Mean	Std
1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
2	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
3	$0.00e+00$	$3.29e+01$	$0.00e+00$	$7.04e-01$	$4.57e+00$
4	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
6	$0.00e+00$	$9.81e+00$	$2.42e-03$	$1.10e+00$	$2.85e+00$
7	$1.99e+00$	$1.78e+02$	$4.01e+01$	$5.33e+01$	$4.63e+01$
8	$2.00e+01$	$2.06e+01$	$2.03e+01$	$2.03e+01$	$1.36e-01$
9	$1.10e+00$	$5.48e+00$	$3.63e+00$	$3.59e+00$	$1.03e+00$
10	$0.00e+00$	$5.17e-02$	$9.86e-03$	$1.24e-02$	$1.33e-02$
11	$0.00e+00$	$7.96e+00$	$2.98e+00$	$3.57e+00$	$1.46e+00$
12	$1.99e+00$	$2.59e+01$	$1.29e+01$	$1.29e+01$	$5.36e+00$
13	$4.48e+00$	$5.00e+01$	$2.64e+01$	$2.56e+01$	$1.07e+01$
14	$3.60e+00$	$2.60e+02$	$1.13e+02$	$1.02e+02$	$7.32e+01$
15	$1.42e+02$	$1.03e+03$	$6.33e+02$	$6.17e+02$	$1.72e+02$
16	$0.00e+00$	$3.55e-01$	$1.76e-01$	$1.64e-01$	$7.49e-02$
17	$1.69e+00$	$1.79e+01$	$1.18e+01$	$1.04e+01$	$3.70e+00$
18	$1.95e+01$	$4.59e+01$	$2.95e+01$	$2.98e+01$	$6.10e+00$
19	$3.94e-01$	$1.62e+00$	$7.61e-01$	$8.14e-01$	$2.71e-01$
20	$3.35e+00$	$4.70e+00$	$4.25e+00$	$4.16e+00$	$3.95e-01$
21	$1.00e+02$	$4.00e+02$	$2.00e+02$	$1.61e+02$	$5.97e+01$
22	$5.05e+01$	$4.77e+02$	$2.68e+02$	$2.44e+02$	$1.07e+02$
23	$3.40e+02$	$1.23e+03$	$8.34e+02$	$8.35e+02$	$1.89e+02$
24	$1.07e+02$	$1.32e+02$	$1.18e+02$	$1.19e+02$	$5.63e+00$
25	$1.14e+02$	$2.17e+02$	$2.07e+02$	$1.93e+02$	$3.39e+01$
26	$1.04e+02$	$2.00e+02$	$2.00e+02$	$1.61e+02$	$4.02e+01$
27	$2.25e+02$	$4.00e+02$	$3.08e+02$	$3.13e+02$	$2.27e+01$
28	$1.74e-06$	$3.00e+02$	$3.00e+02$	$2.06e+02$	$1.06e+02$

TABLE IV. RESULTS FOR 30D

Func.	Best	Worst	Median	Mean	Std
1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
2	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
3	$0.00e+00$	$7.73e+04$	$9.65e-01$	$2.24e+03$	$1.09e+04$
4	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
6	$0.00e+00$	$9.78e-03$	$0.00e+00$	$6.94e-04$	$1.99e-03$
7	$1.03e+00$	$1.25e+02$	$3.79e+01$	$4.48e+01$	$2.93e+01$
8	$2.07e+01$	$2.10e+01$	$2.09e+01$	$2.09e+01$	$8.11e-02$
9	$1.99e+01$	$2.76e+01$	$2.38e+01$	$2.37e+01$	$1.93e+00$
10	$0.00e+00$	$2.46e-02$	$7.40e-03$	$8.31e-03$	$5.41e-03$
11	$1.49e+01$	$4.97e+01$	$2.39e+01$	$2.54e+01$	$6.30e+00$
12	$2.79e+01$	$1.61e+02$	$5.77e+01$	$7.94e+01$	$4.35e+01$
13	$5.57e+01$	$2.41e+02$	$1.45e+02$	$1.56e+02$	$5.36e+01$
14	$3.27e+02$	$1.30e+03$	$8.08e+02$	$7.92e+02$	$2.19e+02$
15	$1.90e+03$	$4.60e+03$	$3.12e+03$	$3.13e+03$	$4.53e+02$
16	$2.28e-02$	$3.17e-01$	$9.03e-02$	$1.07e-01$	$6.71e-02$
17	$4.00e+01$	$6.71e+01$	$5.49e+01$	$5.50e+01$	$5.19e+00$
18	$1.34e+02$	$2.44e+02$	$1.89e+02$	$1.89e+02$	$2.71e+01$
19	$1.54e+00$	$4.21e+00$	$2.71e+00$	$2.80e+00$	$6.35e-01$
20	$1.23e+01$	$1.45e+01$	$1.45e+01$	$1.43e+01$	$5.69e-01$
21	$1.00e+02$	$3.00e+02$	$2.00e+02$	$1.86e+02$	$3.97e+01$
22	$6.34e+02$	$1.74e+03$	$1.18e+03$	$1.17e+03$	$2.90e+02$
23	$2.71e+03$	$5.35e+03$	$4.01e+03$	$4.03e+03$	$5.38e+02$
24	$1.97e+02$	$2.80e+02$	$2.65e+02$	$2.59e+02$	$1.74e+01$
25	$2.59e+02$	$3.01e+02$	$2.82e+02$	$2.82e+02$	$8.41e+00$
26	$1.32e+02$	$2.00e+02$	$2.00e+02$	$1.97e+02$	$1.20e+01$
27	$4.00e+02$	$1.01e+03$	$8.15e+02$	$7.49e+02$	$1.85e+02$
28	$1.00e+02$	$9.57e+03$	$3.00e+02$	$5.39e+02$	$1.32e+03$

ACKNOWLEDGMENT

INCAS[3] is co-funded by the Province of Drenthe, the Municipality of Assen, the European Fund for Regional Development and the Ministry of Economic Affairs, Peaks in the Delta. This research is supported by the Academy of Finland, Akatemiatutkija 130600, "Algorithmic design issues in Memetic Computing" and and Tutkijatohtori 140487, "Algorithmic Design and Software Implementation: a Novel Optimization Platform". The numerical experiments have been

TABLE V. RESULTS FOR 50D

Func.	Best	Worst	Median	Mean	Std
1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
2	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
3	$9.56e-05$	$3.96e+06$	$8.55e+03$	$2.83e+05$	$7.80e+05$
4	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
5	$1.29e-08$	$7.64e-08$	$3.26e-08$	$3.54e-08$	$1.44e-08$
6	$0.00e+00$	$4.34e+01$	$3.52e+00$	$9.51e+00$	$1.41e+01$
7	$1.39e+01$	$1.09e+02$	$4.50e+01$	$4.81e+01$	$2.16e+01$
8	$2.09e+01$	$2.12e+01$	$2.11e+01$	$2.10e+01$	$6.02e-02$
9	$4.21e+01$	$5.37e+01$	$4.69e+01$	$4.72e+01$	$2.76e+00$
10	$0.00e+00$	$2.71e-02$	$9.86e-03$	$8.60e-03$	$5.86e-03$
11	$2.89e+01$	$9.55e+01$	$5.07e+01$	$5.36e+01$	$1.31e+01$
12	$7.46e+01$	$4.28e+02$	$2.91e+02$	$2.66e+02$	$1.02e+02$
13	$2.04e+02$	$5.94e+02$	$4.70e+02$	$4.56e+02$	$8.54e+01$
14	$8.13e+02$	$2.40e+03$	$1.49e+03$	$1.49e+03$	$3.07e+02$
15	$4.60e+03$	$8.73e+03$	$6.38e+03$	$6.30e+03$	$6.74e+02$
16	$2.37e-02$	$2.02e-01$	$7.76e-02$	$8.66e-02$	$4.20e-02$
17	$8.39e+01$	$1.24e+02$	$1.01e+02$	$1.02e+02$	$1.04e+01$
18	$2.89e+02$	$5.43e+02$	$4.18e+02$	$4.18e+02$	$5.24e+01$
19	$2.97e+00$	$6.86e+00$	$4.88e+00$	$5.04e+00$	$9.33e-01$
20	$2.11e+01$	$2.45e+01$	$2.45e+01$	$2.43e+01$	$5.49e-01$
21	$1.00e+02$	$8.36e+02$	$2.00e+02$	$2.85e+02$	$2.20e+02$
22	$1.48e+03$	$3.34e+03$	$2.36e+03$	$2.39e+03$	$3.67e+02$
23	$6.73e+03$	$1.19e+04$	$8.27e+03$	$8.37e+03$	$1.01e+03$
24	$2.70e+02$	$3.50e+02$	$3.29e+02$	$3.22e+02$	$1.95e+01$
25	$3.45e+02$	$3.94e+02$	$3.67e+02$	$3.66e+02$	$9.94e+00$
26	$2.00e+02$	$2.00e+02$	$2.00e+02$	$2.00e+02$	$3.25e-02$
27	$4.00e+02$	$1.58e+03$	$1.24e+03$	$1.25e+03$	$2.08e+02$
28	$4.00e+02$	$7.25e+03$	$4.00e+02$	$1.24e+03$	$1.53e+03$

carried out on the computer network of the De Montfort University by means of the software for distributed optimization Kimeme [25].

REFERENCES

[1] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[2] D. Molina, M. Lozano, C. Garcia-Martinez, and F. Herrera, "Memetic Algorithms for Continuous Optimization Based on Local Search Chains," *Evolutionary Computation*, vol. 18, no. 1, pp. 27–63, 2010.

[3] M. A. Montes de Oca, T. Stutzle, M. Birattari, and M. Dorigo, "Frankenstein's PSO: A Composite Particle Swarm Optimization Algorithm," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 5, pp. 1120–1132, 2009.

[4] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[5] J. A. Vrugt, B. A. Robinson, and J. M. Hyman, "Self-Adaptive Multimethod Search for Global Optimization in Real-Parameter Spaces," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 243–259, 2009.

[6] F. Peng, K. Tang, G. Chen, and X. Yao, "Population-Based Algorithm Portfolios for Numerical Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 782–800, 2010.

[7] G. Iacca, F. Neri, E. Mininno, Y. S. Ong, and M. H. Lim, "Ockham's Razor in Memetic Computing: Three Stage Optimal Memetic Exploration," *Information Sciences*, vol. 188, pp. 17–43, 2012.

[8] F. Caraffini, F. Neri, G. Iacca, and A. Mol, "Parallel memetic structures," *Information Sciences*, vol. 227, pp. 60–82, 2013.

[9] A. Caponio, F. Neri, and V. Tirronen, "Super-fit control adaptation in memetic differential evolution frameworks," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 13, pp. 811–831, 2009.

[10] G. Iacca, R. Mallipeddi, E. Mininno, F. Neri, and P. N. Suganthan, "Super-fit and Population Size Reduction Mechanisms in Compact Differential Evolution," in *Proceedings of IEEE Symposium on Memetic Computing*, 2011, pp. 21–28.

[11] N. Hansen and A. Ostermeier, "Adapting arbitrary normal mutation distributions in evolution strategies: The covariance matrix adaptation," in *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1996, pp. 312–317.

[12] ——, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[13] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the Time Complexity of the Derandomized Evolution Strategy with Covariance Matrix Adaptation (CMA-ES)," *Evolutionary Computation*, vol. 11, no. 1, pp. 1–18, 2003.

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

[15] I. Rechenberg, "Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution," Ph.D. dissertation, Technical University of Berlin, 1971.

[16] A. Auger and N. Hansen, "A Restart CMA Evolution Strategy With Increasing Population Size," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2005, pp. 1769–1776.

[17] S. Ghosh, S. Roy, S. Islam, S. Das, and P. Suganthan, "A differential covariance matrix adaptation evolutionary algorithm for global optimization," in *Differential Evolution (SDE), 2011 IEEE Symposium on*, april 2011, pp. 1 –8.

[18] C. Igel, T. Suttorp, and N. Hansen, "A Computational Efficient Covariance Matrix Update and a (1+1)-CMA for Evolution Strategies," in *Proceedings of the Genetic and Evolutionary Computation Conference*. ACM Press, 2006, pp. 453–460.

[19] W. Dong and X. Yao, "Covariance matrix repairing in Gaussian based EDAs," in *IEEE Congress on Evolutionary Computation*, 2007, pp. 415–422.

[20] F. Neri and V. Tirronen, "Recent Advances in Differential Evolution: A Review and Experimental Analysis," *Artificial Intelligence Review*, vol. 33, no. 1–2, pp. 61–106, 2010.

[21] F. Neri, G. Iacca, and E. Mininno, "Disturbed Exploitation compact Differential Evolution for Limited Memory Optimization Problems," *Information Sciences*, vol. 181, no. 12, pp. 2469–2487, 2011.

[22] L.-Y. Tseng and C. Chen, "Multiple trajectory search for Large Scale Global Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2008, pp. 3052–3059.

[23] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernndez-Daz, "Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization," Zhengzhou University and Nanyang Technological University, Zhengzhou China and Singapore, Tech. Rep. 201212, 2013.

[24] L. Sheng, *The Java Native Interface: Programmer's Guide and Specification (The Java Series)*. Addison-Wesley, 1999.

[25] Cyber Dyne Srl Home Page, "Kimeme," 2012, http://cyberdynesoft.it/.

[26] S. Islam, S. Das, S. Ghosh, S. Roy, and P. Suganthan, "An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 42, no. 2, pp. 482–500, april 2012.

[27] X. Li and X. Yao, "Cooperatively Coevolving Particle Swarms for Large Scale Optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 2, pp. 210–224, april 2012.

[28] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[29] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

[30] S. Garcia, A. Fernandez, J. Luengo, and F. Herrera, "A study of statistical techniques and performance measures for genetics-based machine learning: accuracy and interpretability," *Soft Computing*, vol. 13, no. 10, pp. 959–977, 2008.

[31] N. Hansen, "The CMA Evolution Strategy," 2012, http://www.lri.fr/~hansen/cmaesintro.html.

TABLE VI. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE=CMA-ES-RIS) FOR CMA-ES-RIS AGAINST STATE-OF-THE-ART ALGORITHMS IN 10D

	CMA-ES-RIS	MDE-pBX		CMA-ES		CCPSO2	
f_1	$\mathbf{0.00e+00 \pm 0.00e+00}$	$0.00e+00 \pm 0.00e+00$	=	$0.00e+00 \pm 0.00e+00$	=	$1.92e-04 \pm 1.16e-03$	+
f_2	$\mathbf{0.00e+00 \pm 0.00e+00}$	$4.15e+02 \pm 9.60e+02$	+	$0.00e+00 \pm 0.00e+00$	=	$9.93e+05 \pm 7.58e+05$	+
f_3	$7.04e-01 \pm 4.57e+00$	$4.96e+03 \pm 4.66e+04$	+	$\mathbf{5.69e-01 \pm 1.81e+00}$	=	$2.13e+07 \pm 3.13e+07$	+
f_4	$\mathbf{0.00e+00 \pm 0.00e+00}$	$6.50e-02 \pm 6.38e-01$	+	$0.00e+00 \pm 0.00e+00$	=	$8.80e+03 \pm 2.50e+03$	+
f_5	$0.00e+00 \pm 2.76e-14$	$0.00e+00 \pm 7.54e-14$	=	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$2.94e-03 \pm 8.06e-03$	+
f_6	$\mathbf{1.10e+00 \pm 2.85e+00}$	$6.18e+00 \pm 4.73e+00$	+	$6.74e+00 \pm 6.74e+00$	+	$1.52e+00 \pm 2.91e+00$	+
f_7	$5.33e+01 \pm 4.63e+01$	$\mathbf{5.63e+00 \pm 7.94e+00}$	-	$5.45e+08 \pm 5.38e+09$	=	$3.27e+01 \pm 8.31e+00$	=
f_8	$\mathbf{2.03e+01 \pm 1.36e-01}$	$2.05e+01 \pm 1.06e-01$	+	$2.03e+01 \pm 1.32e-01$	=	$2.04e+01 \pm 7.66e-02$	+
f_9	$3.59e+00 \pm 1.03e+00$	$\mathbf{2.37e+00 \pm 1.41e+00}$	-	$1.50e+01 \pm 3.65e+00$	+	$4.98e+00 \pm 9.13e-01$	+
f_{10}	$\mathbf{1.24e-02 \pm 1.33e-02}$	$1.25e-01 \pm 9.20e-02$	+	$1.33e-02 \pm 1.39e-02$	=	$1.47e+00 \pm 6.44e-01$	+
f_{11}	$3.57e+00 \pm 1.46e+00$	$2.48e+00 \pm 1.61e+00$	-	$2.31e+02 \pm 2.67e+02$	+	$\mathbf{1.97e+00 \pm 1.18e+00}$	-
f_{12}	$1.29e+01 \pm 5.36e+00$	$\mathbf{1.06e+01 \pm 4.76e+00}$	-	$3.49e+02 \pm 3.81e+02$	+	$2.64e+01 \pm 7.93e+00$	+
f_{13}	$2.56e+01 \pm 1.07e+01$	$\mathbf{2.01e+01 \pm 7.94e+00}$	-	$2.94e+02 \pm 3.99e+02$	+	$3.56e+01 \pm 8.30e+00$	+
f_{14}	$1.02e+02 \pm 7.32e+01$	$1.25e+02 \pm 1.12e+02$	=	$1.88e+03 \pm 4.25e+02$	+	$\mathbf{5.27e+01 \pm 3.86e+01}$	-
f_{15}	$\mathbf{6.17e+02 \pm 1.72e+02}$	$7.26e+02 \pm 2.61e+02$	+	$1.80e+03 \pm 3.92e+02$	+	$8.92e+02 \pm 2.18e+02$	+
f_{16}	$\mathbf{1.64e-01 \pm 7.49e-02}$	$5.43e-01 \pm 4.49e-01$	+	$4.51e-01 \pm 5.06e-01$	+	$1.18e+00 \pm 2.19e-01$	+
f_{17}	$\mathbf{1.04e+01 \pm 3.70e+00}$	$1.32e+01 \pm 1.85e+00$	+	$9.55e+02 \pm 3.42e+02$	+	$1.60e+01 \pm 2.07e+00$	+
f_{18}	$2.98e+01 \pm 6.10e+00$	$\mathbf{1.93e+01 \pm 4.67e+00}$	-	$9.01e+02 \pm 3.10e+02$	+	$5.41e+01 \pm 6.33e+00$	+
f_{19}	$8.14e-01 \pm 2.71e-01$	$\mathbf{6.44e-01 \pm 2.09e-01}$	-	$1.19e+00 \pm 5.00e-01$	+	$7.65e-01 \pm 2.52e-01$	=
f_{20}	$4.16e+00 \pm 3.95e-01$	$\mathbf{2.87e+00 \pm 5.25e-01}$	-	$4.68e+00 \pm 3.79e-01$	+	$3.50e+00 \pm 1.96e-01$	-
f_{21}	$\mathbf{1.61e+02 \pm 5.97e+01}$	$3.98e+02 \pm 1.99e+01$	+	$3.68e+02 \pm 8.71e+01$	+	$3.73e+02 \pm 6.75e+01$	+
f_{22}	$2.44e+02 \pm 1.07e+02$	$1.33e+02 \pm 1.04e+02$	-	$2.32e+03 \pm 4.25e+02$	+	$\mathbf{7.27e+01 \pm 4.98e+01}$	-
f_{23}	$\mathbf{8.35e+02 \pm 1.89e+02}$	$8.82e+02 \pm 3.08e+02$	=	$2.25e+03 \pm 4.48e+02$	+	$1.15e+03 \pm 2.58e+02$	+
f_{24}	$\mathbf{1.19e+02 \pm 5.63e+00}$	$2.02e+02 \pm 1.56e+01$	+	$3.83e+02 \pm 1.57e+02$	+	$2.01e+02 \pm 2.54e+01$	+
f_{25}	$\mathbf{1.93e+02 \pm 3.39e+01}$	$2.00e+02 \pm 1.23e+01$	+	$2.62e+02 \pm 4.92e+01$	+	$2.12e+02 \pm 1.09e+01$	+
f_{26}	$1.61e+02 \pm 4.02e+01$	$\mathbf{1.47e+02 \pm 4.36e+01}$	-	$2.57e+02 \pm 1.13e+02$	+	$1.58e+02 \pm 2.40e+01$	=
f_{27}	$3.13e+02 \pm 2.27e+01$	$\mathbf{3.06e+02 \pm 2.76e+01}$	-	$4.23e+02 \pm 1.35e+02$	+	$4.27e+02 \pm 4.91e+01$	+
f_{28}	$\mathbf{2.06e+02 \pm 1.06e+02}$	$3.07e+02 \pm 5.78e+01$	=	$1.21e+03 \pm 1.21e+03$	+	$3.01e+02 \pm 1.28e+02$	+

TABLE VII. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE=CMA-ES-RIS) FOR CMA-ES-RIS AGAINST STATE-OF-THE-ART ALGORITHMS IN 30D

	CMA-ES-RIS	MDE-pBX		CMA-ES		CCPSO2	
f_1	$0.00e+00 \pm 1.68e-13$	$0.00e+00 \pm 4.09e-13$	=	$\mathbf{0.00e+00 \pm 5.57e-14}$	=	$2.27e-13 \pm 1.21e-12$	+
f_2	$0.00e+00 \pm 1.83e-13$	$9.56e+04 \pm 6.16e+04$	+	$\mathbf{0.00e+00 \pm 7.88e-14}$	=	$9.95e+05 \pm 5.24e+05$	+
f_3	$2.24e+03 \pm 1.09e+04$	$1.80e+07 \pm 3.12e+07$	+	$\mathbf{1.53e+01 \pm 1.23e+02}$	-	$5.59e+08 \pm 5.57e+08$	+
f_4	$0.00e+00 \pm 1.68e-13$	$1.32e+01 \pm 5.46e+01$	+	$\mathbf{0.00e+00 \pm 5.08e-14}$	=	$5.57e+04 \pm 2.06e+04$	+
f_5	$2.88e-10 \pm 4.04e-10$	$1.14e-13 \pm 8.35e-13$	-	$\mathbf{1.14e-13 \pm 1.14e-14}$	-	$2.62e-08 \pm 6.05e-08$	+
f_6	$\mathbf{6.94e-04 \pm 1.99e-03}$	$1.99e+01 \pm 2.22e+01$	+	$3.05e+00 \pm 8.33e+00$	+	$2.19e+01 \pm 2.27e+01$	+
f_7	$\mathbf{4.48e+01 \pm 2.93e+01}$	$5.70e+01 \pm 1.77e+01$	+	$9.83e+03 \pm 6.44e+04$	=	$1.15e+02 \pm 3.11e+01$	+
f_8	$\mathbf{2.09e+01 \pm 8.11e-02}$	$2.11e+01 \pm 5.94e-02$	+	$2.09e+01 \pm 6.66e-02$	+	$2.10e+01 \pm 4.60e-02$	+
f_9	$2.37e+01 \pm 1.93e+00$	$\mathbf{2.22e+01 \pm 4.80e+00}$	-	$4.45e+01 \pm 6.99e+00$	+	$2.84e+01 \pm 2.08e+00$	+
f_{10}	$\mathbf{8.31e-03 \pm 5.41e-03}$	$1.64e-01 \pm 1.20e-01$	+	$1.73e-02 \pm 1.25e-02$	+	$1.48e-01 \pm 6.90e-02$	+
f_{11}	$2.54e+01 \pm 6.30e+00$	$4.62e+01 \pm 1.44e+01$	+	$9.47e+01 \pm 2.36e+02$	+	$\mathbf{1.19e-01 \pm 2.90e-01}$	-
f_{12}	$7.94e+01 \pm 4.35e+01$	$\mathbf{6.94e+01 \pm 2.00e+01}$	=	$7.11e+02 \pm 9.66e+02$	=	$2.12e+02 \pm 5.24e+01$	+
f_{13}	$1.56e+02 \pm 5.36e+01$	$\mathbf{1.49e+02 \pm 3.66e+01}$	=	$1.64e+03 \pm 1.66e+03$	+	$2.44e+02 \pm 3.44e+01$	+
f_{14}	$7.92e+02 \pm 2.19e+02$	$1.17e+03 \pm 3.95e+02$	+	$5.21e+03 \pm 7.37e+02$	+	$\mathbf{4.48e+00 \pm 2.87e+00}$	-
f_{15}	$\mathbf{3.13e+03 \pm 4.53e+02}$	$3.95e+03 \pm 6.57e+02$	+	$5.37e+03 \pm 6.73e+02$	+	$3.85e+03 \pm 4.52e+02$	+
f_{16}	$\mathbf{1.07e-01 \pm 6.71e-02}$	$1.25e+00 \pm 6.19e-01$	+	$1.26e-01 \pm 8.87e-02$	=	$2.16e+00 \pm 3.76e-01$	+
f_{17}	$5.50e+01 \pm 5.19e+00$	$7.05e+01 \pm 1.24e+01$	+	$3.95e+03 \pm 7.89e+02$	+	$\mathbf{3.07e+01 \pm 3.03e+00}$	-
f_{18}	$1.89e+02 \pm 2.71e+01$	$\mathbf{8.26e+01 \pm 1.89e+01}$	-	$4.23e+03 \pm 8.31e+02$	+	$2.31e+02 \pm 5.43e+01$	+
f_{19}	$2.80e+00 \pm 6.35e-01$	$9.54e+00 \pm 5.54e+00$	+	$3.66e+00 \pm 9.52e-01$	+	$\mathbf{7.77e-01 \pm 1.58e-01}$	-
f_{20}	$1.43e+01 \pm 5.69e-01$	$\mathbf{1.07e+01 \pm 7.75e-01}$	-	$1.50e+01 \pm 6.45e-02$	+	$1.35e+01 \pm 5.50e-01$	-
f_{21}	$\mathbf{1.86e+02 \pm 3.97e+01}$	$3.40e+02 \pm 7.62e+01$	+	$3.05e+02 \pm 9.01e+01$	+	$2.37e+02 \pm 6.71e+01$	+
f_{22}	$1.17e+03 \pm 2.90e+02$	$1.17e+03 \pm 4.92e+02$	=	$6.97e+03 \pm 1.06e+03$	+	$\mathbf{9.87e+01 \pm 6.70e+01}$	-
f_{23}	$\mathbf{4.03e+03 \pm 5.38e+02}$	$4.70e+03 \pm 7.70e+02$	+	$6.76e+03 \pm 6.82e+02$	+	$4.99e+03 \pm 6.31e+02$	+
f_{24}	$2.59e+02 \pm 1.74e+01$	$\mathbf{2.31e+02 \pm 8.60e+00}$	-	$8.19e+02 \pm 6.15e+02$	+	$2.80e+02 \pm 6.34e+00$	+
f_{25}	$2.82e+02 \pm 8.41e+00$	$\mathbf{2.79e+02 \pm 1.38e+01}$	-	$3.46e+02 \pm 1.45e+02$	+	$2.98e+02 \pm 6.94e+00$	+
f_{26}	$\mathbf{1.97e+02 \pm 1.20e+01}$	$2.26e+02 \pm 5.15e+01$	+	$5.51e+02 \pm 5.14e+02$	+	$2.00e+02 \pm 6.76e-01$	+
f_{27}	$7.49e+02 \pm 1.85e+02$	$\mathbf{6.50e+02 \pm 1.04e+02}$	-	$8.53e+02 \pm 2.42e+02$	=	$1.04e+03 \pm 8.09e+01$	+
f_{28}	$5.39e+02 \pm 1.32e+03$	$\mathbf{3.09e+02 \pm 1.50e+02}$	-	$1.96e+03 \pm 3.40e+03$	=	$4.35e+02 \pm 5.10e+02$	-

TABLE VIII. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REFERENCE=CMA-ES-RIS) FOR CMA-ES-RIS AGAINST STATE-OF-THE-ART ALGORITHMS IN 50D

	CMA-ES-RIS	MDE-pBX		CMA-ES		CCPSO2	
f_1	$\mathbf{2.27e-13 \pm 0.00e+00}$	$1.11e-11 \pm 7.68e-11$	+	$2.27e-13 \pm 0.00e+00$	=	$6.82e-13 \pm 1.31e-12$	+
f_2	$\mathbf{2.27e-13 \pm 0.00e+00}$	$4.37e+05 \pm 1.64e+05$	+	$2.27e-13 \pm 0.00e+00$	=	$1.85e+06 \pm 9.33e+05$	+
f_3	$2.83e+05 \pm 7.80e+05$	$8.45e+07 \pm 1.46e+08$	+	$\mathbf{9.64e+02 \pm 4.49e+03}$	-	$1.98e+09 \pm 2.09e+09$	+
f_4	$\mathbf{2.27e-13 \pm 0.00e+00}$	$3.05e+01 \pm 6.62e+01$	+	$2.27e-13 \pm 0.00e+00$	=	$1.00e+05 \pm 3.57e+04$	+
f_5	$3.54e-08 \pm 1.44e-08$	$1.93e-11 \pm 9.76e-11$	-	$\mathbf{6.82e-13 \pm 1.57e-12}$	-	$2.70e-10 \pm 8.03e-10$	-
f_6	$\mathbf{9.51e+00 \pm 1.41e+01}$	$5.48e+01 \pm 2.12e+01$	+	$4.32e+01 \pm 7.10e+00$	+	$4.35e+01 \pm 1.36e+01$	+
f_7	$4.81e+01 \pm 2.16e+01$	$6.59e+01 \pm 1.06e+01$	+	$\mathbf{4.19e+01 \pm 1.70e+01}$	=	$1.37e+02 \pm 2.31e+01$	+
f_8	$\mathbf{2.10e+01 \pm 6.02e-02}$	$2.12e+01 \pm 4.44e-02$	+	$2.11e+01 \pm 9.75e-02$	+	$2.11e+01 \pm 4.49e-02$	+
f_9	$4.72e+01 \pm 2.76e+00$	$\mathbf{4.32e+01 \pm 7.71e+00}$	-	$7.66e+01 \pm 7.77e+00$	+	$5.79e+01 \pm 4.39e+00$	+
f_{10}	$\mathbf{8.60e-03 \pm 5.86e-03}$	$1.34e-01 \pm 1.23e-01$	+	$2.24e-02 \pm 1.49e-02$	+	$1.24e-01 \pm 4.62e-02$	+
f_{11}	$5.36e+01 \pm 1.31e+01$	$1.24e+02 \pm 2.87e+01$	+	$2.19e+02 \pm 4.56e+02$	+	$\mathbf{4.31e-01 \pm 5.74e-01}$	-
f_{12}	$2.66e+02 \pm 1.02e+02$	$\mathbf{1.58e+02 \pm 3.25e+01}$	-	$2.25e+03 \pm 1.37e+03$	+	$4.46e+02 \pm 7.92e+01$	+
f_{13}	$4.56e+02 \pm 8.54e+01$	$\mathbf{3.24e+02 \pm 4.74e+01}$	-	$3.36e+03 \pm 1.09e+03$	+	$5.49e+02 \pm 6.67e+01$	+
f_{14}	$1.49e+03 \pm 3.07e+02$	$2.65e+03 \pm 8.86e+02$	+	$8.82e+03 \pm 1.04e+03$	+	$\mathbf{6.45e+00 \pm 3.20e+00}$	-
f_{15}	$\mathbf{6.30e+03 \pm 6.74e+02}$	$7.46e+03 \pm 7.95e+02$	+	$9.09e+03 \pm 9.43e+02$	+	$7.95e+03 \pm 7.11e+02$	+
f_{16}	$8.66e-02 \pm 4.20e-02$	$1.75e+00 \pm 7.40e-01$	+	$\mathbf{8.01e-02 \pm 4.72e-02}$	=	$2.39e+00 \pm 5.90e-01$	+
f_{17}	$1.02e+02 \pm 1.04e+01$	$1.75e+02 \pm 3.72e+01$	+	$6.97e+03 \pm 1.07e+03$	+	$\mathbf{5.14e+01 \pm 2.84e-01}$	-
f_{18}	$4.18e+02 \pm 5.24e+01$	$\mathbf{1.85e+02 \pm 3.40e+01}$	-	$7.08e+03 \pm 9.14e+02$	+	$4.94e+02 \pm 1.08e+02$	+
f_{19}	$5.04e+00 \pm 9.33e-01$	$4.25e+01 \pm 2.66e+01$	+	$6.32e+00 \pm 1.18e+00$	+	$\mathbf{1.40e+00 \pm 2.19e-01}$	-
f_{20}	$2.43e+01 \pm 5.49e-01$	$\mathbf{2.00e+01 \pm 9.04e-01}$	-	$2.50e+01 \pm 9.73e-02$	+	$2.28e+01 \pm 7.85e-01$	-
f_{21}	$\mathbf{2.85e+02 \pm 2.20e+02}$	$9.22e+02 \pm 3.06e+02$	+	$8.12e+02 \pm 3.73e+02$	+	$3.27e+02 \pm 2.64e+02$	=
f_{22}	$2.39e+03 \pm 3.67e+02$	$3.09e+03 \pm 9.98e+02$	+	$1.19e+04 \pm 1.26e+03$	+	$\mathbf{7.58e+01 \pm 8.58e+01}$	-
f_{23}	$\mathbf{8.37e+03 \pm 1.01e+03}$	$8.88e+03 \pm 1.20e+03$	+	$1.18e+04 \pm 8.52e+02$	+	$1.05e+04 \pm 1.11e+03$	+
f_{24}	$3.22e+02 \pm 1.95e+01$	$\mathbf{2.87e+02 \pm 1.47e+01}$	-	$1.64e+03 \pm 1.05e+03$	+	$3.56e+02 \pm 9.89e+00$	+
f_{25}	$\mathbf{3.66e+02 \pm 9.94e+00}$	$3.69e+02 \pm 1.78e+01$	=	$4.94e+02 \pm 1.88e+02$	+	$3.96e+02 \pm 1.19e+01$	+
f_{26}	$\mathbf{2.00e+02 \pm 3.25e-02}$	$3.50e+02 \pm 7.93e+01$	+	$6.04e+02 \pm 7.11e+02$	+	$2.09e+02 \pm 3.92e+01$	+
f_{27}	$1.25e+03 \pm 2.08e+02$	$\mathbf{1.24e+03 \pm 1.56e+02}$	=	$1.28e+03 \pm 2.51e+02$	=	$1.79e+03 \pm 8.78e+01$	+
f_{28}	$1.24e+03 \pm 1.53e+03$	$\mathbf{4.33e+02 \pm 3.26e+02}$	=	$3.27e+03 \pm 5.60e+03$	=	$6.33e+02 \pm 8.95e+02$	-

PVIII

A DIFFERENTIAL EVOLUTION FRAMEWORK WITH ENSEMBLE OF PARAMETERS AND STRATEGIES AND POOL OF LOCAL SEARCH ALGORITHMS

by

Iacca, Giovanni and Neri, Ferrante and Caraffini, Fabio and Suganthan, Ponnuthurai Nagaratnam 2014

Applications of Evolutionary Computation (EvoApplications), Lecture Notes in Computer Science, Volume 8602, pages 615-626

PIX

MULTI-STRATEGY COEVOLVING AGING PARTICLE OPTIMIZATION

by

Giovanni Iacca, Fabio Caraffini and Ferrante Neri 2014

International journal of neural systems, volume 24, number 1, pages 1450008(1)-1450008(19)

PX

A SEPARABILITY PROTOTYPE FOR AUTOMATIC MEMES WITH ADAPTIVE OPERATOR SELECTION

by

Michael G. Epitropakis, Fabio Caraffini, Ferrante Neri and Edmund K. Burke
2014

Applications of Evolutionary Computation (EvoApplications), Lecture Notes in Computer Science, Volume 8602, pages 615-626

A Separability Prototype for Automatic Memes with Adaptive Operator Selection

Michael G. Epitropakis*, Fabio Caraffini†, Ferrante Neri† and Edmund K. Burke*
*Computing Science and Mathematics, School of Natural Sciences, University of Stirling,
Stirling FK9 4LA, Scotland, United Kingdom
†Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University,
The Gateway, Leicester LE1 9BH, England, United Kingdom,
Email: {mge@cs.stir.ac.uk, fabio.caraffini@dmu.ac.uk, fneri@dmu.ac.uk, e.k.burke@stir.ac.uk}

***Abstract*—One of the main challenges in algorithmics in general, and in Memetic Computing, in particular, is the automatic design of search algorithms. A recent advance in this direction (in terms of continuous problems) is the development of a software prototype that builds up an algorithm based upon a problem analysis of its separability. This prototype has been called the Separability Prototype for Automatic Memes (SPAM). This article modifies the SPAM by incorporating within it an adaptive model used in hyper-heuristics for tackling optimization problems. This model, namely Adaptive Operator Selection (AOS), rewards at run time the most promising heuristics/memes so that they are more likely to be used in the following stages of the search process. The resulting framework, here referred to as SPAM-AOS, has been tested on various benchmark problems and compared with modern algorithms representing the-state-of-the-art of search for continuous problems. Numerical results show that the proposed SPAM-AOS is a promising framework that outperforms the original SPAM and other modern algorithms. Most importantly, this study shows how certain areas of Memetic Computing and Hyper-heuristics are very closely related topics and it also shows that their combination can lead to the development of powerful algorithmic frameworks.**

I. Introduction

The No Free Lunch Theorems (NFLTs) [1] state that, under certain hypotheses, the performance of any pair of algorithms A and B averaged over all possible problems is the same. The hypotheses of the NFLTs are that the algorithms are non-revisiting and that the decision space is discrete. The satisfaction of these hypotheses is often non-realistic. For example, Auger and Teytaud [2] show that NFLTs are not valid for continuous problems while Poli and Graff [3] show that the NFLTs are not valid in meta-spaces, i.e. the space of operators. Nonetheless, NFLTs has been an extremely important result for the computer science community as it changed the landscape of search methodology research. More specifically, until the 1990s, many researchers were attempting to define a "super-algorithm" that is an algorithm which is superior to all the other algorithms over all problems, see e.g. [4]. After the publication of the NFLTs, researchers had to radically change their way of thinking. The search algorithm became a domain specific solver. The problem can be seen as the starting point for an algorithmic design process whose outcome is a procedure that addresses the features of that given problem, see e.g. [5], [6], and [7].

Since a problem change would result in a new design (and thus human effort), some research has been oriented towards a design of "flexible" search algorithm, i.e. those algorithms that change their features and adapt to (usually slightly) diverse problems hence displaying a reasonable good performance on an array of problems without the need of a major human intervention from the algorithmic designer, see [8]. This flexibility is usually achieved by means of the use of multiple algorithmic operators and an adaptive system that coordinates these operators, see e.g. [9] and [10]. The main idea is that multiple diverse algorithmic operators compensate for each other with their search logics. In this way, the algorithmic framework that employs multiple operators can reliably and robustly tackle various problem features, thus adapting to a new problem. This idea is the backbone of two algorithmic philosophies, Hyper-heuristics and Memetic Computing.

A hyper-heuristic is an algorithm composed of multiple algorithms coordinated by another software component. Usually, the latter incorporates a machine learning technique and acts as a supervisor structure that learns which algorithms are the most suitable for a given problem. An extensive literature on this topic is available, for example, see [11] and [12]. More recently, graph colouring heuristics have been hybridized with a random ordering heuristic [13]. Extensive discussions about advances in hyper-heuristics are given in [14]–[17]. The most challenging part of the hyper-heuristic design is the logic behind the coordination of the algorithms. A classical approach consists of assigning a score and rewarding the most promising heuristics: the so called choice function [11]. Recently, a wide range of diverse approaches have been proposed. Many of these approaches are based on reinforcement learning in a stand alone and combined fashion, see for example [12], [18]–[20] and [21]. Reinforcement learning, making use of memory-based mechanisms has also been proposed [22]. Modern hyper-heuristics also make use of multi-agent operators, see [23] and [24]. A competition for hyper-heuristic development was held in 2011 based on a framework called HyFlex [25]. This competition generated several effective hyper-heuristic methods[1], see for example [26].

The concept of the hyper-heuristic is explored also under the name of algorithm portfolios where the emphasis is placed upon the diversity of the available algorithms that are available to compose the entire framework. Algorithms of this kind can be designed using heuristic rules for the coordination, see [27]

[1]See http://www.asap.cs.nott.ac.uk/external/chesc2011/

978-1-4799-4491-0/14/$31.00 ©2014 IEEE

and [28]. A famous portfolio platform oriented towards the solution of the propositional satisfiability problem is called SATzilla, see [29] and [30]. Among the several studies that have been carried out on this platform, we highlight here the work on the automatic coordination system in [31] and on the prediction of the algorithm run time [32].

Memetic Computing (MC) is an area that investigates algorithmic structures that include multiple interacting heterogeneous operators. The interaction (coordination) amongst operators allows the proper functioning of the entire framework, see [33] and [8]. Clearly, there is a major overlap between hyper-heuristics and many areas of MC. However, the two concepts can be distinguished for a philosophical point that has some important implications in the algorithmic implementation. While a hyper-heuristic can be seen as a list of search methods and a component performing their coordination, a MC algorithm can be viewed as a whole algorithm composed of many parts in an unspecified way. In addition, in a hyper-heuristic, a unit representing the external framework can be thought of as a 'whole' search method with a 'budget'. In contrast, the unit of a MC structure can be seen as an algorithmic operator, that is a tuple composed of search move and a selection logic, see [34] and [35]. In other words, the unit representing many MC structures can (as a generalized statement) be viewed as being much 'smaller' than the unit composing a hyper-heuristic.

In many areas of MC, as well as for hyper-heuristics, the coordination between operators is the hardest and most challenging aspect in guaranteeing a good algorithmic performance. An example of meme coordination within memetic frameworks is meta-Lamarckian learning, [36], where the local search components are rewarded on the basis of their success history. In [37] [38], and [39], the coordination of the operators relies on evolution. The operators are encoded within the solution (self-adaptation) or in in a parallel population linked to the solutions (co-evolution). The operators related to the fittest individuals are selected for the subsequent generations. Another way to perform the coordination of the operators is by means of a control on the population diversity or its estimates [40]–[42].

All these adaptive optimization algorithms coordinate the sub-algorithms at runtime on the basis of feedback given by the algorithmic functioning. In most cases, this feedback is based on the success of the sub-algorithm/local search/algorithmic operator. Recent studies in MC propose a different logic: the algorithm is automatically built up on the basis of an analysis carried out on the problem and aiming at extracting its features. These studies at first focused on elementary structures, namely sequential and parallel [43], [44]. Subsequently a first working prototype of automatic design based on problem analysis restricted to the separability, namely Separability Prototype for Automatic Memes (SPAM), was proposed in [35].

This paper, whilst considering the algorithmic operators in [35], proposes to coordinate them by means of a hyper-heuristic rule. More specifically, we adopt the general concept of adaptive operator selection [18], [45]–[48] and incorporate it in the core structure of the SPAM algorithm. In this way, the resulting framework has the structure and operators of SPAM but, at run time, the selection of the components is intended to adapt progressively , on the basis of a success criterion, to the requirements of the problem.

The remainder of this paper is structured in the following way. Section II briefly illustrates the logic and the structure of the SPAM. Section III describes the proposed modified SPAM with hyper-heuristic coordination of the operators. Section IV displays the comparative results and the performance of the proposed algorithm with respect to the state-of-the-art. Section V gives the conclusions of this work.

II. Separability Prototype for Automatic Memes

The SPAM algorithm is a prototype of a large and ambitious project, that is a software platform for the automatic design of search methods in continuous optimization. This platform, currently under development, will analyze a given problem, in order to extract its features. These features will then be used to design the algorithm by selecting, combining, and linking the suitable algorithmic operators.

Before entering into the implementation details, let us define the notation used in this paper. Without loss of generality, we will refer to the minimization problem of an objective function (or fitness) $f(\mathbf{x})$, where the candidate solution $\mathbf{x}$ is a vector of n design variables (or genes) in a decision space $\mathbf{D}$. Thus, the optimization problem considered in this paper consists of the detection of that solution $\mathbf{x}^* \in \mathbf{D}$ such that $f(\mathbf{x}^*) < f(\mathbf{x})$, and this is valid $\forall \mathbf{x} \in \mathbf{D}$. Array variables are highlighted in bold face throughout this paper.

The SPAM algorithm is composed of two local search operators connected in parallel, see [44]. The first operator, here indicated with S, is a local search algorithm which deals with a single solution along its n axes, i.e. it separately perturbs each design variable. This meme can be viewed as a fairly straightforward hill-descent algorithm. It was previously used in [49] within a memetic structure.

The S implementation requires a generic input solution $\mathbf{x}$ and a trial solution $\mathbf{x^t}$. S perturbs the candidate by computing, for each coordinate i (each gene), $x_i^t = x_i - \xi$, where ξ is the exploratory radius. Subsequently, if $\mathbf{x^t}$ outperforms $\mathbf{x}$, the solution $\mathbf{x}$ is updated (the values of $\mathbf{x^t}$ are copied in it), otherwise a half step in the opposite direction is performed: $x_i^t = x_i + \frac{\xi}{2}$. Again, $\mathbf{x^t}$ replaces $\mathbf{x}$ if it outperforms it. If there is no update, then the exploration was unsuccessful. In this case, the radius ξ is halved. This is repeated for all the design variables. It has to be noted that the initial value of radius ξ is usually fixed and proportionate to the range of optimization space bounds (here ξ in fixed to 0.4). For the sake of clarity, Algorithm 1 describes the working principles of the S operator.

The second operator is the Rosenbrock algorithm (R) [50]. This operator, has been shown to always converge to a local optimum, under specific conditions [51]. At the beginning of the process, R is similar to S as it explores each of the n directions, by perturbing the input solution $\mathbf{x}$ with an initial step size vector $\mathbf{h}$. A matrix $\mathbf{A}$ is initialized as the identity matrix. While ever new improvements are found, for $j = 1, 2, \ldots, n$, a new trial point $\mathbf{x^t}$ is generated by perturbing the i^{th} design variable of solution $\mathbf{x}$ in the following way:

$$x_i^t = x_i + h_j \cdot A_{i,j} \tag{1}$$

Algorithm 1 Pseudo-code of the S operator.

```
INPUT x
while condition on the local computational budget do
    for i = 1 : n do
        x_i^t = x_i − ξ
        if f(x^t) ≤ f(x) then
            x = x^t
        else
            x_i^t = x_i + ξ/2
            if f(x^t) ≤ f(x) then
                x = x^t
            end if
        end if
    end for
    if x == x^t then
        ξ = ξ/2
    end if
end while
OUTPUT x
```

for $i = 1, 2, \ldots, n$. When successful, $\mathbf{x}$ is updated and the step size is increased by a factor α ($h_j = \alpha \cdot h_j$), otherwise it is decreased by means of a factor β then the opposite direction will be considered ($h_j = -\beta \cdot h_j$). We repeat this procedure until an improvement of solution $\mathbf{x}$ is found. After every success has been determined and examined in each base direction, then the Gram-Schmidt orthogonalization procedure is used to rotate the coordinate system towards the approximated gradient. This operation yields an updated version of the matrix $\mathbf{A}$. After this, the step size vector $\mathbf{h}$ is reinitialized and the process is repeated, employing the rotated coordinate system, and changing the value of $\mathbf{x}$ according to Eq. (1). It is important to note that, the use of a rotated coordinate system means that the trial generation mechanism corresponds to a diagonal move by following the direction determined by the gradient. The termination criterion of R is determined by two conditions. The first criterion is based on the minimum element of the vector $\mathbf{h}$. The second criterion concerns the minimum difference between $\mathbf{x^t}$ and $\mathbf{x}$ design variables. In formal terms, R is run until the following condition is true: $\min(|\mathbf{h}|) > \varepsilon \; OR \; \min(|\mathbf{x^t} - \mathbf{x}|) > \varepsilon$, where $\min()$ is the minimum vector element. If there is no improvement at all, only the first condition is considered[2].

These two local search operators are clearly very diverse in terms of the search logic. Whilst S attempts to optimize the problem by perturbing the axes separately (moves along the axes), R attempts to follow the directions of the gradient, hence performing diagonal moves.

The SPAM is based on the idea that (due to their features), S is efficient for tackling separable problems whilst R should be used only when the problem is not separable. Furthermore, the separability of a problem is considered in [35] as a fuzzy property of optimization problems. More specifically, it is possible to consider a problem as being separable to a certain degree. In this light, a problem analyzer has been designed to extract the *degree of separability* of the given problem. The degree of separability is an index between 0 and 1 that results from the problem analysis phase. On the basis of this parameter, an activation probability is given to S and R. Thus, a fully separable problem (e.g. the sphere function) will be solved only by S activations. In contrast, in the case of a highly non separable problem, the entire budget or a large portion of it is devoted to R. In the case of intermediate features, the budget will be shared between the two local searchers.

The parameter representing the degree of separability is calculated, for each problem, at the beginning of the search, by letting the Covariance Matrix Adaptation Evolution Strategy (CMAES) [52] perform some optimization steps and then manipulate the estimated covariance matrix. More specifically, from the covariance matrix, the Pearson correlation matrix $|\rho|$ is computed, and its elements are averaged and normalized in order to extract the index that describes the separability of the problem. For implementation details, see [35].

III. The Proposed Framework

As discussed previously, a problem analysis might reveal useful information on selecting an appropriate operator for the problem at hand. However, the selection of the "most suitable" operator might be a very challenging task, especially in cases where the problem analysis is not insightful enough. To alleviate the problem of selecting the most "suitable" operator *a priori*, we incorporate an Adaptive Operator Selection (AOS) model in SPAM's structure to adaptively choose the most preferable operator at hand. This section briefly describes the main algorithmic concepts behind the proposed framework SPAM-AOS, (SPAM with an Adaptive Operator Selection model).

The SPAM-AOS framework incorporates two main concepts, a credit assignment module $\mathcal{C}$ and an adaptive operator selection model $\mathcal{M}$, or a selective hyper-heuristic. The main role of the former is to estimate the quality of the applied operators based on feedback from the search process, whilst the latter adopts the estimated qualities to adaptively select which search operator to apply at the current stage. Intuitively, the model will select the most successful operator based on the historical information that has been acquired by the search process until the current stage. Several operator selection models have been proposed in the literature which are inspired by similar concepts from different scientific fields. Each model exhibits different characteristics and dynamics that are based on the acquired feedback during the search process. The proposed framework is a general strategy that is able to adopt any of the aforementioned adaptive operator selection models. Representative examples of such models include: probability matching [47], [48], adaptive pursuit [47], [48], statistical based models like the multinomial distribution with history forgetting [45], [46], and reinforcement learning approaches [18], [53]. However, for simplicity (and due to space limitations), in this paper we incorporate the well-known and widely used Probability Matching model [47], [48]. Future work will include extensive comparisons between the most representative models.

The credit assignment module and the adaptive operator selection model used in this paper are elucidated in the following sections.

A. Credit assignment module

The main role of a credit assignment module is to assign a representative score, or credit, that rewards the most "suitable"

[2]Note that, the parameter values are fixed to $\alpha = 2, \beta = 0.5, \varepsilon = 10^{-5}$ and the initial values of h_j to $h_j = 0.1, j = 1, 2, \ldots, n$.

operator at the current search stage. For each available operator we adopt as credit the simple fitness improvement [18], [45] between the current solution x^p and the best solution ever found by the algorithm x^e (the elite solution). Thereby, the fitness improvement η_a of an operator a can be easily calculated according to:

$$\eta_a = |f(x^p) - f(x^e)| \tag{2}$$

Notice that the credit equals zero, if no improvement is achieved.

In general, various different reward approaches can be used at this point, such as the latest reward (instantaneous), the average or a ranked-based reward [18]. However some of them tend to be unstable and noisy estimations of credit due to the stochastic nature of the search process. To alleviate this drawback, we estimate the empirical quality (or credit) of an operator by utilizing the average value of a sliding window of its latest w rewards. In detail, suppose that we have a pool of κ available operators, $A = \{a_1, a_2, \ldots, a_\kappa\}$; let S_i be a set of the latest w fitness improvement rewards (η_{a_i}) achieved by the operator a_i during the time step t of the algorithm. We adopt as the final credit assignment value $r_{a_i}(t)$, the average reward of the sliding window S_{a_i}, which can be calculated according to:

$$r_{a_i}(t) = \frac{\sum_{j=1}^{|S_{a_i}|} S_{a_i}(j)}{|S_{a_i}|} \tag{3}$$

where $|S_{a_i}|$ denotes the cardinality of the set S_{a_i}.

B. Adaptive Operator Selection model: Probability Matching

We utilize, as the adaptive operator, selection model the simple and well-known Probability Matching (PM) [18], [47], [48]. Intuitively, PM updates the selection probability of a specific operator proportionally to its empirical quality with respect to the others. In detail, suppose we have a set of κ available operators $A = \{a_1, a_2, \ldots, a_\kappa\}$ and a selection probability vector $P(t) = \{p_1(t), p_2(t), \ldots, p_\kappa(t)\}$. Initially, all operators have equal probability of being selected ($p_i = 1/\kappa, \forall i \in \{1, 2, \ldots, \kappa\}$). After the application of an operator on a solution vector, a reward ($r_{a_i}(t)$) is calculated based on the credit assignment module. The environment in which the adaptation procedure operates is non-stationary and the estimation of the empirical quality can be more reliable if the newest rewards influence the empirical quality more than the older ones. Thereby, PM estimates the empirical quality $q_{a_i}(t)$ of the operator a_i according to the following relaxation mechanism:

$$q_{a_i}(t+1) = q_{a_i}(t) + \gamma(r_{a_i}(t) - q_{a_i}(t)) \tag{4}$$

where $\gamma \in (0, 1]$ is the adaptation rate (γ is fixed to 0.1) [18], [47], [48].

Based on the quality estimate value of the operator a_i, PM updates its selection probability (p_{a_i}) according to the following formula:

$$p_{a_i}(t+1) = p_{\min} + (1 - \kappa \cdot p_{\min}) \frac{q_{a_i}(t+1)}{\sum_{i=1}^{\kappa} q_{a_i}(t+1)} \tag{5}$$

where $p_{\min} \in [0, 1]$ denotes the minimal probability value of each operator, to ensure that the chances of applying each operator will not vanish [47], [48]. Having calculated the probabilities of all operators in the pool, we employ a stochastic selection procedure (a simple roulette wheel) to select which operator to apply.

Algorithm 2 The general SPAM-AOS algorithmic scheme

1: Perform problem analysis as in [35] and calculate x^e.
2: INPUT: an initial solution x^e, a credit assignment module $\mathcal{C}$, and an AOS model $\mathcal{M}$.
3: Initialize the AOS model $\mathcal{M}$ and the credit assignment module $\mathcal{C}$.
4: $x^p = x^e$
5: **while** the remaining budget is available **do**
6: **Select** k_{str} **based on the AOS model** $\mathcal{M}$
7: **if** $k_{str} = 1$ **then**
8: Apply the S operator to x^p
9: **else**
10: Apply the R operator to x^p
11: **end if**
12: **Update credit assignment module** $\mathcal{C}$ based on the **fitness improvement** of x^e, x^p. (based on Eqs. (2), (3))
13: **Update the AOS model** $\mathcal{M}$ (based on Eqs. (4), (5))
14: **if** the operator succeeded at improving upon x^e performance **then**
15: $x^e = x^p$
16: **end if**
17: **if** the operator failed at improving upon x^e performance **and** the selected operator is the same that failed **then**
18: Perturb x^p according to an exponential crossover perturbation [35].
19: **if** x^p has a better performance against x^e **then**
20: $x^e = x^p$
21: **end if**
22: **end if**
23: **end while**

C. SPAM with an adaptive operator selection model

Having defined the SPAM algorithm and the two main concepts of the proposed framework, we proceed with the description of the proposed approach.

More specifically, the algorithmic framework of SPAM-AOS is strongly based on SPAM's structure. The proposed general algorithmic framework is briefly demonstrated in Algorithm 2. Initially the separability problem analysis is performed (line 1 of Alg. 2) and the best located solution or *elite* solution, x^e, is used as the initial point of the algorithm (line 2 of Alg. 2). As previously described, the separability problem analysis involves the application of the CMAES algorithm for a limited budget of function evaluations. Next, an initialization process of the credit assignment module $\mathcal{C}$ and the AOS model $\mathcal{M}$ is performed, if necessary (line 3 of Alg. 2). $\mathcal{C}$ initializes the data structures for the sliding windows and $\mathcal{M}$ assigns equal probabilities to all the used operators. For each step of the algorithm, we employ an AOS model $\mathcal{M}$ to select which operator k_{str} to apply on the x^p solution. In the general case, we might have a pool of κ available search operators, i.e., $k_{str} \in \{1, 2, \ldots, \kappa\}$. As in SPAM, we incorporate the S and R operators ($\kappa = 2$) and select one of them accordingly (lines 6–11 of Alg. 2). The current version of SPAM-AOS incorporates the PM model described previously, which stochastically selects k_{str}. After the application of the k_{str} operator, we score it based on the credit assignment module $\mathcal{C}$, which appropriately rewards the most suitable operator. Here, we utilize a reward based on the fitness improvement between the resulting solution x^p and the elite solution x^e, (line 12 of Alg. 2). Having calculated the reward of the applied operator, we update its quality estimation value and the corresponding

selection probability based on Eqs. (4), (5) (line 13 of Alg. 2). The remaining steps of the algorithm are similar to SPAM (lines 14–22 of Alg. 2), i.e., we update the elite solution if we have improved it and we apply an exponential crossover perturbation strategy, if the applied operator failed to improve the elite solution x^e [35].

IV. Numerical Results

The proposed SPAM-AOS has been compared with the original SPAM algorithm [35], the CCPSO2 [54], and the MDE-pBX [55]. Their effectiveness is assessed on two benchmark suites that include scalable problems with different characteristics: the recently proposed CEC 2013 [56] (28 benchmark functions) and BBOB 2010 benchmark suites [57] (25 benchmark functions). In this paper, we consider the 10, 30 and 50-dimensional versions of the CEC 2013 suite and the 100-dimensional versions of the BBOB 2010 suite. A detailed description of the benchmarks can be found in the [56], [57]. It has to be noted that all considered algorithms use their default parameter settings [35], [54], [55], whilst the proposed SPAM-AOS utilizes its default parameter values as previously described.

Throughout this section, we adopt the experimental protocol used in [35]. Specifically, for each algorithm and each benchmark function, we conduct 100 independent runs. For each run, a budget of $maxFES = 5000 \cdot D$ function evaluations is employed, where D is the dimensionality of the problem. Tables I, II, III, and IV, show the results of our experiments in terms of final average error and corresponding standard deviation. A **boldface** font has been used to indicate the best performing algorithm, for each problem. To assess the statistical significance of the observed performance differences, for each problem and each algorithm we apply the non-parametric Wilcoxon rank sum test [58] between the corresponding algorithm and the proposed SPAM-AOS. The mark "+" denotes the cases where the null hypothesis is rejected at the 5% significance level and SPAM-AOS performs significantly better. Similarly, the mark "-" shows the rejection of the null hypothesis at the 5% level of significance and SPAM-AOS exhibits inferior performance. In the remaining cases, which we mark with "=", the null hypothesis is accepted which indicates that the performance of the algorithms being compared is statistically indistinguishable.

To this end, Tables I, II, III, and IV exhibit the extensive experimental results of all algorithms on the 10, 30, 50-dimensional CEC 2013 and on the 100-dimensional BBOB 2010 benchmark sets respectively. It can be clearly observed that, in the majority of the cases, SPAM-AOS either significantly outperforms the other algorithms or it behaves equally well. There are some cases where it produces an inferior performance (10-dimensional cases against the MDE-pBX algorithm), but this behavior radically changes as the dimensionality increases. Specifically, the impact of the adaptive operator selection approach on SPAM-AOS is evident since for the majority of the cases the performance gains are significantly better than the original SPAM algorithm. Similarly, comparing SPAM-AOS with CCPSO2, we can observe that SPAM-AOS exhibits superior performance in more than 60% of the tested cases. MDE-pBX is better than SPAM-AOS in the majority of the 10-dimensional CEC 2013 functions, but once more this behavior radically changes in the remaining experiments.

Finally, to have a general statistical sense of the significance of the algorithms across all problems and to alleviate the problem of having Type I errors in multiple comparisons with a higher probability, we employ the Holm-Bonferroni correction, see [59]. As such, Table V reports the rankings of the algorithm and the numerical results from the Holm-Bonferroni test. Clearly, the performance differences between SPAM-AOS and the other implemented algorithms are statistically significant.

TABLE V. Holm-Bonferroni procedure (reference: SPAM-AOS, Rank = 2.94e+00)

j	Optimizer	Rank	z_j	p_j	δ/j	Hypothesis
1	MDE-pBX	2.56e+00	-2.86e+00	2.13e-03	5.00e-02	Rejected
2	SPAM	2.25e+00	-5.10e+00	1.67e-07	2.50e-02	Rejected
3	CCPSO2	2.18e+00	-5.65e+00	8.14e-09	1.67e-02	Rejected

To summarize the performance obtained by the implemented algorithms, we provide a graphical illustration of their overall performance across all benchmark functions used in this suite of experiments. As such, we employ the Empirical Cumulative Distribution Function (ECDF) graph of the normalized performance. In detail, to be able to compare all algorithms on different benchmark problems we normalize their performance values per benchmark problem linearly in a common range, such as in $[0, 1]$. Note that, in the current setting, performance is measured by the optimal objective value found by an algorithm on one execution run. Thus, normalized performance values close to zero indicate best performance, whilst the performance becomes worse as the values increase to one. Having calculated the normalized performance values of an algorithm $\mathcal{A}$ over all benchmark functions, ECDF values can be measured according to $ECDF(x) = \frac{1}{n}\sum_{i=1}^{n} I_{(-\infty,x]}(x_i)$, where n is the number of benchmark functions times the number of independent executions of algorithm $\mathcal{A}$ per benchmark function and $I_{(-\infty,x]}(\cdot)$ is the indicator function which is equal to one if $x_i \leq x$, and to zero otherwise. Intuitively, an ECDF curve of an algorithm $\mathcal{A}$ depicts the empirical probability of observing a performance value y that is less than or equal to y. This enables a summarizing comparison between different algorithms, since higher ECDF values for the same normalized performance value indicate better performance.

Figure 1 illustrates the ECDF graph of the four implemented algorithms across all benchmark sets considered in this paper. It can be clearly observed that the SPAM-AOS algorithm exhibits great potential across all benchmark functions, since for the same performance value it always has higher ECDF values against the other algorithms. As such, it is more likely that it can provide better performance gains against the other three algorithms. Regarding the overall performance of the remaining algorithms, SPAM comes second since, for the lower normalized performance values, it exhibits higher cumulative frequencies against CCPSO2 and MDE-pBX. Similarly, MDE-pBX outperforms CCPSO2, since its ECDF curve values are always higher than the corresponding ECDF values of the CCPSO2 algorithm.

V. Conclusion

This paper proposes the integration of a technique normally used within hyper-heuristic frameworks within an MC

TABLE I. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REF.: SPAM-AOS) ON CEC2013 [56] IN 10 DIMENSIONS.

	SPAM-AOS	SPAM		CCPSO2		MDE-pBX	
f_1	$\mathbf{0.00e+00 \pm 0.00e+00}$	$0.00e+00 \pm 0.00e+00$	=	$3.08e-03 \pm 1.05e-02$	+	$0.00e+00 \pm 2.27e-14$	=
f_2	$0.00e+00 \pm 1.36e-13$	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$1.80e+06 \pm 1.21e+06$	+	$2.54e+03 \pm 5.07e+03$	+
f_3	$\mathbf{1.31e+00 \pm 3.50e+00}$	$1.08e+02 \pm 7.77e+02$	=	$7.41e+07 \pm 1.12e+08$	+	$1.41e+05 \pm 1.23e+06$	+
f_4	$\mathbf{0.00e+00 \pm 0.00e+00}$	$0.00e+00 \pm 0.00e+00$	=	$1.05e+04 \pm 2.69e+03$	+	$3.82e+00 \pm 3.15e+01$	+
f_5	$1.14e-13 \pm 6.63e-14$	$3.46e-10 \pm 1.34e-09$	+	$2.20e-02 \pm 6.13e-02$	+	$\mathbf{0.00e+00 \pm 7.01e-14}$	-
f_6	$\mathbf{4.11e+00 \pm 4.78e+00}$	$5.63e+00 \pm 4.78e+00$	+	$4.67e+00 \pm 7.85e+00$	+	$5.70e+00 \pm 4.83e+00$	=
f_7	$6.04e+01 \pm 6.04e+01$	$7.57e+10 \pm 7.18e+11$	=	$3.99e+01 \pm 1.26e+01$	=	$\mathbf{7.37e+00 \pm 1.02e+01}$	-
f_8	$\mathbf{2.04e+01 \pm 1.38e-01}$	$2.05e+01 \pm 1.22e-01$	+	$2.04e+01 \pm 7.48e-02$	+	$2.05e+01 \pm 9.69e-02$	+
f_9	$6.73e+00 \pm 1.72e+00$	$1.42e+01 \pm 4.69e+00$	+	$5.48e+00 \pm 8.99e-01$	-	$\mathbf{2.16e+00 \pm 1.39e+00}$	-
f_{10}	$1.48e-02 \pm 1.40e-02$	$\mathbf{1.47e-02 \pm 1.40e-02}$	=	$1.93e+00 \pm 9.27e-01$	+	$1.06e-01 \pm 8.03e-02$	+
f_{11}	$6.30e+00 \pm 2.86e+00$	$1.67e+01 \pm 7.81e+01$	=	$\mathbf{2.76e+00 \pm 1.85e+00}$	-	$2.89e+00 \pm 1.72e+00$	-
f_{12}	$1.80e+01 \pm 9.61e+00$	$1.27e+02 \pm 2.01e+02$	+	$3.39e+01 \pm 1.02e+01$	+	$\mathbf{1.02e+01 \pm 4.53e+00}$	-
f_{13}	$3.55e+01 \pm 1.52e+01$	$2.83e+02 \pm 4.49e+02$	+	$4.22e+01 \pm 8.88e+00$	+	$\mathbf{1.94e+01 \pm 8.85e+00}$	-
f_{14}	$2.18e+02 \pm 1.05e+02$	$7.63e+02 \pm 6.39e+02$	+	$\mathbf{8.67e+01 \pm 6.15e+01}$	-	$1.08e+02 \pm 9.77e+01$	-
f_{15}	$1.00e+03 \pm 3.65e+02$	$1.64e+03 \pm 4.51e+02$	+	$1.03e+03 \pm 2.70e+02$	=	$\mathbf{7.56e+02 \pm 2.63e+02}$	-
f_{16}	$\mathbf{2.92e-01 \pm 1.98e-01}$	$5.32e-01 \pm 6.39e-01$	+	$1.31e+00 \pm 2.35e-01$	+	$5.74e-01 \pm 4.62e-01$	+
f_{17}	$1.58e+01 \pm 4.28e+00$	$2.19e+02 \pm 4.34e+02$	+	$1.79e+01 \pm 2.64e+00$	+	$\mathbf{1.32e+01 \pm 1.92e+00}$	-
f_{18}	$4.17e+01 \pm 1.42e+01$	$7.78e+02 \pm 5.07e+02$	+	$5.82e+01 \pm 6.30e+00$	+	$\mathbf{2.02e+01 \pm 5.18e+00}$	-
f_{19}	$7.33e-01 \pm 3.44e-01$	$9.20e-01 \pm 3.30e-01$	+	$1.00e+00 \pm 3.69e-01$	+	$\mathbf{6.57e-01 \pm 2.22e-01}$	=
f_{20}	$4.11e+00 \pm 3.64e-01$	$4.45e+00 \pm 4.82e-01$	+	$3.59e+00 \pm 2.16e-01$	-	$\mathbf{2.73e+00 \pm 6.04e-01}$	-
f_{21}	$\mathbf{2.86e+02 \pm 1.26e+02}$	$3.25e+02 \pm 1.14e+02$	+	$3.68e+02 \pm 6.68e+01$	+	$3.98e+02 \pm 1.99e+01$	+
f_{22}	$3.34e+02 \pm 1.19e+02$	$1.19e+03 \pm 8.95e+02$	+	$\mathbf{1.23e+02 \pm 6.60e+01}$	-	$1.77e+02 \pm 1.37e+02$	-
f_{23}	$1.52e+03 \pm 4.16e+02$	$2.23e+03 \pm 5.07e+02$	+	$1.37e+03 \pm 2.82e+02$	-	$\mathbf{8.43e+02 \pm 3.48e+02}$	-
f_{24}	$\mathbf{1.93e+02 \pm 4.46e+01}$	$2.72e+02 \pm 1.43e+02$	+	$2.11e+02 \pm 1.80e+01$	=	$2.05e+02 \pm 5.21e+00$	+
f_{25}	$2.15e+02 \pm 1.90e+01$	$2.52e+02 \pm 7.08e+01$	+	$2.12e+02 \pm 1.46e+01$	-	$\mathbf{2.01e+02 \pm 8.24e+00}$	-
f_{26}	$1.67e+02 \pm 6.02e+01$	$2.51e+02 \pm 1.45e+02$	+	$1.71e+02 \pm 2.37e+01$	+	$\mathbf{1.40e+02 \pm 4.16e+01}$	-
f_{27}	$3.81e+02 \pm 7.32e+01$	$4.24e+02 \pm 2.72e+02$	=	$4.33e+02 \pm 5.71e+01$	+	$\mathbf{3.04e+02 \pm 1.72e+01}$	-
f_{28}	$\mathbf{2.92e+02 \pm 9.79e+01}$	$1.00e+03 \pm 1.14e+03$	+	$4.01e+02 \pm 1.63e+02$	+	$3.04e+02 \pm 5.53e+01$	+

TABLE II. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REF.: SPAM-AOS) ON CEC2013 [56] IN 30 DIMENSIONS.

	SPAM-AOS	SPAM		CCPSO2		MDE-pBX	
f_1	$0.00e+00 \pm 2.05e-13$	$\mathbf{0.00e+00 \pm 2.01e-13}$	=	$1.36e-12 \pm 6.01e-12$	+	$2.27e-13 \pm 4.86e-13$	+
f_2	$1.75e+03 \pm 1.90e+03$	$\mathbf{1.58e+03 \pm 1.50e+03}$	=	$2.14e+06 \pm 1.04e+06$	+	$2.70e+05 \pm 2.62e+05$	+
f_3	$\mathbf{7.88e+05 \pm 1.79e+06}$	$9.96e+05 \pm 1.94e+06$	=	$1.13e+09 \pm 1.18e+09$	+	$5.19e+07 \pm 1.18e+08$	+
f_4	$\mathbf{5.16e-04 \pm 4.54e-03}$	$4.05e-02 \pm 4.03e-01$	=	$5.64e+04 \pm 2.09e+04$	+	$3.49e+02 \pm 3.18e+02$	+
f_5	$\mathbf{1.14e-13 \pm 6.49e-13}$	$1.09e-07 \pm 1.00e-06$	+	$3.04e-07 \pm 8.74e-07$	+	$1.09e-10 \pm 1.00e-09$	+
f_6	$\mathbf{9.73e-02 \pm 4.74e-01}$	$4.80e-01 \pm 2.74e+00$	=	$3.44e+01 \pm 2.78e+01$	+	$3.41e+01 \pm 2.77e+01$	+
f_7	$\mathbf{4.00e+01 \pm 2.58e+01}$	$7.54e+04 \pm 7.50e+05$	=	$1.19e+02 \pm 2.33e+01$	+	$5.61e+01 \pm 1.90e+01$	+
f_8	$\mathbf{2.09e+01 \pm 7.13e-02}$	$2.10e+01 \pm 5.45e-02$	+	$2.10e+01 \pm 5.45e-02$	+	$2.10e+01 \pm 5.93e-02$	+
f_9	$3.00e+01 \pm 3.58e+00$	$3.16e+01 \pm 5.34e+00$	+	$3.02e+01 \pm 2.20e+00$	=	$\mathbf{2.16e+01 \pm 4.36e+00}$	-
f_{10}	$1.28e-02 \pm 7.93e-03$	$\mathbf{1.01e-02 \pm 5.30e-03}$	-	$2.00e-01 \pm 9.45e-02$	+	$1.81e-01 \pm 1.10e-01$	+
f_{11}	$2.94e+01 \pm 6.43e+00$	$2.50e+01 \pm 6.22e+00$	-	$\mathbf{5.76e-01 \pm 6.49e-01}$	-	$4.68e+01 \pm 1.54e+01$	+
f_{12}	$9.80e+01 \pm 6.27e+01$	$3.34e+02 \pm 6.51e+02$	=	$2.13e+02 \pm 5.62e+01$	+	$\mathbf{6.91e+01 \pm 2.20e+01}$	=
f_{13}	$1.99e+02 \pm 6.82e+01$	$5.18e+02 \pm 9.98e+02$	=	$2.58e+02 \pm 4.39e+01$	+	$\mathbf{1.50e+02 \pm 3.56e+01}$	-
f_{14}	$7.83e+02 \pm 2.03e+02$	$1.85e+03 \pm 1.67e+03$	+	$\mathbf{6.57e+00 \pm 3.69e+00}$	-	$1.20e+03 \pm 4.25e+02$	+
f_{15}	$4.78e+03 \pm 7.59e+02$	$4.63e+03 \pm 8.62e+02$	=	$4.03e+03 \pm 4.77e+02$	-	$\mathbf{4.01e+03 \pm 7.00e+02}$	-
f_{16}	$1.42e-01 \pm 1.23e-01$	$\mathbf{1.26e-01 \pm 6.29e-02}$	=	$2.40e+00 \pm 4.03e-01$	+	$1.32e+00 \pm 8.61e-01$	+
f_{17}	$5.71e+01 \pm 7.70e+00$	$2.69e+02 \pm 8.25e+02$	+	$\mathbf{3.13e+01 \pm 4.89e-01}$	-	$6.89e+01 \pm 1.24e+01$	+
f_{18}	$2.37e+02 \pm 5.44e+01$	$8.63e+02 \pm 1.37e+03$	+	$2.44e+02 \pm 5.78e+01$	=	$\mathbf{8.31e+01 \pm 1.66e+01}$	-
f_{19}	$2.64e+00 \pm 6.63e-01$	$2.76e+00 \pm 8.35e-01$	=	$\mathbf{8.55e-01 \pm 1.71e-01}$	-	$9.10e+00 \pm 4.94e+00$	+
f_{20}	$1.45e+01 \pm 5.14e-01$	$1.46e+01 \pm 4.86e-01$	+	$1.39e+01 \pm 4.52e-01$	-	$\mathbf{1.09e+01 \pm 7.97e-01}$	-
f_{21}	$2.40e+02 \pm 5.62e+01$	$\mathbf{2.35e+02 \pm 5.54e+01}$	=	$2.58e+02 \pm 7.21e+01$	+	$3.09e+02 \pm 7.63e+01$	+
f_{22}	$1.08e+03 \pm 3.02e+02$	$2.37e+03 \pm 2.02e+03$	+	$\mathbf{1.21e+02 \pm 7.28e+01}$	-	$1.11e+03 \pm 5.46e+02$	=
f_{23}	$6.05e+03 \pm 9.37e+02$	$5.95e+03 \pm 1.04e+03$	=	$5.26e+03 \pm 7.22e+02$	-	$\mathbf{4.47e+03 \pm 7.32e+02}$	-
f_{24}	$3.00e+02 \pm 1.88e+02$	$3.20e+02 \pm 2.59e+02$	=	$2.81e+02 \pm 1.08e+01$	-	$\mathbf{2.31e+02 \pm 1.11e+01}$	-
f_{25}	$2.94e+02 \pm 1.84e+01$	$2.95e+02 \pm 1.75e+01$	=	$3.03e+02 \pm 6.25e+00$	+	$\mathbf{2.75e+02 \pm 1.55e+01}$	-
f_{26}	$2.80e+02 \pm 8.66e+01$	$3.07e+02 \pm 2.44e+02$	=	$\mathbf{2.02e+02 \pm 4.53e+00}$	=	$2.16e+02 \pm 4.31e+01$	-
f_{27}	$8.27e+02 \pm 1.92e+02$	$8.63e+02 \pm 2.08e+02$	=	$1.07e+03 \pm 1.13e+02$	+	$\mathbf{6.55e+02 \pm 1.13e+02}$	-
f_{28}	$6.27e+02 \pm 1.36e+03$	$1.04e+03 \pm 2.30e+03$	+	$5.43e+02 \pm 5.77e+02$	-	$\mathbf{3.11e+02 \pm 1.11e+02}$	-

TABLE III. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REF.: SPAM-AOS) ON CEC2013 [56] IN 50 DIMENSIONS.

	SPAM-AOS	SPAM		CCPSO2		MDE-pBX	
f_1	$\mathbf{2.27e-13 \pm 0.00e+00}$	$2.27e-13 \pm 0.00e+00$	=	$7.05e-12 \pm 3.53e-11$	+	$3.32e-11 \pm 2.60e-10$	+
f_2	$2.66e+04 \pm 1.35e+04$	$\mathbf{2.64e+04 \pm 1.24e+04}$	=	$4.37e+06 \pm 2.29e+06$	+	$9.06e+05 \pm 4.90e+05$	+
f_3	$8.55e+06 \pm 1.43e+07$	$\mathbf{6.32e+06 \pm 1.11e+07}$	=	$3.09e+09 \pm 3.03e+09$	+	$1.42e+08 \pm 1.57e+08$	+
f_4	$5.59e+02 \pm 5.01e+02$	$\mathbf{4.79e+02 \pm 6.48e+02}$	-	$1.08e+05 \pm 3.86e+04$	+	$1.09e+03 \pm 8.33e+02$	+
f_5	$\mathbf{3.41e-13 \pm 1.05e-12}$	$3.81e-10 \pm 6.24e-10$	+	$3.92e-04 \pm 3.89e-03$	+	$2.54e-05 \pm 2.52e-04$	=
f_6	$\mathbf{2.74e+01 \pm 1.75e+01}$	$3.96e+01 \pm 1.34e+01$	+	$4.74e+01 \pm 1.34e+01$	+	$5.67e+01 \pm 2.24e+01$	+
f_7	$\mathbf{4.75e+01 \pm 2.38e+01}$	$4.83e+01 \pm 1.97e+01$	=	$1.43e+02 \pm 2.39e+01$	+	$6.81e+01 \pm 1.22e+01$	+
f_8	$2.11e+01 \pm 6.66e-02$	$\mathbf{2.11e+01 \pm 6.58e-02}$	=	$2.12e+01 \pm 3.86e-02$	+	$2.12e+01 \pm 4.36e-02$	+
f_9	$5.49e+01 \pm 5.03e+00$	$6.85e+01 \pm 1.12e+01$	+	$5.87e+01 \pm 3.26e+00$	+	$\mathbf{4.27e+01 \pm 6.99e+00}$	-
f_{10}	$\mathbf{1.24e-02 \pm 7.26e-03}$	$1.28e-02 \pm 8.01e-03$	=	$2.03e-01 \pm 1.80e-01$	+	$4.09e-01 \pm 5.57e-01$	+
f_{11}	$5.87e+01 \pm 1.05e+01$	$5.13e+01 \pm 8.62e+00$	-	$\mathbf{9.07e-01 \pm 8.53e-01}$	-	$1.21e+02 \pm 2.97e+01$	+
f_{12}	$2.98e+02 \pm 1.43e+02$	$3.18e+02 \pm 2.59e+02$	=	$4.55e+02 \pm 8.03e+01$	+	$\mathbf{1.62e+02 \pm 3.45e+01}$	-
f_{13}	$5.33e+02 \pm 1.12e+02$	$5.61e+02 \pm 2.38e+02$	=	$5.69e+02 \pm 8.18e+01$	+	$\mathbf{3.22e+02 \pm 5.39e+01}$	-
f_{14}	$1.41e+03 \pm 3.09e+02$	$3.09e+03 \pm 2.95e+03$	+	$\mathbf{7.35e+00 \pm 3.55e+00}$	-	$2.79e+03 \pm 8.06e+02$	+
f_{15}	$8.50e+03 \pm 1.09e+03$	$8.54e+03 \pm 1.05e+03$	=	$8.31e+03 \pm 8.71e+02$	=	$\mathbf{7.58e+03 \pm 8.01e+02}$	-
f_{16}	$\mathbf{8.36e-02 \pm 3.88e-02}$	$9.21e-02 \pm 4.43e-02$	=	$2.75e+00 \pm 5.96e-01$	+	$1.93e+00 \pm 8.76e-01$	+
f_{17}	$9.62e+01 \pm 1.11e+01$	$9.83e+01 \pm 7.83e+00$	=	$\mathbf{5.16e+01 \pm 3.28e-01}$	-	$1.79e+02 \pm 3.56e+01$	+
f_{18}	$5.37e+02 \pm 9.93e+01$	$1.86e+03 \pm 2.49e+03$	+	$4.87e+02 \pm 9.77e+01$	-	$\mathbf{1.86e+02 \pm 3.17e+01}$	-
f_{19}	$4.73e+00 \pm 8.92e-01$	$4.99e+00 \pm 1.09e+00$	=	$\mathbf{1.49e+00 \pm 2.32e-01}$	-	$3.94e+01 \pm 2.10e+01$	+
f_{20}	$2.44e+01 \pm 2.86e-01$	$2.44e+01 \pm 4.10e-01$	=	$2.33e+01 \pm 8.19e-01$	-	$\mathbf{2.01e+01 \pm 9.17e-01}$	-
f_{21}	$\mathbf{4.27e+02 \pm 3.30e+02}$	$5.00e+02 \pm 3.85e+02$	=	$4.42e+02 \pm 3.45e+02$	+	$8.91e+02 \pm 3.44e+02$	+
f_{22}	$2.06e+03 \pm 3.17e+02$	$3.99e+03 \pm 3.56e+03$	+	$\mathbf{1.11e+02 \pm 9.60e+01}$	-	$3.22e+03 \pm 1.06e+03$	+
f_{23}	$1.12e+04 \pm 1.27e+03$	$1.16e+04 \pm 1.25e+03$	+	$1.09e+04 \pm 1.34e+03$	=	$\mathbf{9.08e+03 \pm 1.05e+03}$	-
f_{24}	$3.62e+02 \pm 2.14e+02$	$1.20e+03 \pm 1.04e+03$	+	$3.60e+02 \pm 9.64e+00$	-	$\mathbf{2.88e+02 \pm 1.56e+01}$	-
f_{25}	$3.79e+02 \pm 1.99e+01$	$4.51e+02 \pm 1.27e+02$	+	$3.97e+02 \pm 1.08e+01$	+	$\mathbf{3.68e+02 \pm 1.48e+01}$	-
f_{26}	$3.08e+02 \pm 2.97e+02$	$5.15e+02 \pm 6.42e+02$	+	$\mathbf{2.15e+02 \pm 4.95e+01}$	-	$3.55e+02 \pm 7.46e+01$	+
f_{27}	$1.28e+03 \pm 2.33e+02$	$1.32e+03 \pm 3.32e+02$	=	$1.82e+03 \pm 8.56e+01$	+	$\mathbf{1.23e+03 \pm 1.49e+02}$	=
f_{28}	$1.63e+03 \pm 2.22e+03$	$3.40e+03 \pm 5.33e+03$	=	$7.24e+02 \pm 1.08e+03$	-	$\mathbf{5.05e+02 \pm 5.99e+02}$	=

algorithm for continuous optimization. The employed algorithmic structure and operators are the same as those used by a MC approach previously proposed in literature. The hyper-heuristic adaptive technique integrates, on the top of the original memetic logic, a success-based adaptation. The resulting framework appears to improve upon the original MC approach and is competitive with modern meta-heuristics. This study can be seen as an attempt to consider about algorithmic design from a metaphor-free perspective and, in the specific case, to show how some areas of MC and hyper-heuristics contain essentially very similar ideas (if not even the same idea).

TABLE IV. AVERAGE ERROR ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST (REF.: SPAM-AOS) ON BBOB2010 [57] IN 100 DIMENSIONS.

	SPAM-AOS	SPAM		CCPSO2		MDE-pBX	
f_1	$\mathbf{2.42e-13 \pm 2.12e-13}$	$2.42e-13 \pm 2.13e-13$	=	$4.12e-13 \pm 1.97e-13$	+	$8.12e-07 \pm 3.91e-06$	+
f_2	$\mathbf{1.71e-13 \pm 1.55e-13}$	$2.84e-13 \pm 1.92e-13$	+	$9.66e-13 \pm 1.77e-12$	+	$3.22e-02 \pm 2.86e-01$	+
f_3	$1.04e+02 \pm 1.70e+01$	$9.69e+01 \pm 1.51e+01$	-	$\mathbf{8.09e+00 \pm 8.40e+00}$	-	$5.06e+02 \pm 9.57e+01$	+
f_4	$1.37e+02 \pm 2.01e+01$	$1.34e+02 \pm 1.86e+01$	=	$\mathbf{2.24e+01 \pm 1.31e+01}$	-	$8.32e+02 \pm 1.28e+02$	+
f_5	$4.93e-08 \pm 2.91e-07$	$\mathbf{2.13e-11 \pm 8.26e-12}$	-	$2.41e-04 \pm 1.20e-03$	+	$7.53e+00 \pm 1.01e+01$	+
f_6	$6.15e-08 \pm 7.12e-08$	$\mathbf{1.79e-11 \pm 1.14e-11}$	-	$8.94e+01 \pm 4.02e+01$	+	$3.36e+01 \pm 2.62e+01$	+
f_7	$\mathbf{5.26e+01 \pm 1.40e+01}$	$5.32e+01 \pm 1.36e+01$	=	$3.45e+02 \pm 4.90e+01$	+	$2.79e+02 \pm 7.50e+01$	+
f_8	$3.66e+01 \pm 1.11e+01$	$\mathbf{3.49e+01 \pm 9.88e+00}$	-	$1.20e+02 \pm 3.42e+01$	+	$1.78e+02 \pm 6.58e+01$	+
f_9	$\mathbf{4.39e+01 \pm 7.26e+00}$	$4.43e+01 \pm 6.96e+00$	+	$1.06e+02 \pm 2.78e+01$	+	$1.29e+02 \pm 3.80e+01$	+
f_{10}	$5.27e+02 \pm 1.50e+02$	$\mathbf{4.83e+02 \pm 1.43e+02}$	=	$2.62e+04 \pm 6.64e+03$	+	$1.51e+04 \pm 6.24e+03$	+
f_{11}	$8.72e+01 \pm 3.08e+01$	$7.86e+01 \pm 2.53e+01$	=	$5.45e+02 \pm 1.95e+02$	+	$\mathbf{1.62e+01 \pm 6.78e+00}$	-
f_{12}	$5.29e-02 \pm 1.98e-01$	$\mathbf{2.61e-02 \pm 8.27e-02}$	=	$7.95e+00 \pm 1.20e+01$	+	$2.70e+01 \pm 1.29e+02$	+
f_{13}	$\mathbf{1.06e+00 \pm 1.24e+00}$	$1.31e+00 \pm 1.78e+00$	=	$3.16e+00 \pm 4.20e+00$	+	$4.80e+00 \pm 9.83e+00$	+
f_{14}	$5.07e-05 \pm 6.32e-06$	$\mathbf{3.17e-05 \pm 3.66e-06}$	-	$1.32e-03 \pm 2.34e-04$	+	$2.21e-03 \pm 1.70e-03$	+
f_{15}	$2.75e+02 \pm 4.48e+01$	$\mathbf{2.73e+02 \pm 4.25e+01}$	=	$1.33e+03 \pm 2.32e+02$	+	$6.53e+02 \pm 9.63e+01$	+
f_{16}	$\mathbf{2.31e+00 \pm 8.27e-01}$	$2.42e+00 \pm 7.82e-01$	=	$2.74e+01 \pm 4.27e+00$	+	$1.35e+01 \pm 3.23e+00$	+
f_{17}	$8.55e+00 \pm 4.71e+00$	$8.59e+00 \pm 4.52e+00$	=	$8.65e+00 \pm 1.62e+00$	=	$\mathbf{3.36e+00 \pm 4.02e-01}$	-
f_{18}	$1.84e+01 \pm 1.15e+01$	$1.95e+01 \pm 1.32e+01$	=	$3.30e+01 \pm 6.61e+00$	+	$\mathbf{1.19e+01 \pm 2.08e+00}$	-
f_{19}	$1.96e+00 \pm 4.27e-01$	$\mathbf{1.67e+00 \pm 2.95e-01}$	-	$7.90e+00 \pm 1.29e+00$	+	$2.26e+00 \pm 6.47e-01$	+
f_{20}	$1.14e+00 \pm 1.09e-01$	$1.25e+00 \pm 1.66e-01$	+	$\mathbf{4.95e-01 \pm 6.73e-02}$	-	$2.12e+00 \pm 9.26e-02$	+
f_{21}	$3.89e+00 \pm 4.11e+00$	$4.32e+00 \pm 6.61e+00$	=	$\mathbf{3.36e+00 \pm 3.38e+00}$	=	$3.79e+00 \pm 5.07e+00$	=
f_{22}	$7.94e+00 \pm 9.28e+00$	$\mathbf{4.93e+00 \pm 6.86e+00}$	=	$5.11e+00 \pm 5.83e+00$	-	$8.94e+00 \pm 8.84e+00$	+
f_{23}	$8.23e-01 \pm 3.93e-01$	$\mathbf{7.60e-01 \pm 3.66e-01}$	=	$2.52e+00 \pm 4.22e-01$	+	$2.48e+00 \pm 9.31e-01$	+
f_{24}	$\mathbf{3.10e+02 \pm 6.43e+01}$	$3.19e+02 \pm 5.82e+01$	=	$1.08e+03 \pm 1.53e+02$	+	$3.48e+02 \pm 5.00e+01$	+

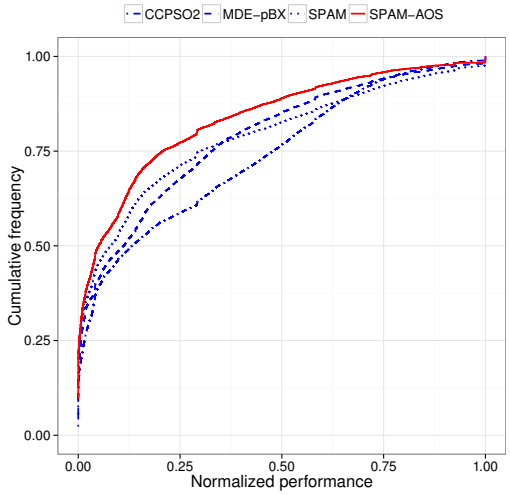


Fig. 1. The Empirical Cumulative Distribution Function graph of the normalized objective values for all algorithms, across all problem instances and runs. It can be clearly observed that SPAM-AOS always achieves higher cumulative frequency of lower objective values across all problem instances and runs (better performance gains).

ACKNOWLEDGMENTS

E.K. Burke and M.G. Epitropakis would like to thank EPSRC for their support for this work through grant EP/J017515/1. This research is also supported by the Academy of Finland, Akatemiatutkija 130600, "Algorithmic design issues in Memetic Computing".

REFERENCES

[1] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[2] A. Auger and O. Teytaud, "Continuous lunches are free!" in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*. ACM, 2007, pp. 916–922.

[3] R. Poli and M. Graff, "There is a free lunch for hyper-heuristics, genetic programming and computer scientists," in *EuroGP*, 2009, pp. 195–207.

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

[5] J. Du and R. Rada, "Memetic algorithms, domain knowledge, and financial investing," *Memetic Computing*, vol. 4, no. 2, pp. 109–125, 2012.

[6] J. Smith, "The co-evolution of memetic algorithms for protein structure prediction," in *Recent Advances in Memetic Algorithms*, ser. Studies in Fuzziness and Soft Computing, W. Hart, N. Krasnogor, and J. Smith, Eds. Springer, 2004, vol. 166, pp. 105–128.

[7] F. Neri and E. Mininno, "Memetic Compact Differential Evolution for Cartesian Robot Control," *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, pp. 54–65, 2010.

[8] F. Neri, C. Cotta, and P. Moscato, *Handbook of Memetic Algorithms*, ser. Studies in Computational Intelligence. Springer, 2011, vol. 379.

[9] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential Evolution Algorithm With Strategy Adaptation for Global Numerical Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[10] R. Mallipeddi, S. Mallipeddi, and P. N. Suganthan, "Ensemble strategies with adaptive evolutionary programming," *Information Sciences*, vol. 180, no. 9, pp. 1571–1581, 2010.

[11] P. Cowling, G. Kendall, and E. Soubeiga, "A hyperheuristic Approach to Scheduling a Sales Summit," in *Proceedings of the Third International Conference on Practice and Theory of Automated Timetabling*, ser. LNCS. Springer, 2000, vol. 2079, pp. 176–190.

[12] E. K. Burke, G. Kendall, and E. Soubeiga, "A Tabu Search hyperheuristic for Timetabling and Rostering," *Journal of Heuristics*, vol. 9, no. 6, pp. 451–470, 2003.

[13] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyperheuristic for educational timetabling problems," *European Journal of Operational Research*, vol. 176, pp. 177–192, 2007.

[14] E. Özcan, B. Bilgin, and E. E. Korkmaz, "A comprehensive analysis of hyper-heuristics," *Intelligent Data Analysis*, vol. 12, no. 1, pp. 3–23, 2008.

[15] E. K. Burke, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and J. R. Woodward, "A classification of hyper-heuristic approaches," in *Handbook of Meta-heuristics*, ser. International Series in Operations Research & Management Science, M. Gendreau and J.-Y. Potvin, Eds. Springer US, 2010, no. 146, pp. 449–468.

[16] E. K. Burke, G. Kendall, J. Newall, E. Hart, P. Ross, and S. Schulenburg, "Hyper-heuristics: An emerging direction in modern search technology," in *Handbook of Metaheuristics*, ser. International Series in Operations Research & Management Science, F. Glover and G. A. Kochenberger, Eds. Springer US, 2003, no. 57, pp. 457–474.

[17] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, "Hyper-heuristics: a survey of the state of the art," *Journal of the Operational Research Society*, vol. 64, no. 12, pp. 1695–1724, 2013.

[18] A. Fialho, "Adaptive operator selection for optimization," Ph.D. dissertation, Université Paris-Sud XI, Orsay, France, 2010.

[19] A. Fialho, L. D. Costa, M. Schoenauer, and M. Sebag, "Analyzing

bandit-based adaptive operator selection mechanisms," *Annals of Mathematics and Artificial Intelligence*, vol. 60, no. 1-2, pp. 25–64, 2010.

[20] J. Maturana, . Fialho, F. Saubion, M. Schoenauer, F. Lardeux, and M. Sebag, "Adaptive operator selection and management in evolutionary algorithms," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds. Springer Berlin Heidelberg, 2012, pp. 161–189.

[21] K. A. Dowsland, E. Soubeiga, and E. Burke, "A simulated annealing based hyperheuristic for determining shipper sizes for storage and transportation," *European Journal of Operational Research*, vol. 179, no. 3, pp. 759–774, 2007.

[22] E. Özcan, M. Misir, G. Ochoa, and E. K. Burke, "A reinforcement learning-great-deluge hyper-heuristic for examination timetabling," *International Journal of Applied Metaheuristic Computing*, vol. 1, no. 1, pp. 39–59, 2010.

[23] G. Acampora, M. Gaeta, and V. Loia, "Hierarchical optimization of personalized experiences for e-learning systems through evolutionary models," *Neural Computing and Applications*, vol. 20, no. 5, pp. 641–657, 2011.

[24] G. Acampora, J. M. Cadenas, V. Loia, and E. M. Ballester, "A multi-agent memetic system for human-based knowledge selection," *IEEE Transactions on Systems, Man, and Cybernetics, Part A*, vol. 41, no. 5, pp. 946–960, 2011.

[25] G. Ochoa, M. Hyde, T. Curtois, J. Vazquez-Rodriguez, J. Walker, M. Gendreau, G. Kendall, B. McCollum, A. Parkes, S. Petrovic, and E. K. Burke, "Hyflex: A benchmark framework for cross-domain heuristic search," in *Evolutionary Computation in Combinatorial Optimization*, ser. LNCS, J.-K. Hao and M. Middendorf, Eds. Springer Berlin Heidelberg, 2012, vol. 7245, pp. 136–147.

[26] M. Misir, K. Verbeeck, P. D. Causmaecker, and G. V. Berghe, "An intelligent hyper-heuristic framework for CHeSC 2011," in *Learning and Intelligent Optimization*, ser. LNCS, Y. Hamadi and M. Schoenauer, Eds. Springer Berlin Heidelberg, 2012, pp. 461–466.

[27] J. A. Vrugt, B. A. Robinson, and J. M. Hyman, "Self-Adaptive Multimethod Search for Global Optimization in Real-Parameter Spaces," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 243–259, 2009.

[28] F. Peng, K. Tang, G. Chen, and X. Yao, "Population-Based Algorithm Portfolios for Numerical Optimization," *IEEE Transactions on Evolutionary Computation*, vol. 14, no. 5, pp. 782–800, 2010.

[29] L. Xu, F. Hutter, H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *Journal of Artificial Intelligence Research*, vol. 32, pp. 565–606, 2008.

[30] H. H. Hoos, "Programming by optimization," *Commun. ACM*, vol. 55, no. 2, pp. 70–80, 2012.

[31] F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Tradeoffs in the empirical evaluation of competing algorithm designs," *Ann. Math. Artif. Intell.*, vol. 60, no. 1-2, pp. 65–89, 2010.

[32] F. Hutter, L. Xu, H. H. Hoos, and K. Leyton-Brown, "Algorithm runtime prediction: Methods & evaluation," *Artificial Intelligence*, vol. 206, pp. 79–111, 2014.

[33] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.

[34] F. Neri, E. Mininno, and G. Iacca, "Compact particle swarm optimization," *Information Sciences*, vol. 239, pp. 96–121, 2013.

[35] F. Caraffini, F. Neri, and L. Picinali, "An analysis on separability for memetic computing automatic design," *Information Sciences*, vol. 265, pp. 1–22, 2014.

[36] Y. S. Ong and A. J. Keane, "Meta-Lamarkian Learning in Memetic Algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 99–110, 2004.

[37] J. E. Smith, "Coevolving Memetic Algorithms: A Review and Progress Report," *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, vol. 37, no. 1, pp. 6–17, 2007.

[38] N. Krasnogor and J. Smith, "A tutorial for competent memetic algorithms: model, taxonomy, and design issues," *IEEE Transactions on Evolutionary Computation*, vol. 9, no. 5, pp. 474–488, 2005.

[39] ——, "Memetic Algorithms: The Polynomial Local Search Complexity Theory Perspective," *J. Math. Model. Algorithms*, vol. 7, no. 1, pp. 3–24, 2008.

[40] A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, "A fast adaptive memetic algorithm for on-line and off-line control design of PMSM drives," *IEEE Transactions on System Man and Cyberneticspart B*, vol. 37, no. 1, pp. 28–41, 2007.

[41] F. Neri, V. Tirronen, T. Kärkkäinen, and T. Rossi, "Fitness diversity based adaptation in Multimeme Algorithms: A comparative study," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2007, pp. 2374–2381.

[42] F. Neri, J. I. Toivanen, G. L. Cascella, and Y. S. Ong, "An Adaptive Multimeme Algorithm for Designing HIV Multidrug Therapies," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 4, no. 2, pp. 264–278, 2007.

[43] G. Iacca, F. Neri, E. Mininno, Y. S. Ong, and M. H. Lim, "Ockham's Razor in Memetic Computing: Three Stage Optimal Memetic Exploration," *Information Sciences*, vol. 188, pp. 17–43, 2012.

[44] F. Caraffini, F. Neri, G. Iacca, and A. Mol, "Parallel memetic structures," *Information Sciences*, vol. 227, no. 0, pp. 60 – 82, 2013.

[45] M. G. Epitropakis, D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Tracking differential evolution algorithms: An adaptive approach through multinomial distribution tracking with exponential forgetting," in *Artificial Intelligence: Theories and Applications*, ser. LNCS, Maglogiannis, Plagianakos, and Vlahavas, Eds. Springer, 2012, no. 7297, pp. 214–222.

[46] M. Epitropakis, D. Tasoulis, N. Pavlidis, V. Plagianakos, and M. Vrahatis, "Tracking particle swarm optimizers: An adaptive approach through multinomial distribution tracking with exponential forgetting," in *IEEE Congress on Evolutionary Computation (CEC)*, 2012, pp. 1–8.

[47] D. Thierens, "An adaptive pursuit strategy for allocating operator probabilities," in *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '05. New York, NY, USA: ACM, 2005, p. 15391546.

[48] ——, "Adaptive strategies for operator allocation," in *Parameter Setting in Evolutionary Algorithms*, ser. Studies in Computational Intelligence, F. G. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Springer Berlin Heidelberg, 2007, no. 54, pp. 77–90, 00042.

[49] L.-Y. Tseng and C. Chen, "Multiple trajectory search for Large Scale Global Optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, 2008, pp. 3052–3059.

[50] H. H. Rosenbrock, "An automatic Method for finding the greatest or least Value of a Function," *The Computer Journal*, vol. 3, no. 3, pp. 175–184, 1960.

[51] M. S. Bazaraa, H. D. Sherali, and C. M. Shetty, *Nonlinear Programming: Theory And Algorithms*. Wiley-Interscience, 2006.

[52] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[53] R. S. Sutton and A. G. Barto, *Introduction to Reinforcement Learning*, 1st ed. Cambridge, MA, USA: MIT Press, 1998, 02767.

[54] X. Li and X. Yao, "Cooperatively Coevolving Particle Swarms for Large Scale Optimization," *Evolutionary Computation, IEEE Transactions on*, vol. 16, no. 2, pp. 210–224, 2012.

[55] S. Islam, S. Das, S. Ghosh, S. Roy, and P. Suganthan, "An Adaptive Differential Evolution Algorithm With Novel Mutation and Crossover Strategies for Global Numerical Optimization," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 42, no. 2, pp. 482–500, 2012.

[56] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernndez-Daz, "Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on Real-Parameter Optimization," Zhengzhou University and Nanyang Technological University, Zhengzhou China and Singapore, Tech. Rep. 201212, 2013.

[57] N. Hansen, A. Auger, S. Finck, R. Ros *et al.*, "Real-Parameter Black-Box Optimization Benchmarking 2010: Noiseless Functions Definitions," INRIA, Tech. Rep. RR-6829, 2010.

[58] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[59] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.

PXI

STRUCTURAL BIAS IN POPULATION-BASED ALGORITHMS

by

Anna V. Kononova, David W. Corne, Philippe De Wilde, Vsevolod Shneer and Fabio Caraffini 2015

Information Sciences, volume 298, number 0, pages 468-490

Information Sciences 298 (2015) 468–490

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

Structural bias in population-based algorithms

Anna V. Kononova [a,*], David W. Corne [a], Philippe De Wilde [b], Vsevolod Shneer [c], Fabio Caraffini [d,e]

[a] Department of Computer Science, Heriot-Watt University, Edinburgh EH14 4AS, UK
[b] School of Computing, University of Kent, Canterbury CT2 7NZ, UK
[c] School of Mathematical and Computer Sciences, Heriot-Watt University, Edinburgh EH14 4AS, UK
[d] Centre for Computational Intelligence, School of Computer Science and Informatics, De Montfort University, Leicester LE1 9BH, UK
[e] University of Jyväskylä, Department of Mathematical Information Technology, P.O. Box 35 (Agora), 40014 Jyväskylä, Finland

ARTICLE INFO

Article history:
Received 8 August 2014
Received in revised form 5 November 2014
Accepted 22 November 2014
Available online 4 December 2014

Keywords:
Structural bias
Algorithmic design
Evolutionary computation
Population-based algorithm
Optimisation

ABSTRACT

Challenging optimisation problems are abundant in all areas of science and industry. Since the 1950s, scientists have responded to this by developing ever-diversifying families of 'black box' optimisation algorithms. The latter are designed to be able to address any optimisation problem, requiring only that the quality of any candidate solution can be calculated via a 'fitness function' specific to the problem. For such algorithms to be successful, at least three properties are required: (i) an effective informed sampling strategy, that guides the generation of new candidates on the basis of the fitnesses and locations of previously visited candidates; (ii) mechanisms to ensure efficiency, so that (for example) the same candidates are not repeatedly visited; and (iii) the absence of structural bias, which, if present, would predispose the algorithm towards limiting its search to specific regions of the solution space. The first two of these properties have been extensively investigated, however the third is little understood and rarely explored. In this article we provide theoretical and empirical analyses that contribute to the understanding of structural bias. In particular, we state and prove a theorem concerning the dynamics of population variance in the case of real-valued search spaces and a 'flat' fitness landscape. This reveals how structural bias can arise and manifest as non-uniform clustering of the population over time. Critically, theory predicts that structural bias is exacerbated with (independently) increasing population size, and increasing problem difficulty. These predictions, supported by our empirical analyses, reveal two previously unrecognised aspects of structural bias that would seem vital for algorithm designers and practitioners. Respectively, (i) increasing the population size, though ostensibly promoting diversity, will magnify any inherent structural bias, and (ii) the effects of structural bias are more apparent when faced with (many classes of) 'difficult' problems. Our theoretical result also contributes to the 'exploitation/exploration' conundrum in optimisation algorithm design, by suggesting that two commonly used approaches to enhancing exploration – increasing the population size, and increasing the disruptiveness of search operators – have quite distinct implications in terms of structural bias.

* Corresponding author.
E-mail addresses: anna.kononova@gmail.com (A.V. Kononova), d.w.corne@hw.ac.uk (D.W. Corne), p.dewilde@kent.ac.uk (P. De Wilde), v.shneer@hw.ac.uk (V. Shneer), fabio.caraffini@gmail.com (F. Caraffini).

http://dx.doi.org/10.1016/j.ins.2014.11.035
0020-0255/

1. Introduction

Successful implementation of any randomised population-based optimisation algorithm depends on the efficiency of both its sampling component and exploitation of previously sampled information. Among other fields, Evolutionary Computation (EC) [13] provides various examples of randomised population-based search strategies. Greatly simplified, any evolutionary computation algorithm is a guided re-sampling strategy where movement of points is directed by its operators assisted by selection criteria based on currently attained values of the objective function. A vast body of research in the field of Evolutionary Computation deals with efficient exploitation of information already contained within the population [49] while little attention has been paid to investigation of whether or not a specific combination of algorithmic operators is actually capable of reaching all parts of the search space efficiently. This paper attempts to draw attention to this issue and starts to investigate the latter question.

Inspection of recent literature [40,9,38,55] confirms the presence of a tendency to (over-) complicate both the design of individual algorithmic operators and the logic of their assembly, counter to the rationale of the well-known Occam's razor,[1] sometimes to such a degree that the end result turns out to be intractable. Researchers seem regularly to be swayed by an attraction towards 'multiplying entities beyond necessity'. We suggest that a materially greater contribution to the understanding of population-based algorithms and their design can be obtained via 'going back to basics'. More specifically, the great majority of optimisation algorithms fall within a class of generate-and-test methods, iteratively alternating between these two components until a termination criterion is met. Ideally, the generating/sampling component of such methods should have the following characteristics [51]:

1. future samples should be biassed by information obtained from previously visited points, i.e., the algorithm should be *informed*,
2. future samples should be previously unvisited samples i.e., the algorithm should be *non-redundant*,
3. every solution in the search space should be equally accessible i.e., the algorithm should be *complete*.

It is worth noting our use of the phrase *equally accessible* within the 'completeness' characteristic. Often, for example, algorithm designers may be subconsciously swayed by the fact that the randomness of the initialisation process means that every part of the search space is *reachable*, and hence feel no further need to consider this characteristic. Reachability and completeness (the way we define it here) are however very different. For example, if a stochastic hill-climbing algorithm includes a uniform random initialisation in $\mathbb{R}^n$, then all points are reachable, however if the perturbation operator is designed to add only integer-valued vectors, then there are extreme variations in the accessibilities of different points in the space.

Clearly, evolutionary computation methods build richly upon their 'generate-and-test' backbone architecture. However the above guidelines remain valid, and, in practice, they translate well into rules for algorithm design. The first two properties – informedness and non-redundancy in the sampling process – have been extensively researched, each from a variety of viewpoints. To some extent, however, contributions related to these two properties have appeared in diverse and unconnected literature, using varying terminology, and there remains a need to creatively assimilate their findings.

For example, with sufficient imagination one can see that the *informedness* property is closely linked to the concepts of exploration, exploitation, and their balance, which is considered to be primary in the behaviour of evolutionary algorithms (EAs), as examples of stochastic ''generate-and-test'' methods [11]. Exploration and exploitation are fundamental for evolutionary optimisation [13] but surprisingly, several decades after the first examples of EAs have been proposed, they still lacked even a proper definition. Over the following years, a lot of research has been carried out in this direction - the latest survey of results can be found in [49]. The current consensus definitions consider *exploration* as the process of visiting entirely new regions of a search space whilst *exploitation* as the process of visiting those regions of a search space within the neighbourhood of previously visited points [49].

The second characteristic, *non-redundancy*, has been investigated under the guise of 'non-revisiting' algorithms. Inspired by ideas from Tabu search [15,16], basic evolutionary algorithms have been extended to ensure the non-revisiting property [52–54]. Another direction of research into the non-redundancy property is the study of diversity management in evolving populations. Diversity in populations can refer to differences in solutions in either the values of coordinates ('genotypic' diversity) or the objective function ('phenotypic' diversity). To date, no single measure exists which can suitably characterise diversity in the face of all kinds of problems and search logics [37]. The situation is further complicated by the fact that a diverse population offers benefits at some stages of evolutionary process (helps avoid premature convergence to local optima) and creates obstacles in others (impedes exploitation) [7]. The most popular diversity-preserving mechanisms include [18] niching, crowding, restricted mating, sharing, multiploidy, elitism, injection, alternative replacement strategies [32] and fitness uniform selection [20].

Much promising research is also carried out that tries to explore the connections between *informedness* and *non-redundancy*, stemming from the fact that exploration of the search space is only possible if populations are diverse enough [49]. However, different amounts of exploration and exploitation are needed for different optimisation problems. Currently there

[1] Originally attributed to William of Occam, reformulated by Betrand Russell as "Whenever possible, substitute constructions out of known entities for inferences to unknown entities". [43]

are no accepted techniques for direct measurement of this balance; it can only be noisily sensed via a proxy (such as 'level of diversity'). Feedback from online monitoring of such a proxy, if it is suitably computationally efficient, can then be used to dynamically tilt the exploration–exploitation balance [49,29,6].

1.1. Completeness and incompleteness: bias and its treatment in the research literature

Meanwhile, in recent decades, the third important property of the generating component, which we can call *completeness*, has been treated as an obscure topic and largely ignored by modern research efforts – the latest reference to this issue, [12], briefly deals with the accessibility of a solution through evolution as a necessary condition for reaching the optimum by a genetic algorithm in the discrete case.

However, there has been a recent revival of interest pertaining to this topic, in the form of a focus on certain deficiencies in completeness – in other words biases – that have been frequently observed in studies on real-valued function optimisation. The interest of a small group of researchers has recently been sparked by the popular belief [14,2] that many population-based algorithms (and particularly PSO) tend to perform better when the true optimum is located at or near the centre of the initialisation region. Two types of bias in particular have been investigated: initialization-region bias (IRB), and centre-seeking bias (CSB). IRB basically refers to the tendency for the search process to stay close to the region of the initial population of points, while CSB refers to the tendency for the search process to gravitate towards the origin of the search space. The reader may find further detail in [8]. The foci of recent articles on this topic have been on (i) observing either IRB or CSB or both in the context of specific algorithms and optimisation landscapes, and (ii) constructing algorithmic mechanisms that avoid them. In [8], for example, metrics are proposed for measurement of both IRB and CSB, and having observed rampant IRB and CSB in a specific algorithm, proposing a modified version that displays much less bias.

To date, such studies have focussed on a small number of algorithm variants and there is little or no evidence to support the presence of such a bias in the more general case. For example, having investigated the effects of modifying the search domains of three benchmark problems on results produced by an unusual variant of PSO, the authors of [36] concluded the presence of origin-seeking bias in their specific algorithm/problem scenarios, and suggested that their results could be generalised towards all population-based methods. However these results were later disputed and have been largely dismissed [24]. As regards theoretical analysis of the movement of particles in a PSO swarm, a study of particle trajectories [48] reveals that under certain conditions every particle converges to a stable point defined by its personal and global best positions, with weights determined by the acceleration coefficients. Experimental results also suggest that, for the well-known *sphere* objective function, the movement of particles is influenced by the direction of the coordinate axis which potentially makes the algorithm sensitive to rotation of the objective function [23]. Further theoretical analysis [45] indicates that there is an angular bias in the core PSO algorithm which consists of two parts. The first part, skew, pushes particles towards bearings parallel with the diagonals, meanwhile the second part, spread, indicates that diagonal directions are highly unstable. The combination of the two parts creates a PSO bias that favours particle bearings that are aligned with the coordinate axes. The latest publication on this topic [8] extends the work from [36] and proposes a metric for centre-seeking and initialisation biases based on multiple re-runs of the algorithm in modified domains.

The majority of authors implicitly suppose their algorithms fare well in the ability to potentially cover the whole search domain. Put another way, researchers tend to take for granted the property of 'completeness'. Such quiet confidence about this property probably stems from the perception that, given the stochastic nature of standard initialisation methods and standard operators, all parts of the search space are *reachable*. However, this ignores the prospect that reachability may actually be highly non-uniform across the search space. Results presented in this paper demonstrate that even the most commonly used algorithms exhibit inherent preferences towards certain parts of the search domain. We refer to this preference as the *structural bias* of the algorithm. In contrast to the recent work discussed above, our focus is on understanding the inherent structural bias of an algorithm, rather than measuring the effects of any such bias on specific optimisation landscapes. Although the latter effects are of course important, our hypothesis is that a more fundamental understanding of structural bias will help to better characterise, and inform how to avoid, CSB, IRB, and other manifestations of inherent structural bias in applied work.

To help illustrate the concept of structural bias, it may be helpful to imagine a pinball machine where a player has to operate a system of mechanical devices to allow a ball to stay on the game surface as long as possible before hitting the drain, see Fig. 1. We can consider the whole system – the pinball machine and the actions of the player – to represent an algorithm, while the ball represents a solution travelling around the search space. We can conceptualise multiple games on such a machine, overlapped in time, to represent the case of an algorithm that maintains a population of solutions. In the ideal case, the population should be able to access the entire game surface. A population-based algorithm exhibiting structural bias is then replicated by an overlapping in time of such machines which are unfairly tilted at some angle. Clearly, even in this case the actions of the player have a certain effect on the movement of the balls. However, due to gravity, the ball ends up constantly rolling to the lower side of the machine i.e., exhibiting a certain preference and limiting the overall coverage of the game surface.

1.2. An overview of the remainder

The remainder of the paper is arranged as follows. In Section 2 we argue that studying the structural bias that may be inherent in an algorithm can be facilitated by decoupling the effects of the search landscape from the algorithmic operations.

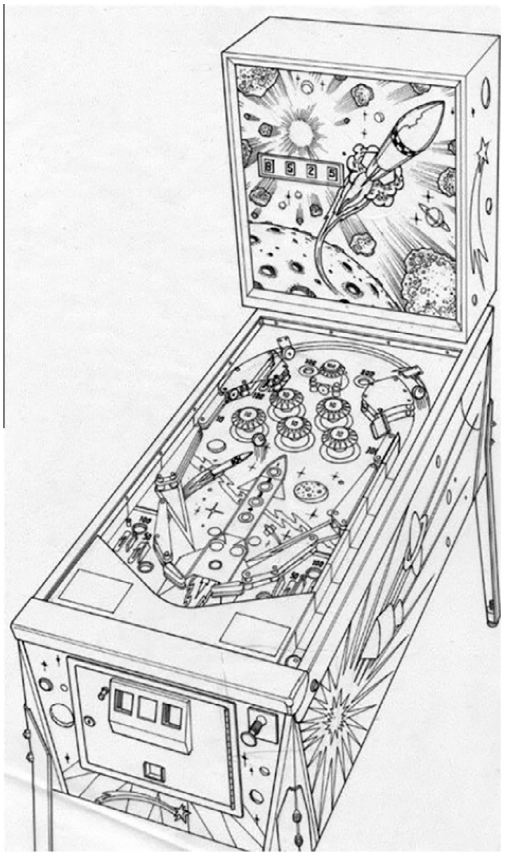


Fig. 1. A typical pinball machine.

This section then introduces and analyses a test function f_0, which allows such decoupling. Section 3 takes the function f_0 and uses it as a 'structural bias probe', presenting several experiments in which we investigate whether structural bias seems to be present in typical designs of a genetic algorithm and a particle swarm optimisation algorithm. Visualisations of the results of our experiments in this section offer evidence that structural bias is indeed present in these algorithms, and sensitive to parameters such as population size. This section ends with remarks concerning pseudorandom number generators, in particular those used in our implementation, and offers evidence – following further empirical investigation – that supports the view that our observations in the previous section were uncontaminated by artefacts of the pseudorandom generator. In Section 4 we turn to a theoretical investigation, which considers a simplified genetic algorithm (nevertheless capturing well the behaviour of a typical genetic algorithm on f_0). The theorem proved in this section shows that the considered (simplified but otherwise standard) algorithm design will, under certain but unexceptional conditions, induce a continual reduction in sample variance over time; this means that the population will increasingly cluster around certain areas of the domain while avoiding others. Reasoning based on the theorem leads to expectations of the relationship between a genetic algorithm's population size and the occurrence of structural bias, which match our empirical findings from Section 3; further reasoning predicts a relationship between structural bias and problem difficulty, which is tested in the experiments of the next section. Section 5 begins by outlining and demonstrating approaches to visually investigate, and then to quantify, the levels of structural bias inherent in the design of an optimisation algorithm. This is followed by an examination of how structural bias seems to manifest differently when we apply our standard genetic algorithm to variety of functions from a well-known test suite. The findings from these experiments again match with theory-grounded expectations arising from our arguments in Section 4. Finally, Section 6 provides a summary of the paper, and brief discussions of its wider relevance, such as the manifestation of structural bias in combinatorial spaces.

2. Structural bias

When faced with the task of optimising a given function, the amount of information usually available regarding its features is highly limited. Therefore, one wishes to design an algorithm capable of locating the optima no matter where exactly they are in the search space. This implies that the generating operators of the algorithm must be able to, first, reach every region of the search space and second, ideally, do so without imposing any preferences for some regions of the domain over others. Clearly, different functions and domains give rise to different situations, greatly complicating the prospects for a general theoretical analysis. In addition, such an analysis cannot be tackled directly due to the apparent coupling between

the landscape of the objective function and artefacts from the iterative application of algorithmic operators i.e., structural bias. It is therefore highly desirable to be able to separate these effects. A closer inspection reveals that, in almost all cases, the action of a selection operator actually can be characterised as the imposition of a stochastic rank-ordering over a specific set of values of the objective function in the current population. If we replaced the objective function with uniform random noise, over a series of statistically significant number of independent runs, this would enable us to separate 'landscape effects' from 'algorithm design effects', eliminating the influence of the ('geographical') position of selected points, but retaining algorithmic artefacts.

Therefore, one way to overcome this coupling issue is to use 'the most random' test function, such that its value at any point does depends neither on the values within its neighbourhood nor on the past (independent) evaluations at this point i.e., be independent and identically distributed (i.i.d.). For the sake of simplicity, and without loss of generality, we can consider an artificial objective function

$$f_0 : D \subset \mathbb{R}^n \to [0,1], \text{ where } x \in \mathit{Uniform}(D),\ f_0(x) \in \mathit{Uniform}[0,1],\ x \text{ and } f_0(x) \text{ are i.i.d.} \tag{1}$$

as this 'most random' function. Again, without loss of generality, we can consider $D = [0,1]^n$. Such an f_0 contains no structure stable over different runs, therefore an *ideal optimisation method* will arrive at different regions of the search space over a series of runs. Indeed, over multiple runs, it will cover the entire search space with uniform probability. In other words, when an algorithm is applied to f_0, we can expect that the distribution of its outcomes over multiple runs will show no bias of any kind, *if and only if* the algorithm itself possesses no inherent structural bias (see Section 2.1). In contrast to the enterprise of observing bias on a real-world (or other) function optimisation problem, this means that f_0 is able to reveal bias that is inherent in the algorithm itself, rather than effects that may be caused, in whole or in part, by features of the optimisation landscape at hand. This also makes f_0 an ideal testing substrate for engineering the algorithm in such a way that bias is minimised or eliminated. Also, when tests on f_0 reveal structural bias, this serves to warn the practitioner that its effects may divert the results of applications of the algorithm. In what way, and to what extent the bias will affect the application, depends (we believe) on aspects of the fitness landscape, and we return to this point specifically in Section 5.1.

As shown in Fig. 2, the typical progress of a capable evolutionary algorithm consists of three stages [13]: in the beginning, the population is spread randomly over the domain, roughly halfway through the optimisation the population starts rolling down the hill, and in the final stages of optimisation the whole population is concentrated around the minima. Thanks to the construction of f_0, independent runs of the algorithm provide different landscapes, all of identical difficulty (due to the i.i.d. property), where populations move/converge towards minima located at different parts of the domain. In other words, over a series of runs of the algorithm, the situation shown in Fig. 2 is replicated for different landscapes where the optimisation process arrives at different parts of the domain – that is, red points will be distributed all over the interval. In the following section we show that minima of f_0 are in fact distributed uniformly over D. This implies that the distribution of minima found by an ideal unbiased algorithm across different runs should be uniform as well.

2.1. Distribution of minima of f_0

Assume that points $Z_1, \ldots, Z_N$ are independent and identically distributed. Assume that each of these points (say, Z_i) is assigned a mark X_i and assume that $X_1, X_2, \ldots, X_N$ is a collection of i.i.d. random variables with an absolutely continuous distribution. Assume also that the sets $X_1, X_2, \ldots, X_N$ and $Z_1, \ldots, Z_N$ are mutually independent. Let $I = \arg\min_i X_i$ be the index of the point with the lowest mark.

Remark 1. We only assume that the distribution is absolutely continuous for convenience here. This ensures that $P(X_i = X_j) = 0$ for any $i \neq j$. This makes our proofs shorter and more transparent but is not essential for our statements to hold.

Remark 2. In the notation above, $Z_1, \ldots, Z_N$ represent the vector of points' coordinates and $X_1, \ldots, X_N$ represent values of the objective function at these points.

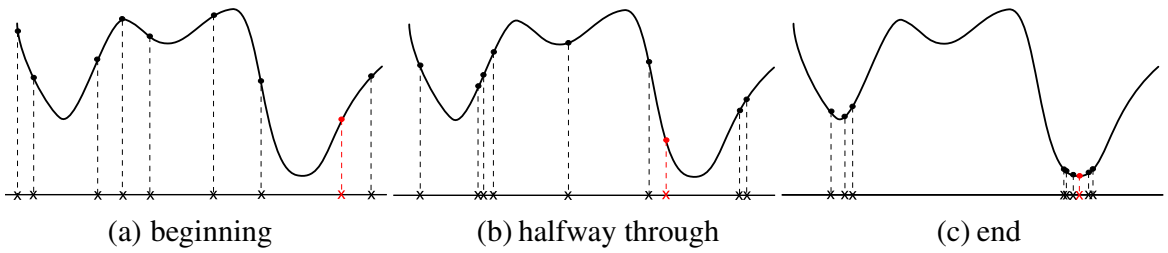


Fig. 2. Sketch of a typical progress of an evolutionary algorithm on a minimisation problem in terms of population distribution with projections of points' coordinate, adapted from [13]. Red points mark best points in the population. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Proposition. The distribution of Z_I is the same as that of Z_1 (or the same as the distribution of any of the initial points).

Proof. Note first that $P(I = i) = 1/N$ for any i. This is evident as

$$\sum_{i=1}^{N} P(I = i) = 1,$$

and all probabilities are equal to each other due to the identical distribution of X.

We can now calculate, for any set of points A

$$P(Z_I \in A) = \sum_{i=1}^{N} P(I = i)P(Z_I \in A | I = i) = \sum_{i=1}^{N} \frac{1}{N} P(Z_i \in A) = P(Z_1 \in A) \qquad \square$$

This shows that minima of f_0 defined in (1) are distributed uniformly over D.

2.2. Further comments on f_0

Since, by construction, f_0 is in effect a noisy signal with zero smoothness i.e., no correlation between neighbouring points, it is clearly not suited for testing the quality of fitness improvement or as a direct guide in assembling the algorithm. As explained previously, the rationale behind using f_0 is solely to elucidate the underlying structural bias of the tested algorithm. More comments on this issue are given in Section 5.1.

3. Numerical results

In this section we illustrate the use of f_0 in investigating the occurrence of structural bias in different algorithm configurations. We apply this 'structural bias probe' to two algorithms that are frequently deployed in optimisation practice and research, namely: a genetic algorithm (GA), and particle swarm optimisation (PSO). In both cases, our instantiations of the algorithms (and the subsequent further analyses), are in the context of optimisation in a continuous decision space (i.e. the optimisation of vectors of real-valued parameters). Combinatorial optimisation is certainly also of interest, and we later briefly speculate on structural bias in that scenario. However, our focus here on real-valued decision spaces is consistent with the observation that real-valued optimisation (particularly via PSO variants) is the most rampant breeding ground for the publication of new algorithms. As such, real-valued optimisation can be considered in relatively more need for techniques that can help researchers or practitioners discern performance-related properties of new algorithm designs.

3.1. Typical genetic algorithm

As the first example of a randomised population-based algorithm used to solve the problem of minimisation of $f_0 : [0, 1]^n \to [0, 1]$, we consider the most straight-forward example – a typical steady-state genetic algorithm (GA) where solutions are encoded as strings of real values of length n and are subject to the following transformations:

1. initialize and evaluate a population of N solutions within the boundaries of problem domain;
2. continue until the maximum number of fitness evaluations is reached (300,000);
2.1. select $parent_1$ *from the population via tournament selection [17] of size* n_t *(here,* $n_t = 2$*) in the following manner;*
2.1.1. select at random n_t *solutions;*
2.1.2. based on their fitness values, choose the best solution to become a parent;
2.2. similarly, select $parent_2$ *via tournament selection of size* n_t*, independently on the choice of another parent;*
2.3. generate child solution as $parent_1 + \alpha * (parent_2 - parent_1), \alpha \sim Uniform(-d, 1 + d)$ *re-sampled for each dimension,* $d = 0.25$*;*
2.4. with probability $p = 1$*, mutate child solution via Gaussian mutation – perturb every coordinate independently with* $\delta \sim N(0, m_d * r), m_d = 0.01, r$ *is the width of search domain in this coordinate;*
2.5. evaluate child solution;
2.6. if child solution is better or equal to current worst solution in population then child solution replaces it;
2.7 end of loop, go to step 2. $\square$

All specified parameters represent standard choices in the field of evolutionary computation. If a result of an operator, in some dimension, goes outside the domain, it is corrected in a saturation manner where it is forced to the closest domain boundary in this dimension. The dimensionality of the problem is set to $n = 30$ as a value high enough to be relevant for the field but low enough to allow clear visualisation. This also dictates the choice of the termination criterion as 300,000 fitness evaluations, following [47]. We consider three settings for population size: $N = 5, N = 20, N = 100$. To provide enough statistical power for the results, 50 independent runs are considered for each parameter setting. According to [41], in the

limit, this algorithm converges to the global minimum of any real-valued function $f : M \to \mathbf{R}$ whose values are bounded below and M is an arbitrary domain.

3.2. Numerical results for a typical genetic algorithm

A convenient way of visualising multidimensional data is by the method of 'parallel coordinates' [10,21] which allows an insight into the space regardless of its dimensionality. Using this technique to visualise an n-dimensional point, a backdrop consisting of n vertical equally spaced parallel lines is drawn and a point in n-dimensional space is represented as a collection of markers on each of these n lines, each matching a value of the corresponding coordinate. Traditionally, these markers are connected to form polylines (piecewise linear curves) which can reveal 2-dimensional patterns for certain high dimensional properties [22]. Unfortunately, finding the correct layout for each dataset to facilitate data exploration is a problem on its own [56], especially for high values of n [19]. Such investigation is currently beyond the scope of our interests, however it may become of interest for algorithms that use highly correlated search strategies. Since the focus of this paper is solely the movement of the population of points in the search domain, unconnected markers suffice. Using colour allows us to visualise the additional dimension – the value of the objective function at the point.

Following the technique described above, Fig. 3 shows, in parallel coordinates, positions of points with the best fitness values in the first (left column) and the last (right column) populations for 50 runs of the considered genetic algorithm, for different population sizes $N = 5, N = 20, N = 100$. Clearly, for all population sizes, the initial distribution of positions in the left columns of the figures is close enough to uniform. However, in the right column, instead of seeing a near-uniform distribution of points, a clear bias towards the centre of the search space becomes more evident in the final populations as population size increases. In other words, a genetic algorithm with bigger populations tends to avoid the corners of the search domain and concentrates more on sampling points closer to the middle of the interval, for no obviously apparent reason. Such behaviour is barely noticeable for $N = 5$, more pronounced for $N = 20$ and is very clear for $N = 100$. These anomalies represent structural bias. Our numerical results also suggest that this behaviour is consistent throughout time and does not depend on termination criterion - consecutive populations spread out less and less from the middle of the search domain, see Fig. 4 which shows the evolution in time of positions of all points of 50 populations in a selected dimension for all three population sizes. *We therefore conclude that, owing to structural bias, a typical genetic algorithm with a large population potentially wastes the fitness evaluations budget via oversampling a region of the search domain, to the detriment of overall performance.*

3.3. Typical particle swarm optimisation algorithm

Particle swarm optimisation (PSO) is another example of a population-based optimisation algorithm introduced by Kennedy and Eberhart in [25], and then developed in various variants for test problems and applications. The main metaphor employed in particle swarm optimisation is that a group of particles makes use of their personal and social experience in order to explore a decision space and detect solutions with high performance.

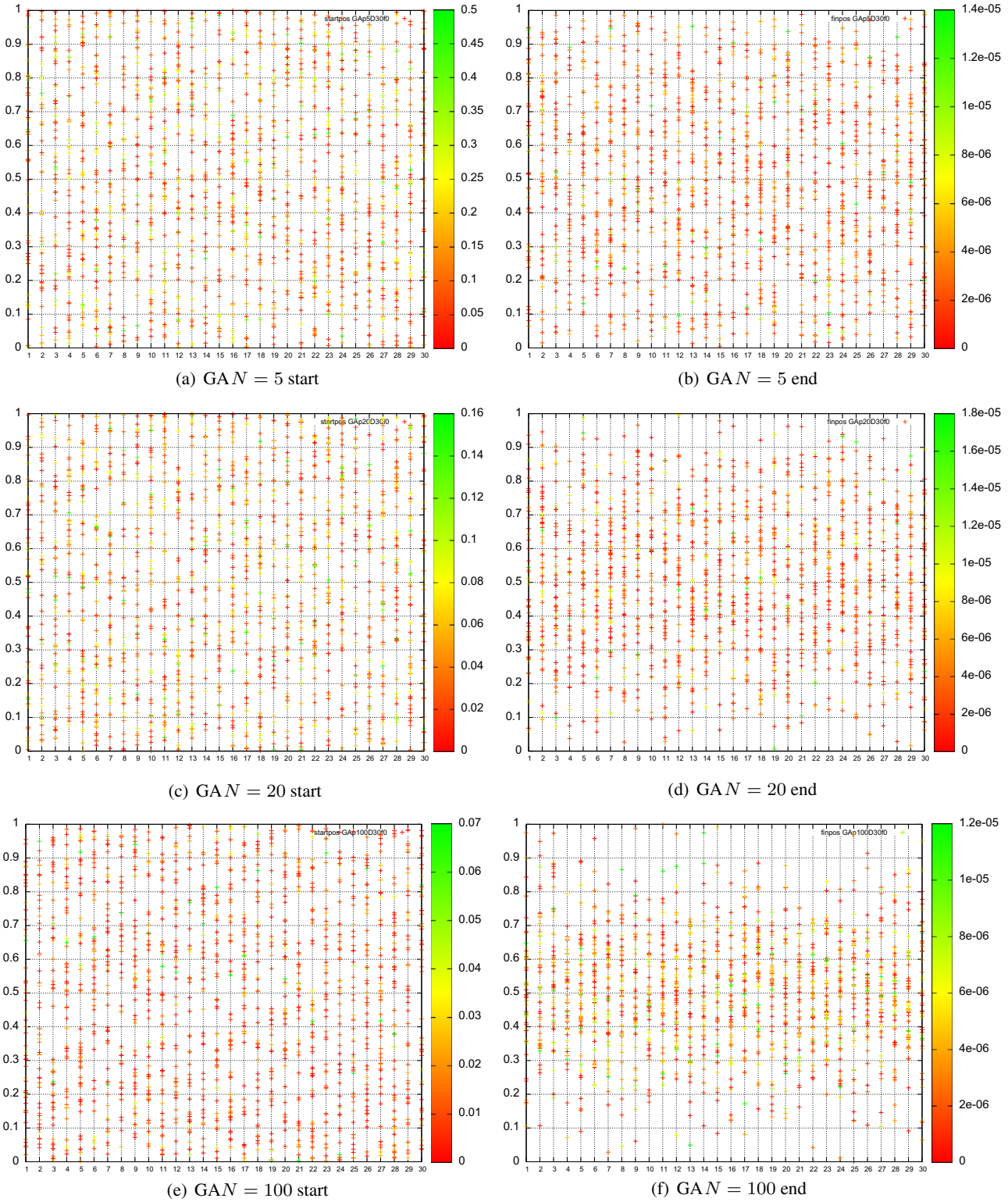
More specifically, to minimise $f_0 : [0,1]^{30} \to [0,1]$, the following steps are taken:

1.1. initialise a population of N solutions within the boundaries of the problem at $t = 0$
1.2. evaluate every solution in the population based on the objective function
1.3. for each solution, assign its personal best position $p_t^{pb} = p_0$
1.4. assign the global best position to p_t^{gb}
1.5. for each solution, initialise a speed vector $v_0 = (v_0^1, \dots, v_0^n)$ *such that* $v_0^i \sim Uniform[0, 0.1]$
2. continue until the maximum number of fitness evaluations is reached (300,000)
2.1. update the speed vector for every solution in the population as $v_{t+1} = c_0 v_t + c_1 \alpha_1 (p_t^{pb} - p_t) + c_2 \alpha_2 (p_t^{gb} - p_t)$, *where* $\alpha_1, \alpha_2 \sim Uniform[0,1]$ *are re-sampled independently for each solution and* $c_0 = 1, c_1 = 2, c_2 = 2$
2.2. if $\|v_{t+1}\|_2 > 0.2$ *substitute it coordinate-wise with* $\frac{0.2 v_{t+1}^i}{\|v_{t+1}\|_2}, i = 1, \dots, n$
2.3. update the position of every solution in the population as $p_{t+1} = p_t + v_{t+1}$, *evaluate the new solution*
2.4. if needed, update personal best position p_t^{pb} *for each solution*
2.5. if needed, update the global best position p_t^{gb}
2.6. $t = t + 1$, *end of loop, go to step 2* □

As well as in the previous section, the algorithm and specified parameters represent standard choices in the field of evolutionary computation. To allow fair comparison, the termination criterion is kept as 300,000 fitness evaluations. Finally, echoing the experiments done with a typical genetic algorithm, here we also use the three settings for population size $N = 5, N = 20, N = 100$, and we perform 50 independent runs for each.

3.4. Numerical results for PSO

The same techniques as in Section 3.2, applied to the analysis of PSO, reveal a rather different situation as shown in Fig. 5. As expected, the distribution of initial positions of points with the best fitness shown in the left columns of the figures is



(a) GA $N = 5$ start (b) GA $N = 5$ end

(c) GA $N = 20$ start (d) GA $N = 20$ end

(e) GA $N = 100$ start (f) GA $N = 100$ end

Fig. 3. Positions of points with the best fitness values in the first (left column) and the last (right column) populations of 50 runs of the considered GA for different population sizes in parallel coordinates; horizontal axis shows the ordinal number of the coordinate, vertical axis shows the range of this coordinate; fitness value of each point is shown in colour. A clear bias towards the centre of the search space is visible in the last populations as population size increases. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

close enough to uniform. But, as in the case of the typical genetic algorithm, instead of a near-uniform distribution, positions of the final points show a clear bias, albeit of a different nature. A more complex type of dependency of the population size on the bias induced is present for the highest and lowest values of the considered population sizes. In the case of $N = 5$, positions of final points clearly group around the corners and avoid regions in the middle of the search domain. Meanwhile,

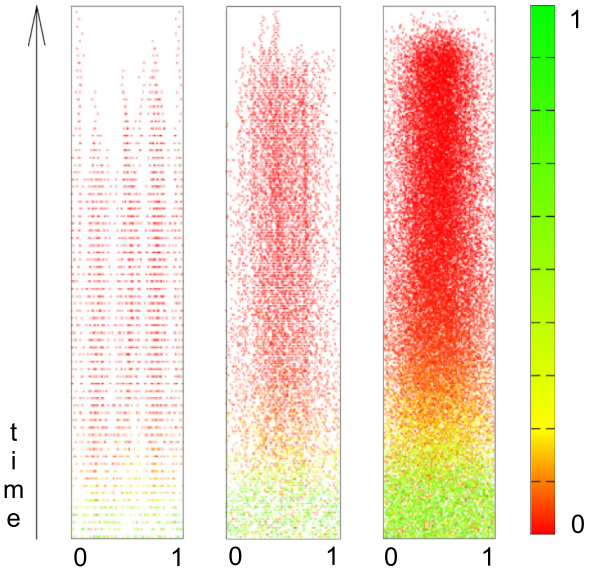


Fig. 4. Evolution in time of 50 populations of a typical genetic algorithm in selected dimension for $N = 5, N = 20$ and $N = 100$. Horizontal axis shows values of coordinate, vertical axis represents time. Colour of the dots corresponds to values of objective function. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

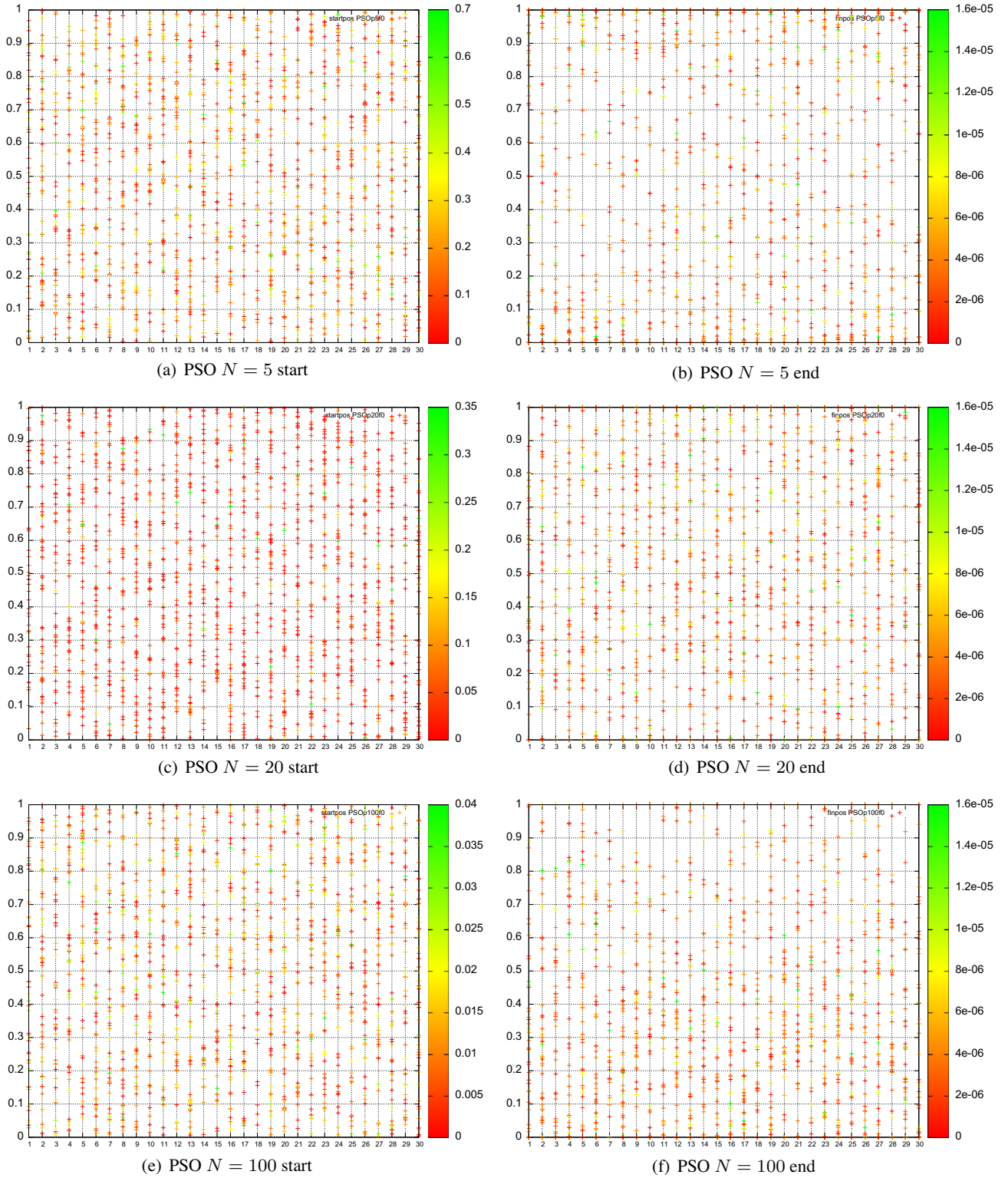
for $N = 100$, the final points tend to be positioned closer to one corner of the hypercube domain and avoid its opposite corner. The behaviour of PSO with $N = 20$ is also not ideal in terms of the distribution of final points, as they appear slightly further apart compared to the initial distribution. These anomalies clearly demonstrate the presence of structural bias in the considered version of PSO. Moreover, since none of the operators that makes up this PSO clearly predispose its population to cluster in such manner, this effect seems to be emerge from their combined dynamics.

3.5. Remarks regarding the random generator

The empirical results for the genetic algorithm in this paper are produced involving the use of a standard Java 48-bit random generator underpinning, where required, the generation of 'random' numbers within the implemented algorithm. This commonly-used pseudorandom generator is based on the linear congruential generator (LCG) with a period of $2^{48} \approx 2.8 \cdot 10^{14}$, while the seed is automatically generated via the system routine *System.currentTimeMillis()*. Meanwhile, our empirical PSO results are produced using the standard Unix function *drand48* with the same parameters as above and the seed value is obtained via *srand48*. It is well-known [33] that when a series of consecutive values are obtained from this type of random generator to form multidimensional points, they end up lying on a finite number of hyperplanes intersecting the intended domain. This property is usually referred to as the Marsaglia effect. Clearly, unless the precision of the random generator is close to the precision used by the algorithm, this constitutes a problem as, even in the limit, these points cannot fill all of the domain. The number of such planes is bounded by $(n!m)^{1/n}$, where n is the dimensionality and m is the modulus of the LCG. For the case of 30 dimensions, the bound on the number of planes is 36. The quality of each version of LCG can be further assessed based on its values of increment and multiplier via estimating the distance between the hyperplanes. However, such calculations are feasible reliably only for low dimensionalities [30].

Another usual concern about random generators is how random their output actually is, in the sense of correlation between successive instances (as opposed to their coverage of the domain). There are two kinds of random generators which differ in how the numbers are produced: true random generators sample some source of entropy [26], whereas pseudorandom number generators use a deterministic algorithm to produce random looking numbers. True random generators measure some physical phenomenon that is expected to be random and then compensates for possible biases in the measurement process. Example sources include measuring atmospheric noise, thermal noise, and other external electromagnetic and quantum phenomena. Being truly non-deterministic and aperiodic, unfortunately, these generators are also slow, costly, inefficient and not reproducible which makes them a bad choice for practical sampling applications. It is still an open question as to whether it is possible in any practical way to distinguish the output of a well designed pseudorandom generator from a perfectly random source without knowledge of the generator's internal state [26].

How should these observations concern us? Like virtually all implemented stochastic algorithms, our 'random' numbers are *pseudorandom*. Intuitively, we might expect bias in the pseudorandom generator to be swamped by the combined action of the algorithmic operations and subsequently be invisible in the results – this is, indeed, the common (implicit) approach. However, the design of f_0 explicitly removes one of the several dynamic forces that we would otherwise expect to contribute to this 'washing out' of any effects from the pseudorandom generator. To some, combining this with the perhaps-unexpected appearance of evidence for structural bias may lead to a suspicion that what we have observed could be artefacts of the



(a) PSO $N = 5$ start (b) PSO $N = 5$ end

(c) PSO $N = 20$ start (d) PSO $N = 20$ end

(e) PSO $N = 100$ start (f) PSO $N = 100$ end

Fig. 5. Positions of points with the best fitness values in the first (left column) and the last (right column) populations of 50 runs of the considered PSO for different population sizes in parallel coordinates; horizontal axis shows the ordinal number of coordinate, vertical axis shows the range of this coordinate; fitness value of each point is shown in colour. A more complex type of dependency of the population size on the bias induced is present for the top and bottom values of the population size. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

pseudorandom generator. Intuition for the opposite conclusion is well-fuelled. For example, the Marsaglia effect is quickly obscured by aspects of the algorithm that distort the uninterrupted sequential mapping that the Marsaglia effect assumes, and (especially) are dense in operations that will move points away from the 'Marsaglia planes'. Also, as we discuss further below, modern pseudorandom generators are quite effective at avoiding periodic correlations. Nevertheless, in this section

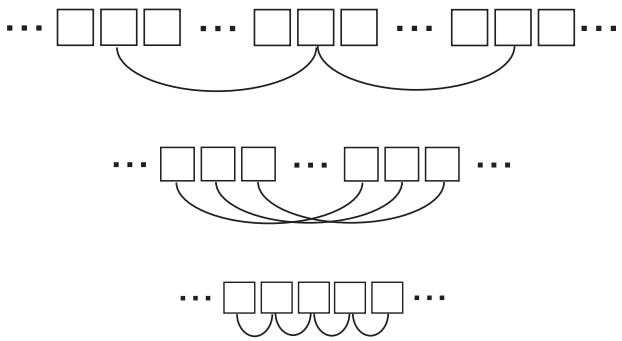


Fig. 6. Schematic explanation of tests 1–3 examining correlations between elements of random sequences. Squares represent consecutive elements of random sequence and loops denote considered correlations where length of the loop is constant for each test and referred to as period of this test. Such types of correlations can potentially induce patterns similar to those produced by structural bias.

we place the pseudorandom generator under close scrutiny in order to uncover evidence as to whether it may have a bearing on our findings.

To achieve this we have devised three tests, borrowing their design from the body of research that has gone into designing new classes of pseudo-random generators and testing their properties from various angles [42,50,34,1,26]. Each test within these test suites is aimed at finding a different kind of non-randomness, but as yet no specific finite set of tests is deemed complete to guarantee that some generator is foolproof [26]. For the purposes of this paper, the aspects mostly of interest are true uniformity and the absence of correlations in a long sequence of random values. There is no problem with uniformity as generators employed for this paper are among the most popular implementations used and tested widely.[2] Regarding cross-correlations in the sequence of random values, apart from the aforementioned Marsaglia effect investigated for low dimensionalities, little in the way of theoretical results is available. Values of cross-correlation lag (or 'effective period' as we refer to it here) which need to be studied usually exceed the dimensionality of the objective function, since the majority of algorithms use random values for altering various parameters throughout the run. Careful examination of the pseudocode provided in Sections 3.1 and 3.3 shows that both the genetic algorithm and PSO start with initialisation of their populations which require $(dim + 1)N_{pop}$ random numbers for the genetic algorithm[3] and $(dim + 1 + dim)N_{pop}$ random numbers for PSO, where $dim = 30$ is the dimensionality of the domain and N_{pop} is population size set to $5, 20$ and 100 for both algorithms. Subsequent functioning of the algorithm is periodic in the following sense: producing every new point to be examined by the algorithm requires the same amount of random numbers – $2dim + 5$ for the genetic algorithm and $2dimN_{pop} + 1$ for PSO. In other words, if i is an index of the element of the pseudorandom sequence which is used to generate the position of the new point in dimension j, then $i + p_e$ is the index of the next pseudorandom element which will be used to generate the position of the subsequent new point in dimension j, where p_e is the effective period.[4]

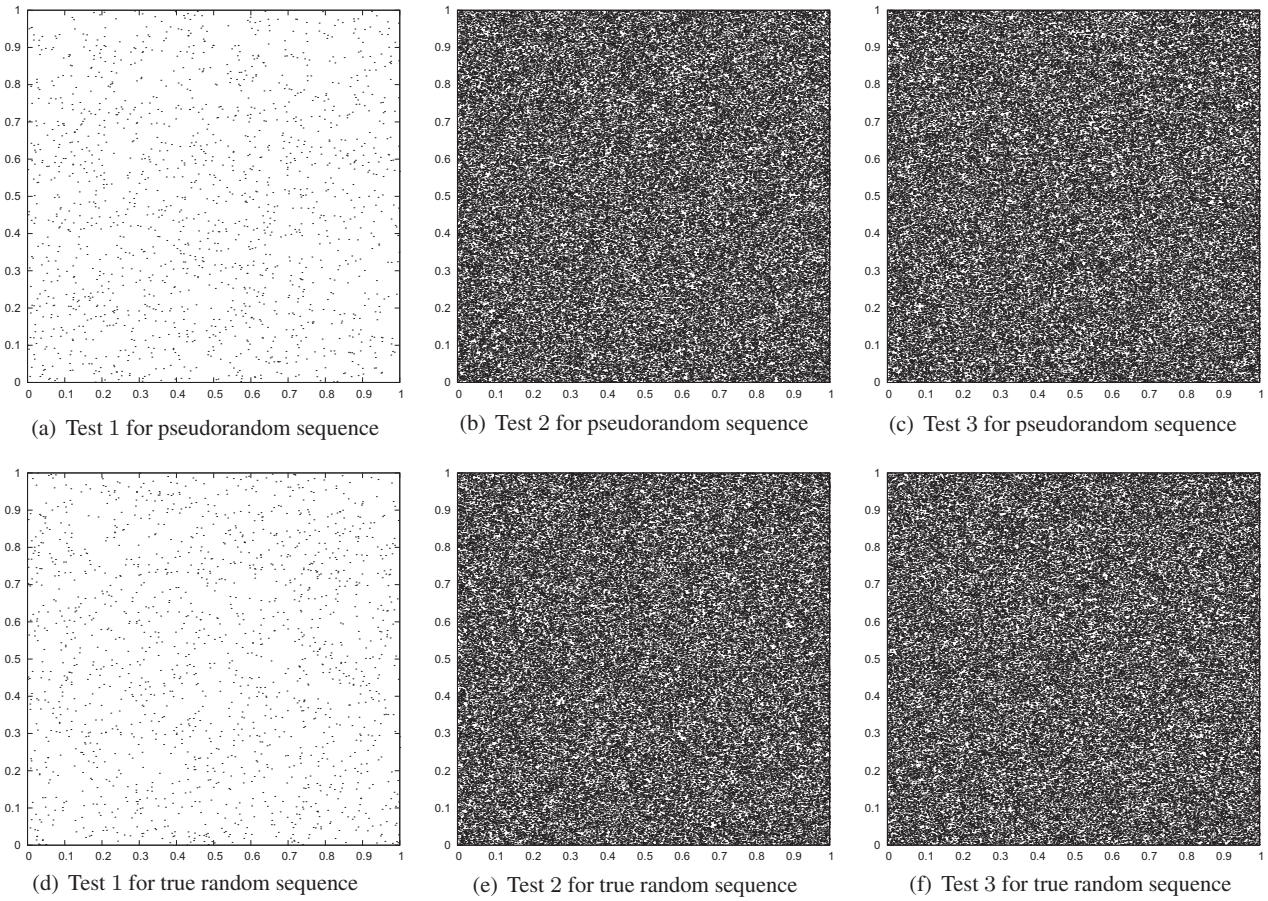
To eliminate the possibility that structural bias observed in algorithms considered in this paper originates from the nature of the pseudorandom number generation rather than being inherent to the algorithm, let us suppose the opposite: there is a correlation between random numbers that are used to generate the values of some coordinate of two subsequent points examined by the algorithm. To examine any such correlations between elements of the pseudorandom sequences, we propose three tests. *Test 1* selects some dimension and examines the correlation between consecutive pairs of random values used to generate points in this dimension. *Test 2* replicates Test 1 for all dimensions simultaneously. *Test 3* tracks the correlation between consecutive values in the pseudorandom string or, in other words, replicates Test 1 with period 1. Schematically, these tests are explained in Fig. 6, where squares represent consecutive elements of the pseudorandom sequence and loops denote the considered correlations; the length of the loops is constant for each test and referred to as the period of the test.

We apply these three tests to each of two kinds of long sequences, one coming from a true random generator and another from a pseudorandom generator used to produce results in Section 3. Our 'true random' sequence uses data from a reputable online service *random.org* which generates randomness via atmospheric noise picked up by a radio. This service is subject to a battery of daily tests which confirm that it maintains all of the randomness properties claimed [26]. In addition, a series of sequences has been produced via the standard Java generator discussed above for a selection of realistic values of seeds. Lengths of all sequences is set to 100,000 elements. Results of tests 1–3 for these two sequences are shown in Fig. 7, where the period for tests 1 and 2 is set to 65, which is the effective period of our genetic algorithm implementation for population size 5. The period for test 3 is 1, as explained above. Visual inspection does not reveal any significant differences between the true and pseudorandom sequences. Results for other period values and for random sequences with different seeds are of identical nature. *This suggests that our observations of structural bias do not originate from the random generator but rather represent artefacts from the iterative application of algorithmic operators.*

[2] This is also supported by our tests.

[3] in this and the next three formulas, one extra random number is added to account for evaluation of f_0.

[4] For reference, the Marsaglia effect bounds for these effective periods are the following: 41 for 65 dimensions for GA with $N_{pop} = 5, 125$ for 301 dimensions for GA with $N_{pop} = 20, 455$ for 1201 dimensions for GA with $N_{pop} = 100$ and 2221 for 6001 dimensions in all three PSO implementations.



(a) Test 1 for pseudorandom sequence (b) Test 2 for pseudorandom sequence (c) Test 3 for pseudorandom sequence

(d) Test 1 for true random sequence (e) Test 2 for true random sequence (f) Test 3 for true random sequence

Fig. 7. Correlations between elements of pseudorandom sequences of different nature. Results of tests 1–3 for two sequences obtained from pseudorandom (first line) and true random (second line) generators each made up of 100,000 numbers from $[0, 1]$. Period for tests 1 and 2 is set to 65 which is a value of effective period of GA implementation considered in this paper for population size 5; period for test 3 is equal to 1. Visual comparison does not reveal any significant differences between true and pseudorandom sequences. Results for other period values and random sequences with different seeds are of identical nature. This suggests that structural bias cannot originate from random generator but rather represents artefacts from the iterative application of algorithmic operators.

In practice, the situation is slightly more complicated, since for some algorithms (like the genetic algorithm considered in this paper), not all examined points enter the population. This means that some of the dots shown in Fig. 7 are sieved out and others are moved via a series of trivial parallel projections depending on whether or not the point constructed with these dots has entered the population based on its fitness values and particulars of the algorithm. At the current stage of our research, simulations involving such tracking is impractical and deemed of no particular value.

Finally, while investigating known properties of pseudorandom generators, we stumbled upon a good example of a highly (structurally) biased algorithm. In what was a serious attempt by a skilled algorithm designer to design a "super-random" number generator, Donald Knuth came up with *Algorithm K*, which turned out to have unexpected properties [27]. Given a 10-digit decimal number, the algorithm functions as follows[5]:

K1. Choose number of iterations. Set $Y \leftarrow \lfloor X/10^9 \rfloor$, the most significant digit of X (Steps K2–K13 are executed exactly $Y + 1$ times, that is randomizing transformations are applied a random number of times.)
K2. Choose random step. Set $Z \leftarrow \lfloor X/10^8 \rfloor \mathrm{mod} 10$, the second most significant digit of X. Go to step K$(3 + Z)$ (i.e., jump to a random step).
K3. Ensure $\geqslant 5 \cdot 10^9$. If $X < 5,000,000,000$, set $X \leftarrow X + 5,000,000,000$.
K4. Middle square. Replace X by $\lfloor X^2/10^5 \rfloor \mathrm{mod} 10^{10}$.
K5. Multiply. Replace X by $1001001001X \mathrm{mod} 10^{10}$.
K6. Pseudo-compliment. If $X < 100,000,000$, then set $X \leftarrow X + 9,814,055,677$; otherwise set $X \leftarrow 10^{10} - X$.
K7. Interchange halves. $X \leftarrow 10^5(X \mathrm{mod} 10^5) + \lfloor X/10^5 \rfloor$ i.e., interchange the low-order five digits of X with the high-order five digits.
K8. Multiply. Same step as K5.

[5] The pseudocode is given here to demonstrate how complicated the algorithm is; there is no need to follow it in detail.

K9. Decrease digits. Decrease each nonzero digit of the decimal representation of X by one.
K10. 99,999 modify. If $X < 10^5$, set $X \leftarrow X^2 + 99,999$; otherwise set $X \leftarrow X - 99,999$.
K11. Normalize. (At this point X cannot be zero.) If $X < 10^9$, set $X \leftarrow 10X$ and repeat this step.
K12. Modified middle square. Replace X by $\lfloor X(X-1)/10^5 \rfloor \bmod 10^{10}$.
K13. Repeat? If $Y > 0$, decrease Y by 1 and return to step K2. If $Y = 0$, the algorithm terminates with X as the desired "random" value □

Initial tests of this generator revealed that, depending on the starting value, the output of this algorithm is far from being "super-random": it either converges to the 10-digit value 6065038420, or the sequence begins to repeat itself after 7401 values, in a cyclic period of length 3178 [27]. This example strengthens the view that thoughtlessly assembled overcomplicated algorithms have elevated chances of possessing undesirable and intractable properties.

4. Structural analysis of a simplified genetic algorithm

In this section we are going to look at a simplified version of a genetic algorithm, and theoretically analyse this algorithm with a view to uncovering dynamics that may cause structural bias. Our simplifications will allow us to analyse changes in the sample variance of the positions of points in the population when we generate a new point (and replace one from the current population). However, the simplifications, though necessary to facilitate analysis, do not materially change the performance of the algorithm on a function such as f_0, as we explain later with both heuristic arguments and numerical experiments.

Consider a genetic algorithm, as in Section 3.1, with the following amendments to its operation.

1. Selection is uniformly random – i.e. there is a purely random choice of parents;
2. the child replaces a randomly chosen member of the population.

More precisely, we define a process $\{\underline{X}(t)\}_{t\in\mathbb{Z}_+}$, where $\underline{X}(t) \in \mathbb{R}^N$ for each t and $X_i(0)$ is uniformly distributed in $[0,1]$ for each $i = 1,\dots,N$. The change from time t to time $t+1$ is as follows:

- Pick two numbers from 1 to N at random (with replacement). Let these numbers be j and k.
- Generate a new coordinate

$$Y = \left(\min\left(\alpha X_j + (1-\alpha)X_k + Z, 1\right)\right)^+, \tag{2}$$

  where $x^+ = \max(x,0)$, α is a random variable uniformly distributed on $(-d, 1+d)$ for a positive d and Z is a Normal random variable with mean 0 and variance σ^2.
  This represents a choice of a new point which is absorbed at the boundaries.
- Pick a number from 1 to N at random; let this number be i.
- Replace X_i by Y.

Let $S^2(t)$ denote the sample variance of the vector $\underline{X}(t)$

$$S^2(t) = \frac{1}{N-1}\left(\sum X_i(t)^2 - \frac{\left(\sum X_i(t)\right)^2}{N}\right).$$

We can prove the following theorem.

Theorem 1. *If*

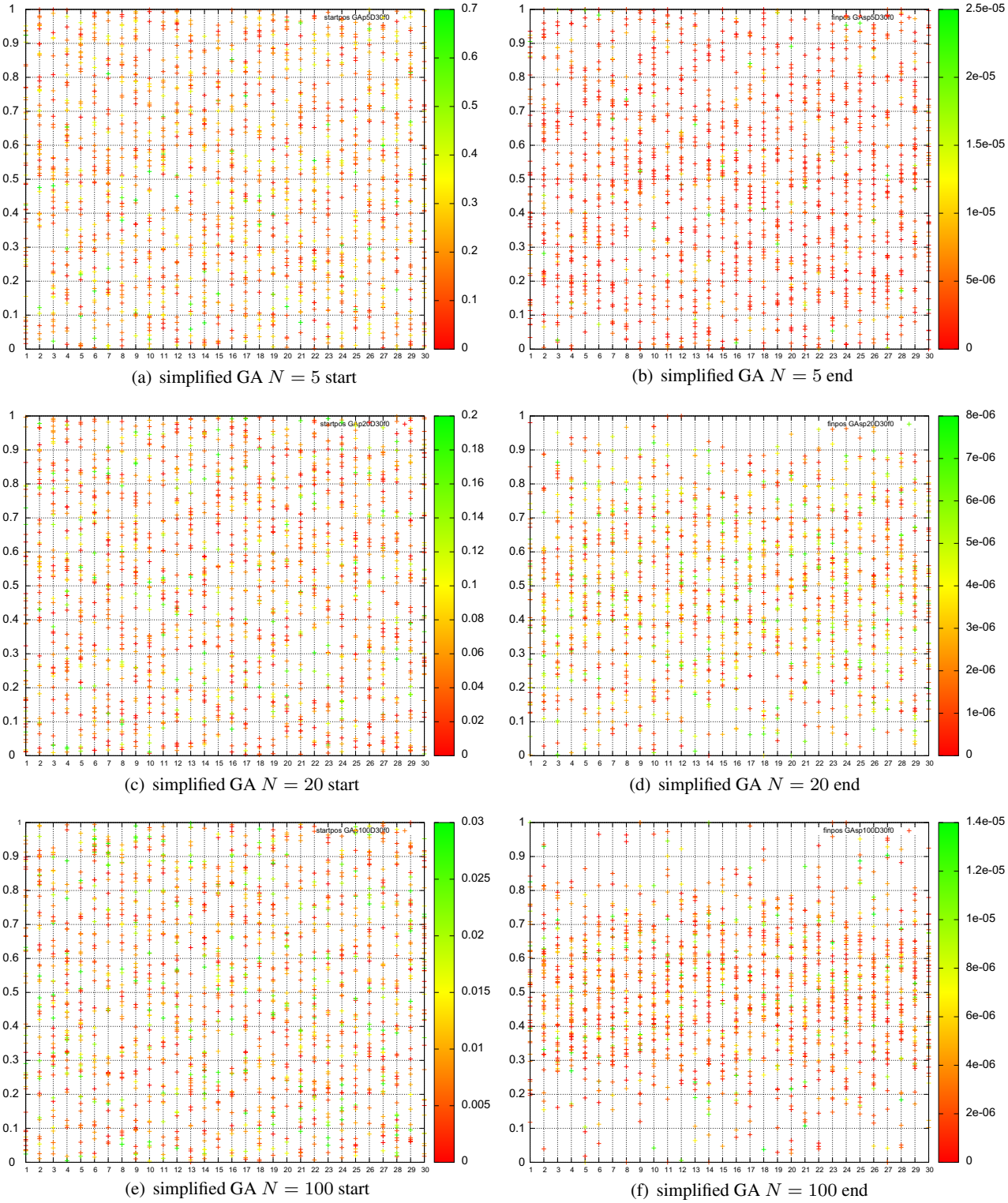
$$d < \frac{-1 + \sqrt{\frac{3N+9}{N-1}}}{2},$$

then there exist $0 < K < \infty$ *and* $\varepsilon > 0$ *such that if* $S^2(t) > K$, *then*

$$\mathbf{E}\left(S^2(t+1) - S^2(t) | \underline{X}(t)\right) < -\varepsilon.$$

We prove the theorem by bounding the sample variance of the real next step by the sample variance of the next step without absorption at the boundaries. Indeed, it is immediate to check that the sample variance of the non-absorbed values at the next step is an upper bound for the sample variance of the original (possibly absorbed) values, and the difference between (non-absorbed) sample values at two subsequent steps is equal to

$$(N-1)\mathbf{E}\left(S^2(t+1) - S^2(t) | \underline{X}(t)\right) = \frac{1}{N^3}\sum_{j,k,l}(S_2 - X_l^2 + \mathbf{E}Y^2) - \frac{1}{N}\mathbf{E}\left(\frac{1}{N^3}\sum_{j,k,l}(S_1 - X_l + Y)^2\right) - S_2 + \frac{1}{N}S_1^2,$$



(a) simplified GA $N = 5$ start (b) simplified GA $N = 5$ end

(c) simplified GA $N = 20$ start (d) simplified GA $N = 20$ end

(e) simplified GA $N = 100$ start (f) simplified GA $N = 100$ end

Fig. 8. Positions of points with the best fitness values in the first (left column) and the last (right column) populations of 50 runs of the *simplified genetic algorithm* for different population sizes in parallel coordinates; horizontal axis shows the number of coordinate, vertical axis shows the range of these coordinate; the fitness value of each point is shown in colour. A clear bias towards the centre of the search space is visible in the last populations as population size increases. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

where Y is defined in (2), $S_2 = \sum X_i(t)^2$ and $S_1 = \sum X_i(t)$. The remainder of the argument consists in re-arranging terms and noting that $\mathbf{E}U = 1/2$ and $\mathbf{E}U^2 = \frac{1+d+d^2}{3}$ □

Note that

$$K = \frac{\sigma^2(1-1/N)}{N+1-2\frac{1+d+d^2}{3}(N-1)}.$$

The theorem implies that if the sample variance of the points' locations is larger than K, then on average it will decrease. This, heuristically, means that the points will tend not to spread over the entire interval $[0,1]$. We conjecture that there is a stronger result showing that points' locations converge to a strict subset of $[0,1]$. This is supported by our numerical results but so far we have not demonstrated it theoretically.

For vectors with N components all taking values in $[0,1]$, it is clear that the largest value of the sample variance is $\frac{N}{2(N-1)}$ and is always bounded away from 0 (in fact, converges to $1/2$ as $N \to \infty$). One can easily see, however, that $K \to 0$ as $N \to \infty$. This means that for a sufficiently large number of points in the population, the range of configurations in which the average change in variance is negative (i.e. configurations from which the points tend to become spread less at the next time instance than at the previous one) is not empty and becomes larger as the number of points increases.

Finally, before closing this section with a brief empirical test of the simplified algorithm that we have analysed, we note certain observations that follow from the theorem, and that we will refer to subsequently. First, numerical exploration of the expression of Theorem 1 with typical and reasonable values suggests that the implied 'reducing variance' dynamics may be commonplace in genetic algorithm designs. Further, and interestingly, as N (population size) increases, the 'burden' on d to be small increasingly relaxes, which suggests that structural bias will become more prominent at larger population sizes, despite perhaps high levels of exploration (larger d) among the algorithm's operators. We note that this expectation, for more prominence in structural bias at higher population sizes, resonates strongly with our empirical findings in Section 3.1. Next, considering that 'difficult' landscapes may tend to keep a population scattered across multiple local optima (at least in early or middle stages of a genetic algorithm's search), the theorem indirectly suggests that the consequent high position variance in such circumstances will exacerbate structural bias. In other words, the theorem provides a theoretical root suggesting that many kinds of 'difficult' landscapes (where we might expect high positional variance during search) will be sensitive to an algorithm's structural bias, while 'easy' landscapes (for which a good algorithm can be expected to focus quickly around optimal areas, with consequent low variance) will be relatively insensitive to an algorithm's structural bias. In Section 5, after first presenting approaches to visualise and quantify structural bias, we perform experiments that allow us to start to evaluate these suggestions.

Numerically, the behaviour on f_0 of the simplified genetic algorithm is very similar to the behaviour of the typical genetic algorithm presented in Section 3.1 as Fig. 8 shows. One can expect this due to simple heuristic arguments. Indeed, given that fitness at every step is chosen from a uniform distribution, independently of the fitness of all other points, and a point will only be accepted if its fitness is better than that of at least one existing point, the fitnesses of all points will converge to the optimal one. Therefore removing a random point instead of the worst one should not strongly influence the performance of the algorithm. The same concerns the choice of parents. Thus, this analysis approximately describes the typical genetic algorithm.

5. Quantifying structural bias and observing its consequences

Returning briefly to analogies, let us consider a football team running trials for a new goal-keeper. Imagine that the final choice is to be made between two persons: one talented but rather lazy keeper who prefers to stand still beside the left goalpost, no matter what the actions of the striker, and one very energetic keeper who can reach every part of the goal but occasionally fails. In this analogy, we intend the goal to represent the problem domain, the goal-keeper plays the role of the algorithm and the strategy of the striker, unknown to the goal-keeper, represents the objective function. It is then the duty of the goal-keeper to locate as close as possible a position where the ball is going to approach the goal, just as the algorithm needs to identify the region of the goal which contains an optimum of the current objective function. In life, it can happen that, by pure luck, the striker is equally limited and can hit only the region of the left goalpost. Clearly, our lazy goalkeeper will have no problem defending the goal from such a striker. However, as it usually happens that strikers tend to target different regions of the goal, a more flexible goalkeeper will end up being a better choice for the team regardless of his or her occasional shortcomings.

We propose, first of all, a simple visual test for structural bias that amounts to visualising the performance of the goalkeeper in such a trial. Our visual test is meant to identify whether or not an algorithm has any structural bias, or to compare the degrees of such bias among a suite of algorithms. In this test, conclusions can be made based on the distribution of coordinates of points with the best fitness values in the final populations of the algorithms under consideration running on f_0 for roughly the same fitness evaluation budget as intended for their deployment on real objective functions.

Application of this visual test to the algorithms explored in Section 3 amounts to observing the parallel coordinates figures presented earlier, thereby inspecting the distributions of the positions of the 50 best points (the best point from each of 50 independent trials), for each of the algorithm configurations. Such inspection suggests that an appreciable level of structural bias is exhibited by the genetic algorithm with population size 100 (Fig. 3(c)), and by PSO with population sizes of both $N = 5$ and $N = 100$ (Fig. 5(a) and (c)). Meanwhile, the genetic algorithm with $N = 20$ exhibits milder structural bias, see Fig. 3(b). The remaining two cases – the genetic algorithm with $N = 5$ and PSO with $N = 20$ – seem to provide satisfactory performance in terms of structural bias, but are clearly more difficult to differentiate objectively based on a purely visual test of Figs. 3(a) and 5(b).

For an objective test of the level of structural bias, we propose use of the Kolmogorov–Smirnov test [28,44,46], specifically to compare the empirical distribution function of the sample of coordinates and the cumulative distribution function of the uniform distribution. By using the Kolmogorov–Smirnov test in this way, we obtain a p-value that expresses the probability that the sample comes from a uniform distribution given the null-hypothesis is correct. Fig. 9 summarises the results of this test, performed independently for each dimension, for the same sets of points discussed earlier in the context of visual tests. These results numerically support our aforementioned conclusions based on visual analysis of Figs. 3 and 5. For example, the cluster of low p-values plotted against "PSOp5f0" corresponds to the observation of high levels of structural bias for PSO with $N = 5$.

Our proposed approach to quantifying an algorithm's inherent structural bias therefore comprises running repeated trials of the algorithm(s) in question using f_0 as the objective function, and subsequently applying one or both of: parallel-coordinates based visual inspection of the final points, and the Kolmogorov–Smirnov test to assess the uniformity of the distribution of those points. The proposed method is computationally highly efficient in comparison to approaches (such as that of [8]) that require numerous optimisation trials with the actual objective function (and also needing to be re-applied for every new objective function of interest). Our proposed approach is potentially suitable as an algorithmic design tool for general use. It is important to note that, on its own, a strategy of maintaining a more even coverage of the search space by the algorithm does not ensure a satisfactory algorithmic design capable of fast convergence to a near-optimum solution. The sole objective of such a strategy is to identify a combination of operators that forces the algorithm to explore the domain with more equal probability. This strategy is therefore complementary and should be used in conjunction with more comprehensive design strategies which ensure other favourable properties of optimisation algorithms such as those discussed in Section 1 or other properties specific to a particular class of algorithms.

5.1. Further numerical results: consequences of structural bias on a suite of test functions

To investigate the consequences of structural bias when one aims to optimise a standard test function, in this subsection we perform experiments using the CEC 2005 test function suite, which is widely used to test and compare algorithms in the field of evolutionary computation [47]. We later consider the results of these experiments in the light of the different levels of inherent bias observed earlier for the same algorithm in our tests on f_0. As we will see, this exercise represents a rehearsal of our proposal that, by observing behaviour on f_0, we are able to better understand the degree to which bias is in effect when the algorithm is applied to a real function; further, as we shall see, our findings (considered also in the light of our theoretical investigation) suggest how we can start to understand the differential effects of inherent structural bias in the presence or absence of different features of the optimisation landscape.

Exact specifications of the CEC 2005 functions can be found in [47]. The benchmark suite comes along with source code that allows users to treat the individual functions in the test suite as 'black box' functions that simply return a fitness value when given an n-dimensional coordinate. No other information is provided to the optimisation algorithm, except for specifications of the range of the search domain. The benchmark suite makes its functions available for specific dimensionalities (e.g. 10, 30 and 50).

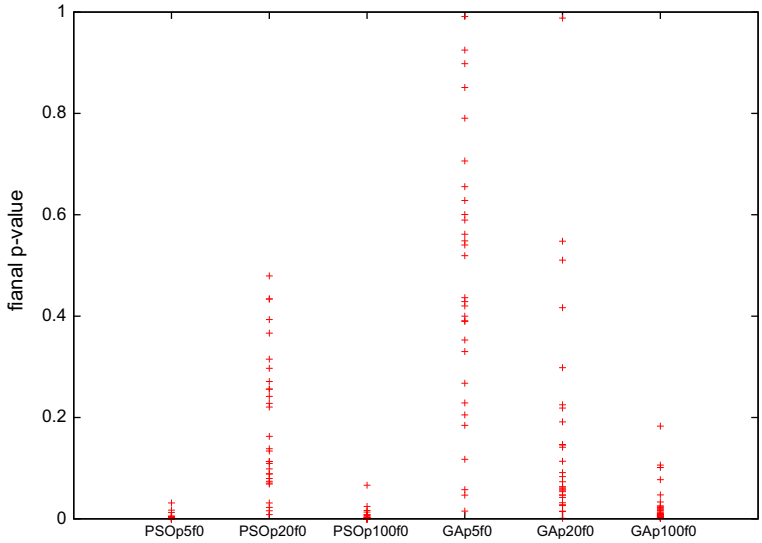
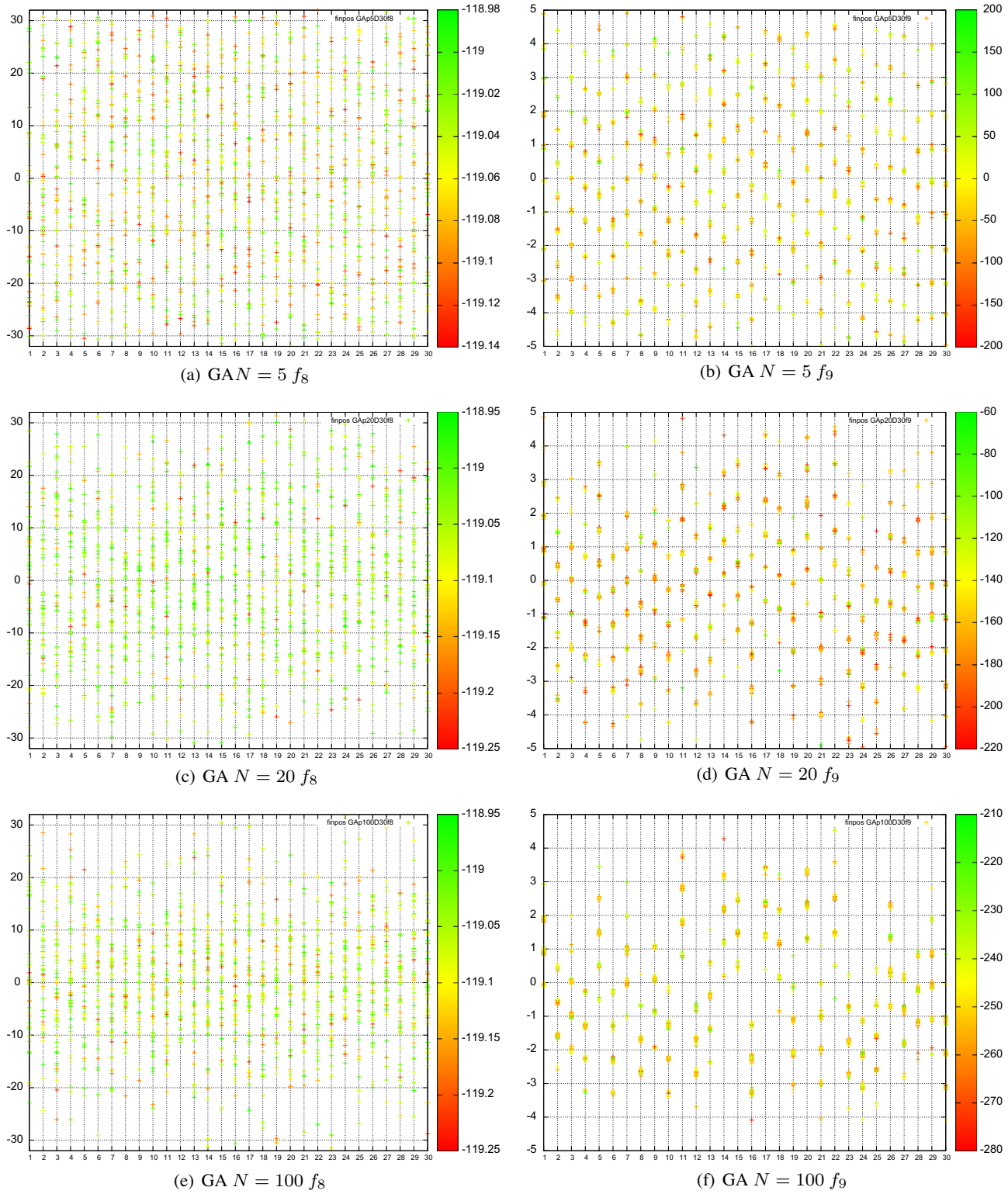


Fig. 9. Results of the Kolmogorov–Smirnov tests for the considered genetic algorithm and PSO, with three population sizes each. For each algorithm, the test is run independently for each dimension of the problem; the p-value returned by each test is shown with a marker. The p-values in the first, third and sixth columns are significantly lower than others, which translates into stronger structural bias present in results for these algorithms. p-values shown in the second column correspond to the case of milder structural bias, meanwhile the fourth and fifth columns characterise algorithms with the weakest structural bias observed in our series of experiments. These results support our conclusions regarding strength of structural bias based on purely visual analysis of Figs. 3 and 5.



(a) GA$N = 5$ f_8 (b) GA $N = 5$ f_9

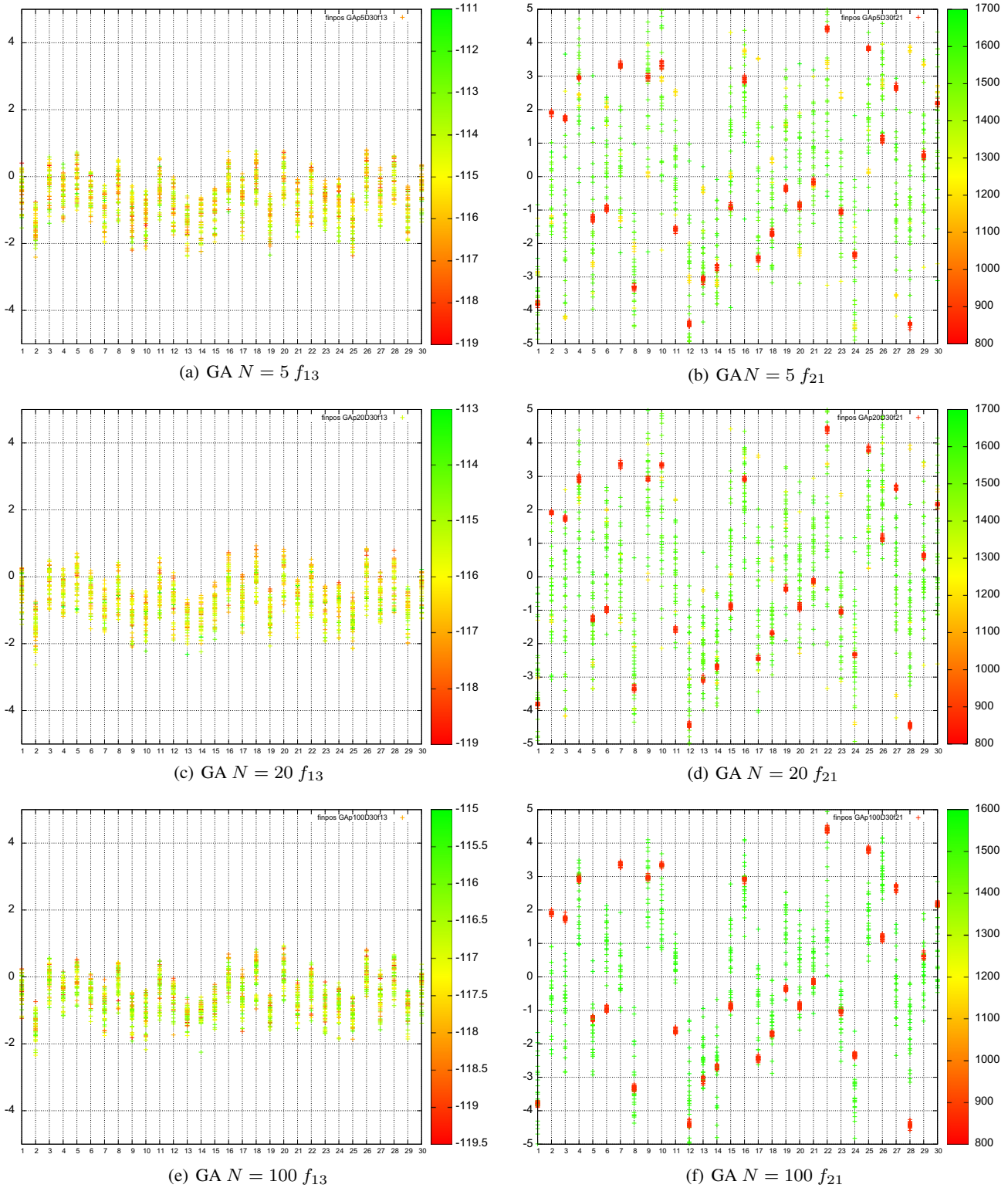
(c) GA $N = 20$ f_8 (d) GA $N = 20$ f_9

(e) GA $N = 100$ f_8 (f) GA $N = 100$ f_9

Fig. 10. Positions of points with the best fitness in the last population of 50 runs of the considered GA for different population sizes in parallel coordinates - *f_8 and f_9 are sensitive to structural bias of GA*. Horizontal axis shows the ordinal number of coordinate, vertical axis shows the full range of domain in this coordinate kept constant for each function; fitness value of each point is shown in colour. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

For the purpose of illustration, we select a limited number of functions from the CEC 2005 benchmark suite for which the genetic algorithm considered in this paper:

- f_8 shifted rotated Ackley function in $[-32, 32]^{30}$ with global optimum on the bounds
- f_9 shifted Rastrigin function in $[-5, 5]^{30}$,



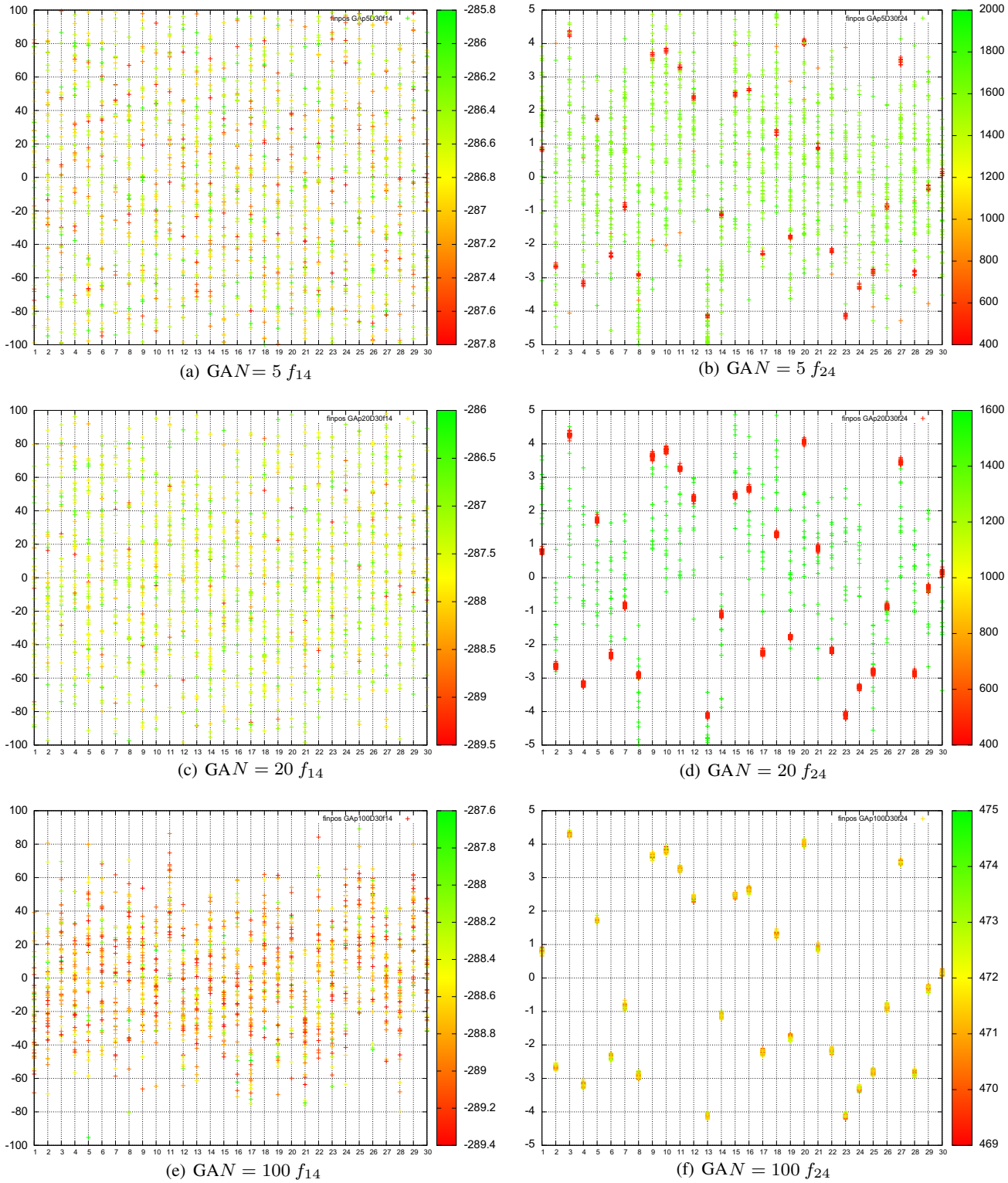
(a) GA $N = 5$ f_{13} (b) GA$N = 5$ f_{21}

(c) GA $N = 20$ f_{13} (d) GA $N = 20$ f_{21}

(e) GA $N = 100$ f_{13} (f) GA $N = 100$ f_{21}

Fig. 11. Positions of points with the best fitness in the last population of 50 runs of the considered GA for different population sizes in parallel coordinates – f_{13} *and* f_{21} *are insensitive to structural bias of GA*. Horizontal axis shows the ordinal number of coordinate vertical axis shows the full range of domain in this coordinate kept constant for each function; fitness value of each point is shown in colour. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

- f_{13} shifted expanded Griewank and Rosenbrock function in $[-5, 5]^{30}$,
- f_{14} shifted rotated expanded Scaffer F6 function in $[-100, 100]^{30}$,
- f_{21} rotated hybrid composition function in $[-5, 5]^{30}$,
- f_{24} rotated hybrid composition function in $[-5, 5]^{30}$.



(a) GA $N = 5$ f_{14} (b) GA $N = 5$ f_{24}

(c) GA $N = 20$ f_{14} (d) GA $N = 20$ f_{24}

(e) GA $N = 100$ f_{14} (f) GA $N = 100$ f_{24}

Fig. 12. Positions of points with the best fitness in the last population of 50 runs of the considered GA for different population sizes in parallel coordinates – f_{14} *and* f_{24} *are highly sensitive to structural bias of GA.* Horizontal axis shows the ordinal number of coordinate, vertical axis shows the full range of domain in this coordinate kept constant for each function; fitness value of each point is shown in colour. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

Parallel coordinates visualisations of our genetic algorithm's results on these functions are shown in Figs. 10–12. Each individual plot summarises the results of 50 independent trials of the genetic algorithm on the function concerned, by showing, in parallel coordinates fashion, 50 30-dimensional points, comprising the best point reached in each trial. It is important to stress that interpretation of these figures should be *entirely different* from interpretation of those shown in Figs. 3 and 5. Unless specifically constructed so, it is not expected that the final distribution of positions of the best points in the final

generation is close to uniform in the case of any function other than f_0. In sharp contrast to the f_0 results, we would naturally expect in these plots to see a strong effect due to the combined activity of selection and the shape of the landscape, resulting in the identification of regions at or close to optima of the objective function.

At any point in time during the optimisation process, two forces can be conceptualised which simultaneously act on the population – landscape bias and structural bias. The first force pulls the population towards better values of the objective function, meanwhile the second force can be thought of as pulling the population towards 'attractors' in the domain (perhaps complex attractors) whose nature arises from the combination of algorithm design choices. Both of these forces are unknown and, therefore, their sum – which defines population movement – is also unknown. The use of f_0 to help quantify structural bias is precisely based on the idea of eliminating one of the unknowns, the 'landscape force', hence revealing any structural bias. It follows that, to interpret the visualisations of Figs. 10–12, we can proceed as follows. For a given objective function, we can observe how the distribution of final points varies as a function of the algorithm configurations considered, and consider how this correlates with the relative degrees of structural bias previously observed (via experiments with f_0) over the same set of configurations. Obviously, attention should also be paid to the final attained values of the objective function and their variances.

In such analysis of results over the CEC 2005 benchmark suite, we have observed three types of behaviour, which we conceptualise as resulting from the combination of structural bias in the algorithm configuration itself combined with more or less sensitivity to that bias inherent in the objective function at hand:

- *Sensitivity* to structural bias, as exemplified by f_8 and f_9, see Fig. 10. For f_8, all three series of runs attained similar values of final fitness over 50 runs, but runs with larger populations failed to find good solutions closer to the boundary of the domain. We attribute such failures to structural bias of the genetic algorithm, as also observed for $N = 50$ on f_0. On f_8, the genetic algorithm exhibits behaviour overall similar to the case of f_0. Meanwhile, for f_9, final fitness values are quite different across the three series of runs, but the variance of positions of final best points demonstrates the pattern of sensitivity to structural bias.
- *Insensitivity* to structural bias, as exemplified by f_{13} and f_{21}, see Fig. 11. All parameter settings considered lead to similar results in terms of the fitness values attained, positions of final best points and variances in their positions.
- *High sensitivity* to structural bias, as exemplified by f_{14} and f_{24}, see Fig. 12. In the case of f_{14}, quite similar values of final fitness are attained over the three series of runs; however, drastic changes are clear from one series to another in terms of variances of positions of final best points. For $N = 5$, these points fill the whole domain and better points, indicated on the figure with red markers, are uniformly spread out across the domain. The situation is to some extent similar for $N = 20$ but all points start to shift towards the middle of the interval and those with better fitness values in particular. For $N = 100$, no final best points are located in the outer regions of the domain, but their distribution in the centre of the domain is rather uniform. As for f_{24}, there are drastic changes both in terms of final fitness values and distribution of positions of final best points. It is interesting to note that for the genetic algorithm with $N = 100$ on f_{24}, it is rather easy to find a region with low fitness values consistently over the series of 50 runs suggesting that this particular function possesses a special property of some kind.

As regards other functions from the CEC2005 suite, it is worth mentioning that functions in the top of the list tend to be less sensitive to the structural bias of our genetic algorithm. These functions are known to be unimodal or close to unimodal [47]. This observation aligns well with our speculation in Section 4, and suggests that the theoretical analysis of the simplified genetic algorithm may have captured at least part of the essence of the factors that underpin structural bias and also the sensitivity to structural bias of any given objective function (via informed expectations of how the landscape may affect population variance). However, we are of course very much only at the beginning of a theoretical understanding of structural bias, in terms of both underpinning causes and of the effects of particular landscapes. This state of affairs goes hand in hand with a need for approaches to investigate and quantify inherent structural bias, such as proposed in this paper. Returning to the relative sensitivity to structural bias of different objective functions, we speculate that further work, involving analyses of particular collections of objective functions, might reveal similarities in the structure of basins of attraction in the landscapes might correlate with similarities in sensitivity. Empirically, we have seen that evolving populations seems to be less "confused" by structural bias in stronger regions of attraction which characterise unimodal optimisation as opposed to a weaker pull from multiple closer regions of attraction in the multimodal situation. This also points towards similarities between the effects of structural bias and noise in the objective function. Just as a noisy objective function induces false optima in the landscape, structural bias deceptively pushes the evolving population towards regions potentially unremarkable in terms of objective function values.

6. Conclusions and discussion

6.1. Summary and main conclusions

A vast body of research in the field of population-based optimisation algorithms deals with efficient exploitation of information already contained within the population, while little attention is paid to investigation of whether or not a

specific combination of algorithmic operators and algorithm strategies is actually capable of reaching all parts of the search space with equal efficiency. When an algorithm is *not* capable in the latter sense – favouring certain areas of the search space over others – independently of the fitness function, it is exhibiting 'structural bias'.

In this paper we have argued, from both theoretical and empirical standpoints, that structural bias is likely to be common in practice, and amplified when we would least expect it (when we increase the population size in hope of a more exploratory search) and when it may cause most damage (on 'difficult' problems). In this paper, in order to empirically investigate the structural bias of an algorithm, we neutralised the role of the optimisation landscape thereby isolating the role of any inherent bias in the algorithm design – by performing experiments with f_0 (as defined in Section 2). This enabled us, by visualising the results of multiple experiments, to observe structural bias in terms of its effects on the distribution of the final best points. Since we would expect these distributions to be uniform in the absence of structural bias, the pattern of observed non-uniformity, in combination with other considerations, can be taken as informative of the structural bias inherent in the optimisation algorithm.

To some extent, one can posit an alternative explanation for such effects by appeal to undesirable properties of the pseudo-random number generator used. Such an objection can be difficult to discount, since the effects of the complex ways in which pseudo-random number generators are indeed non-random are extremely difficult to predict. Nevertheless, our analysis in Section 3.5, coupled with our theoretical findings and the pattern of empirical results, suggest that this approach reveals structural bias rather than pseudo-random artefacts.

Our approach to revealing and quantifying structural bias is easily replicated, and we recommend its use prior to finalising the parametric and design configuration of any optimisation algorithm to be deployed on real-world problems. Such can be seen as an additional validation step, coupled with other investigations of the algorithm design which would normally be done to reveal the configuration that provides the best and/or fastest solutions to the class of problems of interest. For example, if the best algorithm configuration also has minimal or no structural bias, then all is well. However this may very often not be the case. When optimisation is used for system identification problems (for example, determining the parameters of a function or model that best fit a set of empirically obtained points), it is usually deemed important to find, as far as possible, all good solutions, or a fair representation thereof. Finding the true system parameters in such problems is always potentially confounded by factors such as noisy or insufficient empirical data or a poorly chosen fitness function. But, if the otherwise preferred algorithm configuration exhibits structural bias, this will further exacerbate the possibility that the true system parameters may not be uniformly accessible to the algorithm.

We have also contributed a theoretical argument that partially explains how structural bias can arise in a simple population-based algorithm. The analysed algorithm is simplified, but exhibits the primary strategies common to almost all population based optimisation algorithms, including a parameter d that controls the degree of exploration induced by the variation operator – the larger d, the higher the chance and extent to which a new sample will extend beyond the region of search space occupied by its parents. By performing experiments with a simple genetic algorithm on both f_0 and on the functions of the CEC2005 test suite, we were able to link our observations of inherent structural bias with our theoretical results. In particular, we found evidence that increasing the population size, and increasing problem difficulty, both (independently) correlated with increased evidence of bias; both of these findings can be predicted as a consequence of the dynamics set out in Theorem 1. One interesting area of future work would be to replicate such experiments using a selection of state of the art optimisation algorithms from varying styles of population-based search (for example, memetic search [35], differential evolution [39], or advanced PSO [31], thereby gaining understanding of the inherent biases in such varied configurations of strategy/operator recipes.

6.2. Discussion and speculations

The crucial step in the argument leading to our theoretical result for GA is to show that, on average, and under certain conditions, the population variance will decrease with time, despite the clear opportunities for search to extend beyond the current locations of the population. If we consider a truly random algorithm *RA* in such circumstances, in which each new sample is generated uniformly at random in the search space and replaces a randomly chosen previous sample, we can expect unbiased coverage of the search space and maintenance of a constant variance over time, which (under the conditions of the theorem) would be $\frac{1}{12}$. For typical choices of parameters ($\sigma^2 = 0.1, N = 50, d = 0.2$), the value of K in the theorem is much lower than $\frac{1}{12}$, suggesting that such an algorithm will rarely be able to maintain the levels of exploration required to eliminate structural bias without careful design. Algorithm *RA* exhibits 'pure' exploration, however any effective optimisation algorithm incorporates exploitation, which is invariably achieved by biassing samples towards the regions of previously visited points. The 'reducing variance' theorem suggests that such exploitation is intimately related to the emergence of structural bias, but it also suggests that the latter can be controlled by reducing the population size, or by raising d (or, alternatively, by revisiting the algorithm's design to introduce mechanisms that introduce additional new samples in a way that is not tied to the locations of previous samples). By raising d, we (usually) increase the likelihood of structural bias but reduce the efficiency of exploitation; meanwhile, by reducing the population size we reduce the likelihood of structural bias but reduce the level of exploration.

The inverse relationship between structural bias and population size (strictly in the context of standard and simple genetic algorithms, which was the substrate of our theoretical analysis) that is at first counter-intuitive – to increase the population size would seem to inject more diversity, which we should expect to alleviate such bias. However, we believe

this phenomenon can be explained by an effect akin to 'preferential attachment' in the evolution of complex systems. Structural bias, manifested as the concentration of search progress in ever narrower regions, builds on initial seed areas which begin to attract further points. In our context, if two parent solutions happen to be close together, their offspring will stay nearby and increase the density in this region, and the positive feedback dynamics of this process will exacerbate the non-uniform distribution. When the population size is increased, there is more opportunity for such initial seeds to be present.

The results we have presented seem less surprising when iterative population-based algorithms are considered in relation to *Iterated Function Systems* which, by definition, are finite sets of mappings of complete metric space or, symbolically, $\mathcal{F} = (\mathbb{X}, f_1, f_2, \ldots, F_M), f_m : \mathbb{X} \to \mathbb{X},\ m = 1, 2, \ldots,\ M$. Depending on the properties of functions that make them up, IFSs exhibit a variety of behaviours. According to the collage theorem [3], for any given set/image there exists a strictly contractive IFS whose attractor arbitrarily closely approximate this set/image. A linear IFS on $\mathbb{R}^n$ has a unique attractor located at the origin [4]. Any projective IFS has at most one attractor [5] but behaviour of such attractors appears to be more complicated than in the case of affine IFSs as they might not depend continuously on parameters [5]. Moreover, there are examples of non-contractive projective IFS with an attractor [5]. These results point to a potentially fruitful direction for the analysis of algorithms through studying the properties of their operators.

Finally, it is instructive to speculate on the existence of structural bias in combinatorial optimisation. Investigations in this article are pinned to the context of real-valued vector optimisation. At first sight, it is not at all obvious that structural bias may occur in the combinatorial case. However, it is trivial to see that it *could* occur. For example, were we so inclined, we could purposely design operators to favour certain regions of the space independently of fitness. Imagine, say, a permutation space with an even number of objects, in which the only operator in use was to swap an item with its neighbour two steps away; this search is then confined to the cross-product of two subspaces, omitting most of the permutation space. Also, despite the 'real-value' focus of the theoretical argument, it is intuitively reasonable to speculate that a similar argument, couched in terms of suitable metrics, may be meaningful for combinatorial spaces. For example, the perturbation effect of a combinatorial operator on one or more points can be characterised as a distribution of edit distances from those points. Structural bias in combinatorial search algorithms might arise from the dynamics of the variance of this distribution in the context of other aspects of the algorithm's configuration.

References

[1] Brief Description of the Crypt-x Tests, 2000. <http://www.isrc.qut.edu.au/resource/cryptx/tests.php>.
[2] P.J. Angeline, Using selection to improve particle swarm optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation CEC 1998, Anchorage, Alaska, USA, 1998.
[3] M. Barnsley, Fractals Everywhere, Academic Press, 1988.
[4] M. Barnsley, A. Vince, The eigenvalue problem for linear and affine iterated function systems, Linear Algebra Appl. 435 (12) (2011) 3124–3138.
[5] M.F. Barnsley, A. Vince, Real projective iterated function systems, J. Geom. Anal. 22 (4) (2012) 1137–1172.
[6] G. Briscoe, P.D. Wilde, Physical complexity of variable length symbolic sequences, Physica A 390 (21–22) (2011) 3732–3741.
[7] E. Burke, S. Gustafson, G. Kendall, Diversity in genetic programming: an analysis of measures and correlation with fitness, IEEE Trans. Evol. Comput. 8 (1) (2004) :47–62.
[8] M. Davarynejad, J. van den Berg, J. Rezaei, Evaluating center-seeking and initialization bias: the case of particle swarm and gravitational search algorithms, Inf. Sci. (2014).
[9] M.A.M. de Oca, T. Stützle, M. Birattari, M. Dorigo, Frankenstein's PSO: a composite particle swarm optimization algorithm, IEEE Trans. Evolut. Comput. 13 (5) (2009) :1120–1132.
[10] M. d'Ocagne, Coordonnées Paralléles et Axiales: Méthode de transformation géométrique et procédé nouveau de calcul graphique déduits de la considération des coordonnées parallélles, Gauthier-Villars., Paris, 1885.
[11] A. Eiben, C. Schippers, On evolutionary exploration and exploitation, Fundam. Inform. 35 (1–4) (1998) 35–50.
[12] A.E. Eiben, E.H.L. Aarts, K.M.V. Hee, Global convergence of genetic algorithms: a Markov chain analysis, in: Proceedings of the 1st Conference on Parallel Problem Solving from Nature (PPSN-I), Lecture Notes in Computer Science, Springer, 1991, pp. 4–12.
[13] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computation, Springer-Verlag, Berlin, Germany, 2003.
[14] D.K. Gehlhaar, D.B. Fogel, Tuning evolutionary programming for conformationally flexible molecular docking, in: L. Fogel, P. Angeline, T. Back (Eds.), Evolutionary Programming V: Proceedings of the Fifth Annual Conference on Evolutionary Programming, MIT Press, Cambridge, MA, 1996, pp. 419–429.
[15] F. Glover, Tabu search part I, ORSA J. Comput. 1 (1989) 190–206.
[16] F. Glover, Tabu search part II, ORSA J. Comput. 2 (1990) 4–32.
[17] D.E. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.
[18] D. Gupta, S. Ghafir, An overview of methods maintaining diversity in genetic algorithms, Int. J, Emerg. Technol. Adv. Eng. 2 (2012) 56–60.
[19] C.B. Hurley, R.W. Oldford, Pairwise display of high-dimensional information via eulerian tours and hamiltonian decompositions, J. Comput. Graphical Statist. 19 (4) (2010) 861–886.
[20] M. Hutter, S. Legg, Fitness uniform optimization, IEEE Trans. Evol. Comput. 10 (2006) 568–589.
[21] A. Inselberg, The plane with parallel coordinates, Visual Comput. 1 (1985) 69–91.
[22] A. Inselberg, Parallel coordinates, Visual Multidimensional Geometry and its Application, Springer, 2008.
[23] S. Janson, M. Middendorf, On trajectories of particles in PSO, in: IEEE Swarm Intelligence Symposium, SIS 2007, 2007, pp. 150–155.
[24] J. Kennedy, Some issues and practices for particle swarms, in: IEEE Swarm Intelligence Symposium SIS 2007, 2007, pp. 162–169.
[25] J. Kennedy, R.C. Eberhart, Particle swarm optimization, in: Proceedings of IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.
[26] C. Kenny, Random number generators: an evaluation and comparison of random.org and some commonly used generators, in: Technical Report, The Distributed Systems Group, Computer Science Department, Trinity College Dublin, 2005.
[27] D. Knuth, Seminumerical algorithms, The Art of Computer Programming, 2nd ed., vol. 2, Addison Wesley, 1982.
[28] A.N. Kolmogorov, Sulla determinazione empirica di una legge di distribuzione, Giornale dell'Istituto Italiano degli Attuari 4 (1933) 83–91.
[29] A.V. Kononova, K.J. Hughes, M. Pourkashanian, D.B. Ingham, Fitness diversity based adaptive memetic algorithm for solving inverse problems of chemical kinetics, in: Proceedings of the IEEE Congress on Evolutionary Computation, 2007, pp. 2366–2373.
[30] P. L'Ecuyer, Tables of linear congruential generators of different sizes and good lattice structure, Math. Comput. 68 (225) (1999) 249–260.
[31] X. Li, X. Yao, Cooperatively coevolving particle swarms for large scale optimization, IEEE Trans. Evol. Comput. 16 (2) (2012).

[32] M. Lozano, F. Herrera, J.R. Cano, Replacement strategies to preserve useful diversity in steady-state genetic algorithms, Inf. Sci. 178 (23) (2008) 4421–4433.
[33] G. Marsaglia, Random numbers fall mainly in the planes, Proc. Nat. Acad. Sci. 61 (1) (1968) 25–28.
[34] G. Marsaglia, Diehard Battery of Statistical Tests, 1995. <http://stat.fsu.edu/geo/diehard/>.
[35] D. Molina, M. Lozano, C. García-Martínez, F. Herrera, Memetic algorithms for continuous optimization based on local search chains, Evol. Comput. 18 (1) (2010) 27–63.
[36] C.K. Monson, K.D. Seppi, Exposing origin-seeking bias in PSO, in: GECCO'05, 2005, pp. 241–248.
[37] F. Neri, Diversity managements in memetic algorithms, in: F. Neri, C. Cotta, P. Moscato (Eds.), Handbook of Memetic Algorithms, Studies in Computational Intelligence, vol. 379, Heidelberg, Berlin, Heidelberg, 2012, pp. 153–165.
[38] F. Peng, K. Tang, G. Chen, X. Yao, Population-based algorithm portfolios for numerical optimization, IEEE Trans. Evol. Comput. 14 (5) (2010) :782–800.
[39] A.P. Piotrowski, Adaptive memetic differential evolution with global and local neighborhood-based mutation operators, Inf. Sci. 241 (2013) 164–194.
[40] A.K. Qin, V.L. Huang, P.N. Suganthan, Differential evolution algorithm with strategy adaptation for global numerical optimization, IEEE Trans. Evol. Comput. 13 (2) (2009) :398–417.
[41] G. Rudolph, Convergence of evolutionary algorithms in general search spaces, in: Proceedings of IEEE International Conference on Evolutionary Computation, 1996, pp. 50–54.
[42] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, A statistical test suite for random and pseudorandom number generators for cryptographic applications, in: Technical Report, National Institute of Standards and Technology, 2010.
[43] B. Russell, Logical atomism, in: J. Muirhead (Ed.), Contemporary British Philosophy, Allen and Unwin, London, 1924.
[44] N.V. Smirnov, Approximate distribution laws for random variables, constructed from empirical data, Uspekhi Matematicheskih Nauk 10 (1944) 179–206 (in Rusian).
[45] W.M. Spears, D.T. Green, D.F. Spears, Biases in particle swarm optimization, Int. J. Swarm Intell. Res. 1 (2) (2010) 34–57.
[46] M.A. Stephens, Edf statistics for goodness of fit and some comparisons, J. Am. Statist. Assoc. 69 (347) (1974) 730–737.
[47] P.N. Suganthan, N. Hansen, J.J. Liang, K. Deb, Y.P. Chen, A. Auger, S. Tiwari, Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization, in: Technical Report, Nanyang Technological University, Singapore and KanGAL Report Number 2005005 (Kanpur Genetic Algorithms Laboratory, IIT Kanpur), 2005.
[48] F. van den Bergh, A.P. Engelbrecht, A study of particle swarm optimization particle trajectories, Inf. Sci. 176 (8) (2006) 937–971.
[49] M. Črepinšek, S.-H. Liu, M. Mernik, Exploration and exploitation in evolutionary algorithms: a survey, ACM Comput. Surv. 45 (3) (2013) 35:1–35:33.
[50] J. Walker, Ent Program, 2008. <http://www.fourmilab.ch/random/>.
[51] P.H. Winston, Artificial Intelligence, Addison-Wesley, 1984.
[52] S.-Y. Yuen, C.K. Chow, A non-revisiting genetic algorithm, in: IEEE Congress on Evolutionary Computation CEC 2007, 2007, pp. 4583–4590.
[53] S.-Y. Yuen, C.K. Chow, Continuous non-revisiting genetic algorithm, in: IEEE Congress on Evolutionary Computation CEC 2009, 2009, pp. 1896–1903.
[54] S.-Y. Yuen, S.W. Leung, Genetic programming that ensures programs are original, in: IEEE Congress on Evolutionary Computation CEC 2009, 2009, pp. 860–866.
[55] H. Zhang, Y. Zhu, H. Chen, Root growth model: a novel approach to numerical function optimization and simulation of plant root system, Soft. Comput. 18 (3) (2014) 521–537.
[56] H. Zhou, X. Yuan, H. Qu, W. Cui, B. Chen, Visual clustering in parallel coordinates, Comput. Graphics Forum 27 (3) (2008) 1047–1054.