JYVÄSKYLÄ STUDIES IN COMPUTING
194

Ilpo Poikolainen

Simple Memetic Computing Structures for Global Optimization

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 194

Ilpo Poikolainen

Simple Memetic Computing Structures for Global Optimization

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi Mattilanniemen D-rakennuksen salissa MaD259
syyskuun 18. päivänä 2014 kello 14.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in Mattilanniemi, hall MaD259, on September 18, 2014 at 14 o'clock.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2014

Simple Memetic Computing Structures for Global Optimization

JYVÄSKYLÄ STUDIES IN COMPUTING 194

Ilpo Poikolainen

Simple Memetic Computing Structures for Global Optimization

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2014

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

URN:ISBN:978-951-39-5803-9
ISBN 978-951-39-5803-9 (PDF)

ISBN 978-951-39-5802-2 (nid.)
ISSN 1456-5390

Copyright © 2014, by University of Jyväskylä

Jyväskylä University Printing House, Jyväskylä 2014

ABSTRACT

Poikolainen, Ilpo
Simple Memetic Computing Structures for Global Optimization
Jyväskylä: University of Jyväskylä, 2014, 60 p.(+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 194)
ISBN 978-951-39-5802-2 (nid.)
ISBN 978-951-39-5803-9 (PDF)
Finnish summary
Diss.

During the recent years, Memetic Computing (MC) and Memetic Algorithms (MAs) have been drawing increasing attention in the scientific community. While MAs, by classic definition, include a population based algorithm and a local search, MC structures include several algorithmic components (memes) within a co-operative framework. Using several algorithmic components is preferable, over a single component, when wisely selected to complement each other. Along with the selection of these components, the designer needs to define a memetic structure, which determines how the components interact during the optimization process. Defining this structure is crucially important in order to achieve a robust algorithmic framework with good balance between exploration and exploitation. This thesis analyzes MC structures, focusing on the concept of simplicity, and understanding the role of each component.

Several MC structures are presented in the included articles. First, a simple memetic structure is studied in depth, and its performance is enhanced by modified variants. This simple structure contains a resampling mechanism inspired by the exponential crossover of Differential Evolution (DE). The resulting algorithms retain the properties of the original implementation, in terms of simplicity and memory requirements, while remaining competitive against more complex state-of-art algorithms.

Differential Evolution is studied in depth as some novel MC structures designed in this thesis are based on a DE logic. DE is a versatile optimization algorithm which can be applied to a wide range of problems. The overall simple structure is achieved by adding components and/or modifying operators from the original DE scheme. Finally, motivated by the philosophy that the role of each part of an algorithm should be clear to the designer and the algorithm should be tailored around the problem features, a novel DE based MC scheme is introduced. The MC scheme estimates the multimodality of an optimization problem, and detects the most interesting areas of the decision space in order to intelligently guide the initial population sampling for DE.

Keywords: Memetic Computing, Differential Evolution, Evolutionary Algorithms, Memetic Algorithms, Memetic Structures, Local Search

Author

Ilpo Poikolainen
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Supervisor

Ferrante Neri
Department of Mathematical Information Technology
University of Jyväskylä
Finland

Reviewers

Dr. Swagatam Das
Electronics and Communication Sciences Unit
Indian Statistical Institute
India

Prof.Dr. A.E. (Gusz) Eiben
Head of the Computational Intelligence Group
Department of Computer Science
Faculty of Sciences
Vrije Universiteit Amsterdam
Netherlands

Opponent

Giovanni Acampora
College of Arts and Science, School of Science & Technology
Nottingham Trent University
United Kingdom

ACKNOWLEDGEMENTS

My first thanks go to my supervisor Ferrante Neri without whom this work would not exist. I am especially thankful for resourceful discussions on research ideas, problems, and being available when ever, despite the distance. Thank you.

I also want to thank my former assistant supervisor Matthieu Weber, who guided me through the very basics and introduced me to optimization techniques, eventually resulting to this work. I wish to express my gratitude to Ernesto Mininno, Giovanni Iacca, and Fabio Caraffini for their work towards optimization platform Kimeme and from the assistance they provided various times over different cases and the work towards articles included in this dissertation.

For reviewing the dissertation manuscript I wish to thank Prof.Dr. A.E. (Gusz) Eiben and Dr. Swagatam Das. I am also very grateful to Giovanni Acampora for agreeing to be my opponent. Further acknowledgements needs to be given to the Faculty of Information Technology and the employees for all the support they provided. Finally I want to thank all of my family, colleagues, and friends for the assistance and discussions.

LIST OF FIGURES

LIST OF ALGORITHMS

CONTENTS

LIST OF INCLUDED ARTICLES

PI I. Poikolainen, G. Iacca, F. Neri, E. Mininno, M. Weber. Shrinking Three Stage Optimal Memetic Exploration. *Proceedings of the fifth international conference on bioinspired optimization methods and their applications, pages 61-74*, 2012.

PII I. Poikolainen, F. Caraffini, F. Neri, M. Weber. Handling Non-Separability in Three Stage Memetic Exploration. *Proceedings of the fifth international conference on bioinspired optimization methods and their applications, pages 195-205*, 2012.

PIII F. Neri, M. Weber, F. Caraffini, I. Poikolainen. Meta-Lamarckian Learning in Three Stage Optimal Memetic Exploration. *12th UK Workshop on Computational Intelligence (UKCI), pages 1-8*, 2012.

PIV I. Poikolainen, G. Iacca,F. Caraffini, F. Neri. Focusing the search: a progressively shrinking memetic computing framework. *Int. J. Innovative Computing and Applications, pages 3-16*, 2013.

PV F. Caraffini, F. Neri, I. Poikolainen. Micro-Differential Evolution with Extra Moves Along the Axes. *IEEE Symposium on Differential Evolution (SDE), pages 46-53*, 2013.

PVI I. Poikolainen, F. Neri. Differential Evolution with Concurrent Fitness Based Local Search. *IEEE Congress on Evolutionary Computation (CEC), pages 384-391*, 2013.

PVII I. Poikolainen, F. Neri, F.Caraffini. Cluster-Based Population Initialization for Differential Evolution Frameworks. *submitted in April*, 2014.

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

In articles [PI] and [PII] modified versions of Three Stage Optimal Memetic Exploration (3SOME) algorithm are proposed. For these articles the author contributed in design and implementation of the proposed algorithms, while original algorithm and other comparison algorithms were implemented by co-authors. Numerical experiments and statistical tests related to both articles were performed by the author. Writing of the articles was carried out together with co-authors.

Article [PIII] proposes alternative memetic structure for coordination of components in 3SOME algorithm. For article [PIII] the author contributed in the design of memetic structure while implementation of the structure and comparison algorithms were done by co-authors. Numerical experiments were carried by author while statistical tests regarding comparison algorithms and most of the writing of the article was performed by co-authors.

Article [PIV] is revised and extended version of article [PI]. Additional numerical experiments and statistical test were performed by author, while co-authors contributed to writing of the article and experiments regarding computational overhead.

In article [PV] parallel structure of micro-Differential Evolution with Extra Moves Along the Axes (μDEA) is proposed. Author contributed in design of the μDEA, while majority of the implementation was performed by co-authors. Author participated in performing numerical experiments and statistical tests. Majority of the writing of the article was done by co-authors.

Article [PVI] proposes a novel memetic algorithm, namely Differental Evolution with Concurrent Fitness Based Local Search (DEcfbLS). In this article, both the algorithmic design and implementation were performed by author. Majority of the writing was done by the author with occasionally assisted by the co-author. Statistical tests performed, for both of those included in the article and those sent to organizers of CEC2013 competition were performed by the author.

Finally article [PVII] introduces new module for Differential Evolution (DE) based algorithms, namely Clustering-Based Population Initialization (CBPI). The proposed module is tested with classical DE and 5 modern variants of DE. Original idea behind approach came from co-authors while final design of clustering and population initialization were put together by the author. Each of modified versions of DE were implemented by the author and numerical results for BBOB2010, CEC2013, and CEC2010 benchmarks were performed by the author, while numerical results regarding real-world problem were done by co-authors. Author contributed to the writing of the article for numerical results section and miscellaneous parts of other sections.

Many of benchmark problems and comparison algorithms included in all of the articles were implemented by co-authors within Kimeme optimization platform [13], which I am greatly thankful for.

1 INTRODUCTION

An optimization process can be described as trying to find best possible answer for a given problem. These problems can be categorized into two different classes of problems: combinatorial problems and continuous problems. An example of combinatorial problem is a Nesting Problem, where one tries to minimize the wasted space when filling given container with predefined pieces (this problem is very familiar e.g. in clothing industry). An example of continuous problem could be designing an airfoil with best aerodynamic qualities. More formally optimization problem is finding *solution* $x* \in D$ for a given *problem* $f : D \to E$, such that $f(x*) < f(x), \forall x \in D$. It must be noted that sign $<$ used here means "better" and $D \subset \mathbf{R}^n$ is a *design space* or a *search space*, $E \subset \mathbf{R}^m$ is a *result space* and f is an *objective function* or a *fitness function*. If $m > 1$ the optimization problem is called *multi-objective* or *many-objective* optimization problem. In this work we only consider *single-objective* problems where $m = 1$. More specifically, we focus on objective functions

$$f : \mathbf{R}^n \to \mathbf{R} \tag{1}$$

and trying to find optimum $x^* \in \mathbf{R}^n$, such that

$$f(x^*) = \min f \tag{2}$$

For clarity: In future chapters we assume that we are working with minimization problems and thus refer function's minimum as optimum unless otherwise mentioned. In general it is trivial to convert any maximization problem to minimization problem (and vice versa).

This thesis is organized as follows. The present Chapter 1 introduces some classical methods for local and global optimization. Chapter 2 presents some of most popular single-solution and population-based meta-heuristics for global optimization and also discusses combinations of meta-heuristics called memetic algorithms. In Chapter 3 a popular meta-heuristic, namely Differential Evolution (DE), is discussed more in detail. Chapter 4 briefly describes concepts of memetic computing, memetic structures, and introduces a simple memetic computing

structure, namely Three Stage Optimal Memetic Exploration (3SOME). Finally, Chapter 5 presents the main contributions of this work. Additional operators for 3SOME-algorithm are presented regarding scalability and complementary search moves and also modified algorithmic structure of 3SOME applying concept of meta-Lamarckian learning is included. Also two new memetic structures of DE are introduced along with cluster-based population initialization method to improve DE-based algorithms in general.

1.1 Derivative-based optimization

If objective function f has an analytical expression, is continuous, twice differentiable, and *unimodal* (has only one local optima), it's optima can be found analytically using Taylor series:

$$f(\overline{x}) = f(x) + \nabla f(x)(\overline{x} - x) + \frac{1}{2} Hf(x)(\overline{x} - x)(\overline{x} - x)^T + ... \tag{3}$$

where $\nabla f(x)$ is *gradient* and $Hf(x)$ is *Hessian matrix* of function f:

$$Hf(x) = \begin{bmatrix} \frac{\partial^2 f(x)}{\partial x_1 \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f(x)}{\partial x_n \partial x_n} \end{bmatrix} \tag{4}$$

At global optima x^* it must be that $\nabla f(x^*) = 0$ and it can be found by solving equation:

$$x^* = -\nabla f(x) H^{-1} f(x) + x \tag{5}$$

Calculation of Hessian matrix can be difficult and approximation of Taylor series causes new problems to appear. These problems can be worked with *Quasi-newton* methods or *conjugate gradient* based methods. However for arbitrary functions these techniques may fall short and their performance is not guaranteed.

1.2 Derivative-free optimization

If calculation of derivatives is hard or the objective function does not have an analytical expression (for example solution is gained with a computer program or by running a simulation) above mentioned techniques cannot be directly used. In these cases one can use so called *direct search* methods. Simplest direct searcher follows random directions and generates series of approximations for minimum until solution of certain accuracy is found. Direct search methods do not require information about objective function or its derivative although they use "discrete

derivative" when calculating difference of fitness values between points. One example of a such method is Hooke-Jeeves algorithm.

1.2.1 Hooke-Jeeves Algorithm

Hooke-Jeeves algorithm [26] has two basic steps: first it evaluates fitness of nearby points along coordinate-axes and second uses "pattern search" based on previous step to select desirable direction for next step. The algorithm starts from an initial point x_0 and explores along coordinate-axes with step size h, if there is no improvement towards selected axis algorithm tries the opposite direction with similar step size. After going through all search directions along the axes, new point x_1 is gained by adding the step sizes for relevant directions. If no improvement is gained the search is applied again with smaller step size. Otherwise, next step is taken to direction derived by x_0 and x_1 assuming that this direction is leading towards better fitness and then the algorithm is applied again on x_2.

1.2.2 Rosenbrock's Method

Rosenbrock's method, see [60] is similar to Hooke-Jeeves algorithm as it's a single point optimizer and uses somewhat similar search logic. Initially starting from point x_0 Rosenbrock's method searches along coordinates axes with step size h, if no improvement is found it looks to opposite direction with similar step size. If there is still no improvement the step size is halved. When improvement is found, initial coordinate axes are rotated towards approximated gradient and next iteration begins with step size reset following rotated coordinate axes. These steps are repeated until stop criterion is met. Downside of this approach is that for high-dimensional problems calculating rotated coordinate axes can be computationally expensive.

1.2.3 Nelder-Mead Method

While Rosenbrock and Hooke-Jeeves methods are single-point optimizers, Nelder-Mead, see [44], uses set of $n + 1$ solutions forming n dimensional polyhedron, or *simplex* in search space D. At each iteration of the algorithm, these points are sorted according to their fitness, so that x_0 has the best fitness and the x_n has the worst fitness. The algorithm then constructs candidate replacement point x_r for point x_n by reflection of point x_n respect with the center of other points $x_0, ..., x_{n-1}$. Depending on the fitness of replacement point x_r to point x_0 and x_{n-1}, an extension point may be created in an optimistic attempt to explore further in the same direction, or on the contrary a contraction point may be computed closer to point x_m. If above search fails (does not lead to better solution), the simplex is contracted around its best point in order to reduce exploration range in the next iteration of the algorithm.

1.2.4 Deterministic Local Search

Algorithms presented above are all deterministic by their nature, thus they follow a predictable path of search moves as there is no randomness involved. In several articles related to this work an single-point optimizer is used, which is very similar with the family of algorithms mentioned above. This component was first introduced in [28] as an one part of optimization process involving three different parts, which is discussed in detail in Chapter 4, and further modified in [5] where it was successfully used in conjunction with Rosenbrock's method to provide co-operative optimization framework. This particular local searcher is a deterministic single-point search very similar to Hooke-Jeeves. Starting from random point x_e, first it searchers along the coordinate axes with step size h, if there is no improvement along the axis it searches from opposite direction with step size $h/2$. After looking at each coordinate direction, best solution among these coordinates is selected as beginning point x_t for next iteration. Difference between Hooke-Jeeves is that there is no pattern search involved and Hooke-Jeeves is more greedy by nature as it moves to new point instantly when there is improvement, while Deterministic Local Search looks along all coordinates axes before deciding which direction to follow. Pseudocode for Deterministic Local Search is provided in algorithm 1.

Algorithm 1 Determinisitic local search

while termination condition is not met **do**
  $x_t = x_e$
  $x_s = x_e$
  for $i = 1 : n$ **do**
    $x_s[i] = x_e[i] - h$
    if $f(x_s) \leq f(x_t)$ **then**
      $x_t = x_s$
    else
      $x_s[i] = x_e[i] + \frac{h}{2}$
      if $f(x_s) \leq f(x_t)$ **then**
        $x_t = x_s$
      end if
    end if
  end for
  if $f(x_t) \leq f(x_e)$ **then**
    $x_e = x_t$
  else
    $h = \frac{h}{2}$
  end if
end while

2 META-HEURISTICS

Methods presented in previous chapter were classified into two categories: derivative-based and derivative-free techniques. These techniques both work well on uni-modal problems when problem has only one local optima. However often objective functions are more complex by their nature: *multi-modal, non-separable, ill-conditioned* etc. where multi-modality refers to functions having multiple local optimas and in optimization one is interested in finding the *global optima*. For single-point techniques multi-modal functions present a starting point problem: they are highly dependent on an initial selection of the starting point and often converge to a nearest local optima. Selecting proper step size can be problematic especially for ill-conditioned problems. Function is said to be separable if it can be minimized one variable at time. However objective functions often are non-separable and search techniques which highly rely on moving along the coordinate axes are insufficient in finding the global optima in such cases. To overcome these more complicated problems one has to turn to so called meta-heuristics.

Meta-heuristics can be considered as an efficient way to produce new solutions by trial and error to a complex problem within a reasonable amount of time. The complexity of the problem makes it impossible to check every possible solution to find best within acceptable time. In meta-heuristics there is no guarantee that the best solution can be found, and we may not even know whether given algorithm will work or why if it does work. The idea is to have an efficient and also practical algorithm that will work most of the time and is able to produce good quality solutions that are nearly optimal even there is no guarantee for the optimal one.

Many meta-heuristics typically have global convergence properties and thus can find acceptable solution within relatively limited amount of iterations or fitness evaluations. This property makes them suitable for solving global optimization problems. Meta-heuristics employ techniques that use pseudo intelligent ways for exploring new solutions. Such techniques involve both single-point optimizers: *Random Walk or Simulated Annealing* and *population-based meta-heuristics* where algorithms are often inspired by phenomena observed in nature, such as: *Genetic Algorithms, Evolutionary Programming, Evolution Strategies*, and *parti-*

cle swarm optimization. In the following sections we take a short survey on the methods mentioned above.

2.1 Single-solution Meta-heuristics

2.1.1 Random walk

The Random Walk algorithm, see e.g. [22], is one of the simplest ways to perform global optimization. Random Walk starts from a randomly generated point and tries to improve this solution by adding a random vector (point) to the existing solution. In the case where the new point leads to improvement in fitness value the point is stored for next iteration of the algorithm. It is possible that Random Walk can escape a local optima and find the global optima. Unfortunately this probability of the algorithm stepping from a basin of attraction to an another more promising basin is very low and thus is very time consuming process and not very useful approach. However, these kind of techniques inspire the more modern algorithms which use more sophisticated search logics than randomized addition vectors.

2.1.2 Simulated Annealing

Simulated Annealing, see [34], is inspired by how atoms move in molten metal while it's cooling, at the early stages when temperature is still high atoms rapidly move in the metal and "settle down" as the temperature gets lower. Algorithm follows the basic structure of Random Walk with the addition that Simulated Annealing can accept worse solutions with a probability that is controlled by temperature parameter. Temperature lowers each iteration of the algorithm and probability of accepting worse solution gets exponentially lower. Simulated Annealing can escape local optima with higher chance than standard Random Walk algorithm but it is not guaranteed. There exists modifications to improve the performance of the algorithm such as continuing iterations from previous better solutions than the current one, this is called restarting. Restarting can be done when for example if algorithm has run for fixed amount of iterations, no improvements for previous N iterations or by some other criteria.

2.2 Population-Based Meta-heuristics

In population-based meta-heuristics the search space is explored from several points simultaneously: forming a population of solutions. From population based techniques two categories can be distinguished:*Evolutionary Algorithms* (EA) and *Swarm Intelligence* (SI) algorithms. EAs generally mimic evolutionary processes observed in nature and main driving force of these algorithms comes from re-

combination of solutions and their genes (variables). SI algorithms emulate the behavior of flocks of birds or swarms of insects, for example how ants find the shortest path to food source. Instead of recombination of individuals these methods use individuals personal experience and behavior of other members/leader of the flock to produce new solutions. Third category of population-based meta-heuristics that should be mentioned are *Memetic Algorithms* (MAs). MAs utilize both population-based techniques but also include local searchers, which emulate life time learning of the individual, during some stage of optimization in a hybrid fashion. In following sections we take a short survey on these three categories and to related algorithms.

2.2.1 Evolutionary Algorithms

In the first category of population-based meta-heuristics we take a look into EAs. In EAs solutions form a population and by recombination try to adapt to their environment (in terms of fitness), this adaptation is done by four main mechanisms: 1) parent selection, 2) crossover, 3) mutation and 4) survivor selection. Generic pseudo-code for EAs is given in algorithm 2. We will shortly introduce some of more popular EAs used: Genetic Algorithms (GAs), Evolutionary Programming (EP), and Evolution Strategies (ES).

Algorithm 2 Evolutionary algorithm pseudo-code

```
Generate an initial population N
Evaluate fitness of each solution in N
while termination condition is not met do
  Select parents
  Create offspring by crossover
  Mutate the offspring
  Evaluate fitness of each offspring
  Select survivors for next generation
end while
```

2.2.1.1 Genetic Algorithms

Genetic algorithms, see [21], are inspired by reproduction process of living organisms, in GA parent population of solutions (*phenotypes*) are initialized then parents are recombined by mixing their attributes (*genotypes*) by *crossover* or *mutation* forming *offspring* solutions. In traditional GA phenotypes are represented by bit strings, this means that in a way or an another phenotypes need to be binary encoded, using for example Gray coding. In more modern approaches GA is proposed to use real numbers instead.

When selecting parents for recombination, the selection can be done different ways such as *proportional selection* where more fit parents have higher chance to get selected. Another way of selection is *k-tournament selection* where random sample of k parents is taken and the one with best fitness gets selected.

Recombination is done with two selected parents using crossover function. Two commonly used crossover schemes are single-point and two-point crossover where in case of single-point, both genotypes are cut from a random position and the other part is switched with the other candidate forming two new solutions. In two-point crossover string of bits is exchanged between parents. Other more complicated crossovers exist such as multi-point crossover, mask crossover or mapped crossover, see [17].

After recombination is done the two new offspring have probability to undergo a mutation. If the phenotype is presented as a bit string the mutation can be done by switching a random bit from 0 to 1 or vice versa, if phenotype is presented as real-valued vector, mutation can be done by adding (or subtracting) a random value to a random variable. Generally probability of mutation is set to be very low since too high occurrence of mutations can cause a loss of good solutions, that being said the importance of mutation is to provide search moves that would not be possible purely by crossover.

Once recombination phase is complete the final survival selection is done by generational selection where parents are replaced by their offspring.

2.2.1.2 Evolutionary Programming

Evolutionary Programming (EP), see [20], is inspired by how genomes alter in living beings by mutation rather than recombination. While still having a population of solutions as in GA, in EP each solution include additional real-numbered variables and also scaling parameters:

$$(x, v) = (x_1, ..., x_n, v_1, ..., v_n) \tag{6}$$

where x_i is i'th variable and v_i is scaling parameter for the i'th variable. At each iteration of the algorithm for each indivual (x, v) new candidate is generated using following formulas:

$$\begin{cases} v_i &= v_i(1 + \alpha \cdot \mathcal{N}(0,1)) \\ x_i &= x_i + v_i \cdot \mathcal{N}(0,1) \end{cases} \tag{7}$$

where α is a control parameter of the algorithm and $\mathcal{N}(0,1)$ is Gaussian random variable with mean at 0 and standard deviation of 1. After mutations the new candidate population is merged with parent population. Survivor selection is done individually by comparing solutions against a randomly selected set of individuals from merged population. Solutions with highest win percentages against their comparison set are then selected for the next generation.

2.2.1.3 Evolutionary Strategies

Evolution Strategies (ES), see [59], are similar to GA using both recombination and mutation as evolution tools. In ES solutions are coded in real-values so it's more suitable for continuous optimization than GA. As in Evolutionary Programming, ES uses similar way to represent the solutions:

$$(x, v) = (x_1, ..., x_n, v_1, ..., v_n) \tag{8}$$

where each x_i is associated with a self-adaptive scaling parameter v_i.

At recombination, population of μ parents are recombined to form a population of λ offspring. In general offspring population size $\lambda \geq \mu$. Recombination can be *discrete* where components from two parents are combined or *intermediate* where offspring are for example an average of the parents.

As in GA, after recombination follows mutation that uses similar logic which is used in EP:

$$x_i' = x_i + v_i' \cdot \mathcal{N}(0, 1) \tag{9}$$

where x_i is solution being mutated and v_i' is scaling parameter. Several ways to generate scaling parameter v_i' has been proposed in literature for example, see [24].

Finally when selecting survivors, there is two common approaches: In (μ, λ)-ES also know as "comma" variant, the best μ offspring are selected to become next parent population and in $(\mu + \lambda)$-ES also knows as "plus" variant, the parent and offspring populations are merged and then best μ solutions are selected as parents for the next generation.

2.2.2 Swarm Intelligence

While EAs simulate the evolution, Swarm Intelligence (SI) algorithms simulate behavior of swarms of insects/animals where solutions represent members of such swarm. Instead of recombination new solutions are generated by following earlier behavior of the individual, current "swarm leader" and/or neighboring solutions. Replacement is usually done by "one-to-one" spawning logic, where individual is compared against it's old position and only replaced in case of improvement. We present here two well-known SI algorithms: *Particle Swarm Optimization* which is inspired by studying how flock of birds behave and *Ant Colony Optimization* simulates how ants find shortest path to food source.

2.2.2.1 Particle Swarm Optimization

As mentioned above, Particle Swarm Optimization (PSO), see [32], mimics how flock of birds or fishes move. Each solution represents one bird in the flock and when initialized each solution x_i is given not only location but also velocity v_i. Over iterations locations of solutions are updated by adding their velocity:

$$x_i' = x_i + v_i \tag{10}$$

In addition to location and velocity, each solution keeps track of their past best location x_i^{best} called *local best*. From the collection of local bests the one with best fitness x^{best} is called *global best*. By using these memories of individuals and the flock "leader", velocities are adjusted between iterations by following formula:

$$v_i' = v_i + \alpha \cdot \mathcal{U}(0,1)(x_i^{best} - x_i) + \beta \cdot \mathcal{U}(0,1)(x^{best} - x_i) \quad (11)$$

where α, β are scaling parameters and $\mathcal{U}(0,1)$ is uniformly distributed random number between 0 and 1. How particle swarm optimization is guided by using local and global bests is conceptually similar to crossover operation used in many evolutionary algorithms.

2.2.2.2 Ant Colony Optimization

Ant Colony Optimization, see [12], is inspired by ants searching for shortest route to food source. While searching for food ants leave behind pheromones to help other ants to find the source. Pheromones evaporate over time and thus short paths get selected by more and more ants eventually leading to all ants using same path. Originally Ant Colony Optimization was used to find optimal path on graph and while it's applicable to other problems as well, problem needs to be expressed as a search for a best path along given graph.

Solutions in Ant Colony Optimization present single ant traveling the graph. While these ants travel randomly at start the pheromones they leave behind attract other ants to the same path. As mentioned earlier the pheromone levels on paths/nodes of graph evaporate over time, short paths have higher probability to get selected and more ants traveling same paths further enhance pheromone levels of selected path. Eventually all ants will travel optimal path found. Evaporation of pheromones is essential for optimization to not get stuck on local optima, without this feature the paths selected by very first ants would be favored on following iterations leading to this problem.

2.2.3 Memetic Algorithms

While Genetic Algorithms try to emulate biological evolution, Memetic Algorithms (MAs) can be seen trying to emulate a cultural evolution. Word "meme" is introduced as *unit of imitation in cultural transmission* in "The Selfish Gene" by R. Dawkins. Quoting Dawkings:

> "Examples of memes are tunes, ideas, catch-phrases, clothes fashions, ways of makings pots or to building arches. Just as genes propagate themselves in the gene pool by leaping from body to body via sperm or eggs, so memes propagate themselves in the meme pool by leaping from brain to brain via a process which, in the broad sense, can be called imitation."

In practice MAs are mix of population-based global search (by evolution) and individual based local search (by learning). The reasoning behind these type of techniques is that different algorithmic components perform better together than components alone would, when implemented in a co-operative framework. Inclusion of local search can be classified into two categories. In the first category local search is applied within recombination process instead of mutation and/or crossover and the in the second category local search is applied after generation of offspring trying to improve upon their initial fitness value. In the latter case the

process is called *life-time learning* as it simulates the newborn going to school and picking other parts of information along what the individual inherited from the parents. There are generally two different philosophies of how life-time learning can be implemented: *Lamarckian* and *Baldwinian*. In Baldwinianism the improvement in fitness gained by the life-time learning only affects individuals fitness value but not its genotype. Thus, genotype is not transferred into the offspring. While in Lamarckianism the improvement gained by life-time learning causes mutation in genotype and thus can be transferred into the offspring. In human genetics Lamarckianism does not hold, but in MAs such approach can be very successful.

When applying local search within evolutionary framework the designer has to make decisions regarding how often LS is applied, which individuals LS is applied on or what is the depth of the local search (depth e.g how many fitness evaluations LS should do)? All of these decisions are crucial to the performance of the algorithm as the local searchers are usually very expensive in terms of fitness evaluations and unnecessary activations can majorly hinder the performance of the whole algorithm, for example when local search is applied too intensively causing premature convergence to a local optima. In [27] "partial Lamarckianism" is used as a strategy to control when local search is applied by giving it a certain probability. In [43], the volume of the local search applied on individual is dependent on their fitness. In [42], balance between exploration and exploitation is highlighted.

2.2.4 Performance of Meta-heuristics

Key components in any meta-heuristic is to achieve good balance between *exploitation* and *exploration*. In other words algorithm needs to be able to generate new solutions improving on previous solutions and also cover most important areas where global optimum may lie. In MAs exploration is generally done by population based meta-heuristic while applying local searcher to exploit most promising solutions in order to improve on best solution found so far. As possible applications are almost limitless it is more or less impossible task to generate algorithm which would have perfect balance on every possible problem. This is also known as No Free Lunch Theorem (NFL):

> "what an algorithm gains in performance on one class of problems is necessarily offset by its performance on the remaining problems."

as written in [73]. In practice when dealing with a specific problem the optimization algorithm which is to be used should be modified taking into account problem specific knowledge to guarantee best performance. Even though tailored algorithms are indeed better when dealing with specific problem it is still important to develop good general (and specific) purpose algorithms as they often work as a very promising base which tuning can be applied on. In figure 1 we have illustration of general purpose algorithms versus tailored algorithms compared over different types of problems.

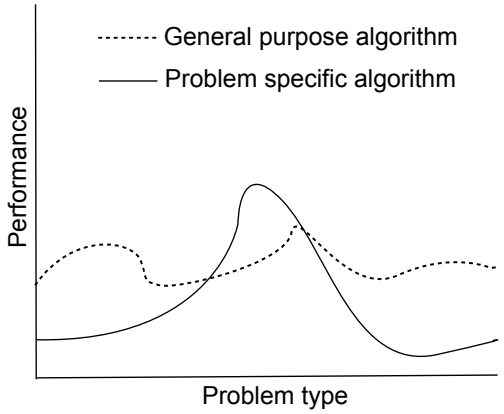


FIGURE 1 Difference between general purpose algorithms and tailored algorithms

3 DIFFERENTIAL EVOLUTION

Differential Evolution (DE), see [66], deserves separate mention among population-based meta-heuristics. DE contains some characteristics from both Evolutionary Algorithms and Swarm Intelligence. Initial population is randomly spread around search space as in other EAs, but recombination schemes have different approach. While sharing same four mechanisms of producing offspring with EAs: parent selection, crossover, mutation, and survivor selection, in DE mutation becomes before crossover and while in EAs crossover is between two parents, in DE crossover is between one parent and the *provisional offspring* generated by mutation. Also mutations generated in DE does not necessarily contain the parent that's being mutated. Selection procedure is also different, in GA the population is replaced by their offspring and in ES most fit solutions are being selected. In GA the parent selection is based on their fitness which allows the generational offspring generation. In DE selection of parent is random and not based on their fitness and thus generational approach does not work. Instead DE uses one-to-one spawn logic where offspring is directly compared against it's parent and the better solution is selected as a member of new population. These differences in parent selection, mutation, crossover and survivor selection do not anymore follow the classic evolution and in this light DE is not directly inspired by nature. Generic pseudo-code for DE is given in algorithm 3.

Being very versatile, easy to implement, and having high performance, DE is one of the most popularly used optimization algorithms among scientists and engineers. This has lead to many proposals of improvements on the standard scheme of DE. In this chapter we will review the classic version of DE and also take a look into some of the modified structures related to parameter settings, self-adaptation, population etc., and to reasons which have lead to these new implementations.

Algorithm 3 Differential Evolution pseudo-code

```
Generate an initial population of Np individuals
Evaluate fitness of each solution in population Np
while termination condition is not met do
  for each x_i in Np do
    Create provisonal offspring x_{i,off} by mutation
    Create offspring x_{off} by crossover
    Evaluate fitness of x_{off}
    Select survivor between x_i and x_{off}
  end for
end while
```

3.1 Population Initialization

Before generating initial population, for each variable x_j upper and lower bounds $b_{j,U}$ and $b_{j,L}$ needs to be specified. Once these bounds have been set, a random Np solutions are generated within these bounds. For example, the initial value of the j^{th} variable of the i^{th} solution is:

$$x_{i,j} = \mathcal{U}(0,1) \cdot (b_{j,U} - b_{j,L}) + b_{j,L}. \tag{12}$$

Even in the case where variables are discrete values they should be initialized with real-value since DE internally treats all variables as floating-point values. Next, each parent in population undergoes a mutation, a crossover and a selection in that order. More detailed pseudo-code for DE is presented in algorithm 4.

3.2 Mutation

After initialization of a population, DE mutates and recombines the parent population to produce an offspring population of another Np trial solutions. To perform mutation DE randomly picks three solutions from the population: x_t, x_r, and x_s, such that $x_t \neq x_r \neq x_s$. In *differential mutation* solution x_i is mutated to create *provisional offspring* or *mutant* $x_{i,off}$:

$$x_{i,off} = x_t + F \cdot (x_r - x_s) \tag{13}$$

where $F \in]0,1[$ is called *scale factor* that controls the distance how far provisional offspring are generated. Scale factor F is one of the parameters in DE and while it has no upper limit, values greater than 1 are rarely considered to be effective. As mentioned earlier the mutation of solution x_i does not contain the solution itself unless it's randomly selected as one of the solutions x_t, x_r or x_s. In standard DE the selection of x_t, x_r, x_s is done by random, but in some mutation schemes the *base vector* x_t can, for example, be selected as x_{best}, where x_{best} refers to best solu-

Algorithm 4 Differential Evolution pseudo-code

Generate an initial population of Np individuals
while termination condition is not met **do**
  while $i = 1 : Np$ **do**
    Calculate $f(x_i)$
  end while
  for $i = 1 : Np$ **do**
    Create provisional offspring $x_{i,off}$ by mutation
    Select x_t, x_r and x_s randomly from population Np such that $t \neq r \neq s$.
    Calculate $x_{i,off} = x_t + F \cdot (x_r - x_s)$
    Create offspring x_{off} by crossover
    $x_{off} = x_i$
    for $j = 1 : n$ **do**
      Generate $p = \mathcal{U}(0, 1)$
      if $p < Cr$ **then**
        $x_{off}[j] = x_{i,off}[j]$
      end if
    end for
    Generate j_{rand} randomly such that $j \in 1, ..., n$
    $x_{off}[j_{rand}] = xi, off[j_{rand}]$
    Evaluate fitness of x_{off}
    Select survivor between x_i and x_{off}
    Calculate $f(x_{off})$
    if $f(x_{off} < x_i$ **then**
      Save index for replacement $x_i = x_{off}$
    end if
  end for
  perform replacements
end while

tion in population. The mutation scheme in eq. 13 is also known as DE/rand/1. Example of generation of provisional offspring by DE/rand/1 can be seen in picture 2.

Also other mutation schemes have been proposed in the literature, see [55]:

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

where x_{best} is the best solution in the population and x_u and x_v are additional pseudo-random individuals selected from the population. One more mutation operator should be mentioned; this operator is known-as *rotation invariant mutation*, see [52]:

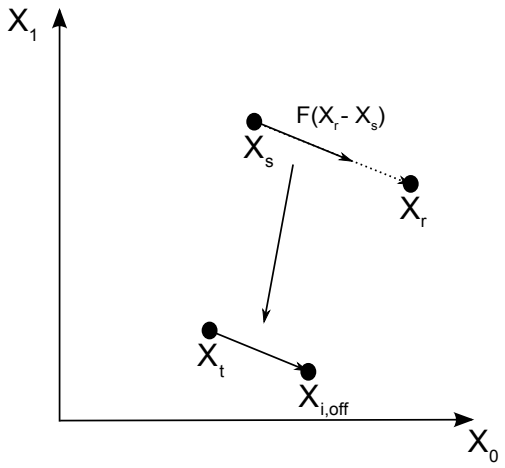


FIGURE 2 DE/rand/1 mutation scheme

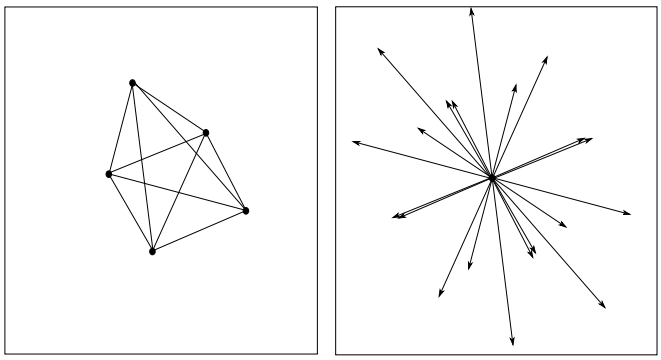


FIGURE 3 Difference vectors and their distribution

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

where combination coefficient K is uniformly distributed random number from $[0, 1]$ and $F' = K \cdot F$. When using this mutation operator solution does not undergo crossover, which is described in the next section.

In figure 3 we have population of five points and all possible difference vectors that can be generated by mutation. Thing to be noted is that possible combinations of directions is defined and thus also limited by current population, we will discuss this matter in the following sections. Mutation schemes using more than one difference vectors (e.g. DE/rand/2) increase this pool of possible provisional offspring generated and thus increasing the exploration capability of the mutation operator.

It must be noted that different mutation schemes perform better/worse when applied on different problems and none of them has been proven to be the best [9].

3.3 Crossover

After provisional offspring $x_{i,off}$ is generated by mutation it is recombined with the parent x_i to form final offspring x_{off}. There is generally two main ways of doing crossover in DE: *binomial* and *exponential crossover*.

3.3.1 Uniform (Binomial) Crossover

In binomial crossover each variable in x_i is exchanged with provisional offspring with probability of Cr, this is called *crossover rate*. More precisely, binomial crossover is done by the following formula:

$$x_{off}[j] = \begin{cases} x_{i,off}[j] & \text{if} \quad (\mathcal{U}(0,1) \leq Cr \quad \text{or} \quad j = j_{rand}) \\ x_i[j] & \text{otherwise} \end{cases} \tag{14}$$

where j is index of gene being selected and $\mathcal{U}(0,1)$ is uniformly distributed random number. Parameter Cr can be interpreted as expected percentage of genes altered by provisional offspring. One gene j_{rand} should be randomly selected to be swapped to ensure that at least one of the genes is altered. Binomial crossover is often referred as "bin" in context to present full DE scheme being used, such as DE/rand/1/bin.

3.3.2 Exponential Crossover

Second commonly used crossover is called exponential crossover (referred as "exp"), see [8] which is two-point crossover similar to crossover used for example in GA. Main idea is that every gene between these two points should be selected from the provisional offspring $x_{i,off}$ while others are preserved from the parent x_i. First index of crossover j_{rand} is selected by random and this and following gene in index $j_{rand} + 1$ is selected with probability Cr, this is repeated until $\mathcal{U}(0,1) > Cr$, which indicates the last point of crossover. Every gene selected this way is taken from provisional offspring, while remaining are kept from the parent. More precisely, formula of exponential crossover is:

$$x_{off}[j] = \begin{cases} x_{i,off}[j] & \text{if} \quad j \in J) \\ x_i[j] & \text{otherwise} \end{cases} \tag{15}$$

where j_{rand} is randomly selected starting index and $J = \{j \in \text{mod } Np \mid j = j_{rand} \text{ or } \quad j-1 \in J \text{ and } \mathcal{U} \leq Cr\}$ is an index set which is defined by starting index and number of occurrences of $\mathcal{U} \leq Cr$ and Np is population size. For sake of clarity pseudo-code for exponential crossover is given in algorithm 5.

One must note that crossover rate Cr has different interpretation in exponential than in binomial crossover, as it does not anymore correspond to expected percentage of genes altered by crossover and it's not anymore independent of problem dimension. This is especially important when problem dimension goes very high, to hundreds or thousand different variables, the meaning of

Algorithm 5 Exponential crossover

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

Cr changes crucially and that needs to be taken into account when setting parameter value of crossover rate Cr. To overcome this situation *inheritance factor* can be used:

$$\alpha_e \approx \frac{n_e}{n} \tag{16}$$

where n_e is the number of mutant genes we expect to copy from mutant $x_{i,off}$ into offspring x_{off} in addition to the gene deterministically copied. In order to achieve that on average n_e are copied into the offspring we need to impose that

$$Cr^{n\alpha_e} = 0.5. \tag{17}$$

It can easily be seen that, for a chosen α_e, the crossover rate can be set on the basis of the dimensionality in the following way, see [47] for details:

$$Cr = \frac{1}{\sqrt[n\alpha_e]{2}}. \tag{18}$$

Another point worth mentioning considering both binomial and exponential crossover is that like mutation these crossover schemes also have limited amount of search moves. For example in 2-dimensions there is only four possible combinations of crossovers where one is "degenerate" as there is no difference between offspring x_{off} and parent x_i:

- Neither of genes of mutant $x_{i,off}$ are transferred to parent x_i resulting to degenerate offspring which is identical with parent.
- One of the genes of mutant $x_{i,off}$ is transferred to parent x_i providing two possible offspring (x_0 and x_1 in figure 4).
- Both genes are transferred and offspring x_{off} equals to mutant $x_{i,off}$.

In figure 4 one can see these four different offspring on 2-dimensional problem. Possible drawbacks in DE regarding limited search moves for both mutation and crossover are discussed in section 3.5.

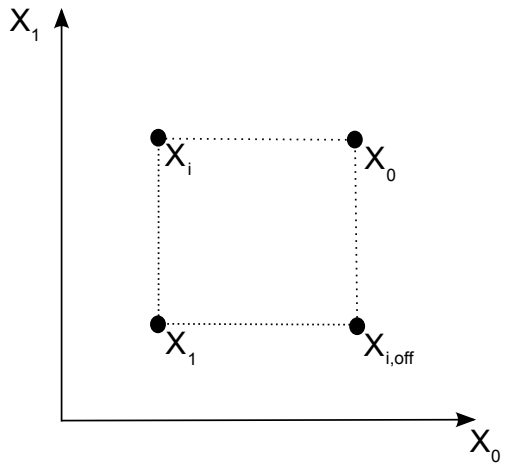


FIGURE 4 Four possible crossover outcomes between parent x_i and mutant $x_{i,off}$. Point x_i being degenerate as no genes are swapped.

Other crossover strategies such as *one-point crossover*, see [31] and *N-point* crossover, see [18], can also be considered. Whichever crossover strategy is used, lower Cr corresponds to lower mutation rate of solutions while increasing Cr increases mutation rate, as value close to 0 produces solution nearly similar to parent and with values closer to 1 most genes come from the offspring.

3.4 Selection

Final step in one iteration of DE is selection step where it is decided which solutions among offspring and current parent population form the next generation.

As discussed earlier, in GA the parent selection is based on their fitness which allows age-based offspring generation. However parents are selected randomly in DE and thus such approach does not work. Objective function based selection is present e.g. in $(\mu + \lambda)$-ES on page 18, where parent population and trial population are combined, sorted and then best μ solutions are selected as parents for next generation. It's clear that age of solution does not play role in this selection procedure, fitness being only one that does. This fitness based selection ensures that best solution is always retained for next generation, such property of an algorithm is also known as *elitism*.

Tournament selection is used for example in Evolutionary Programming, where tournaments are held between solutions. Selection is then based on solutions win percentages in the tournaments. One problem present in the tournament selection is that population may suffer from *premature convergence* as dominated solutions are eliminated from the population in few generations and genetic diversity of the subsequent population of children suffers. In DE selection procedure is *one-to-one tournament* selection which is also used in Particle Swarm

Optimization algorithms. One-to-one tournament selection is based both on fitness and age as generated offspring solutions are directly compared to their parent in the current index x_i and the solution with best fitness gets selected as part of new population. Traditionally selection is done after all parents have generated offspring solutions but it's also possible to do selection as offspring are generated thus replacing the parent while iteration is still in progress. The latter selection procedure is called *continuous selection* and the former is called *discrete selection*. Clear difference is that in continuous selection newly selected solutions can be selected as x_r, x_s and x_t, continuous selection mechanism has been studied e.g. in [68].

Generally selection procedure in DE is indeed elitist as best individual is always preserved and offspring population never contains worse solutions than parent population. Compared to other tournament selection mechanisms, one-to-one tournament selection is able to maintain population diversity better as less fit solutions do not disappear as rabidly from the population as they do not have to compete against the best solutions in the whole population but only their own offspring.

3.5 Implicit Self-Adaptation and Stagnation

As discussed earlier in section 2.2.4, balance of explorative and exploitative search moves is essential for any successful meta-heuristic. Mutation and crossover form an implicit self-adaptation mechanism in DE: At the very beginning the population is initialized randomly in decision space and thus when selecting mutation vectors x_t, x_r and x_s and combining them, for example using $DE/rand/1/bin$-strategy, leads to step sizes that result to offspring which are generally far away from their parent. As a result of convergence more solutions will be gathered in the more interesting areas of search space, to so called basins of attraction, and offspring generated during later generations are generally more near the parent solutions than during first iterations. This is due to fact that mutation vectors x_t, x_r and x_s can be selected from the same basin of attraction and step size in the mutation is reduced. In this sense DE shifts from explorative to more exploitative nature as optimization process progresses.

At the first glance this can be seen as a very nice feature but it can also lead to an undesired *stagnation* condition, as pointed out in [38], where algorithm is not capable to produce search moves that lead to improved solutions. If all solutions happen to converge to same basin of attraction it is impossible for DE to generate solutions outside this basin because of the limited search moves, as mentioned at section 3.3, the original DE implementation has. This stagnation condition can also happen even when DE has not converged towards an optima (even a local optima).

3.6 Ways Around Stagnation in DE

During the last years much effort has been put in overcoming stagnation in DE. As a result multiple new variants of DE has also been proposed in literature, in [49] are listed number of DE-variants for single-objective optimization problems and comparison among these variants. In aforementioned study the variants are categorized in to following two groups:

- Modified structures of DE. This class includes those algorithms which make considerable modifications in the search logic of basic DE as a form of mutation, crossover or selection mechanism.
- DE integrating extra components. In this class belong algorithms which integrate one or more additional algorithmic components into DE, such as local searcher, alternate recombination mechanism, etc.

3.6.1 Modified Structures of Differential Evolution

Selection of control parameters crossover rate Cr, scale factor F and population size Np having huge impact on how DE performs is widely agreed fact. Discussion on optimal parameter settings for these control parameters can be found e.g. in [53], where it is indicated that a reasonable value Np could be chosen from $[5D, 10D]$, where D is problem dimension, and initial choice for scale factor was $F = 0.5$ and effective range of Cr is $[0, 0.2]$ when the function is separable and $[0.9, 1]$ for non-separable functions. There are some contradicting report considering selection of these parameters as the selection is always problem related and can be considered as an optimization task itself. This difficulty of adjusting parameter settings can be a very confusing task for people who try to solve practical problems with DE and thus researchers have considered approaches with dynamic control parameters and self-adaptation mechanisms to find optimal settings for these values also to tackle the stagnation problem.

If stagnation is caused by having too large step-sizes, which depends on difference of the two vectors in the population and the scale factor F, where scale factor is something that can be dynamically modified. Instead of using constant scale factor, the scale factor can be selected separately for each individual in population and is called *dithering*. Sampling different scale factor for each component of each individual is called *jitter*. While dithering changes the norm of the difference vector, jitter changes the orientation as well. In [75] the scale factor F is replaced by normally distributed random variable F_i or F_j, where F_i is scale factor for each individual x_i and F_j is scale factor of each component $x_{i,j}$ of solution x_i. Differential Evolution with Random Scale Factor presented in [15], mainly designed for the optimization of noisy functions, is another example of dithering where F is selected with uniform distribution $F_i = 0.5 \cdot (1 + \mathcal{U}(0, 1))$. The algorithm also presents a threshold margin of $\tau = k \cdot \sigma_n^2$, where σ_n^2 is variance of noise, for offspring to get selected to be selected over it's parent.

3.6.1.1 Self-Adaptive Control Parameters in Differential Evolution

In Self-Adaptive Control Parameters in Differential Evolution, presented in [2], namely jDE, self-adaptive strategy has been proposed to avoid manual parameter setting. This approach employs the standard DE/rand/1/bin strategy, presented earlier in section 13, with some modifications. When initial population is generated another extra values in range $[0, 1]$ are generated per each individual in the population. These values represent F_i and Cr_i for individual x_i thus each individual will have their own control parameters. When generating the offspring for x_i new parameter values are generated by following formulas:

$$F_i = \begin{cases} F_l + F_u \cdot rand_1, & if \quad rand_2 < \tau_1 \\ F_i, & otherwise \end{cases} \tag{19}$$

and

$$Cr_i = \begin{cases} rand_3, & if \quad rand_4 < \tau_2 \\ Cr_i, & otherwise \end{cases} \tag{20}$$

where $rand_j$, $j \in 1, 2, 3, 4$ are uniform pseudo random numbers between 0 and 1, τ_1 and τ_2 are the probabilities that parameters are updated and F_l and F_u are lower and upper boundaries for scale factor, respectively. These values are then used in the generation of offspring. Approach has also been extended to large-scale problems in [77] and multi-objective problems in [76].

3.6.1.2 Adaptive Differential Evolution With Optional External Archive

In modified DE structure called JADE, presented in [78], the values of the control parameters are updated when offspring lead to improvements by outperforming their parents. The algorithm uses specific mutation scheme called DE/current-to-*p*best/1/bin:

$$x_{i,off} = x_i + F_i \cdot (x_{best}^p - x_i) + F_i \cdot (x_r - x_s), \tag{21}$$

where x_{best}^p is randomly chosen among the $100 \cdot p\%$ best individuals in the current population with $p \in]0, 1]$, F_i is the scale factor for individual x_i and x_r, x_s are two random individuals in the population. The provisional offspring is then recombined with parent individual x_i by uniform (binomial) crossover, where the crossover rate Cr_i is the crossover rate assigned to x_i. The values for F_i and Cr_i are randomly generated at the beginning of each iteration based on the parameters μ_F and $\mu_{Cr}]$. The values for crossover rates Cr_i are generated by normally distributed random variable with mean μ_{Cr} and for scale factors F_i: one third are generated by normally distributed random variable centered around μ_F and rest by uniformly distributed random variable. After selection the parameters of more promising solution are saved into two sets S_F and S_Cr and at the end of generation the values for μ_F and μ_{Cr} are updated by calculating weighted average between value of the parameter and for μ_{Cr} the mean value of S_{Cr} and for μ_F the Lehmer mean of S_F, where Lehmer mean is defined by:

$$L(S_F) = \frac{\sum_{F \in S_F} F^2}{\sum_{F \in S_F} F}. \tag{22}$$

3.6.1.3 Self-Adaptive Differential Evolution

Differential Evolution employing multiple mutation schemes along with individual based control parameters has been originally proposed in [55]. This algorithm is called Self-Adaptive Differential Evolution (SaDE) and improved version on the original scheme was proposed in [54]. In SaDE each individual has a probability to select one of four different mutation schemes: DE/rand/1, DE/rand-to-best/2, DE/rand/2 and DE/current-to-rand/1, while using binomial crossover strategy for everything except for DE/current-to-rand/1 (as it uses no crossover). For initial learning period of LP generations probabilities for each mutation p_i^k is set to 0.25 and for each individual x_i success ratios of each mutation strategy $n_s^{i,k}$ are recorded, also failures to generate outperforming offspring $n_f^{i,k}$ are recorded. At the end of learning period and at the end of each generation G the probabilities p_i^k are updated by following formula:

$$p_i^k = \frac{S_i^k}{\sum_1^4 S_i^k} \tag{23}$$

where

$$S_i^k = \frac{\sum_{G-LP}^{G-1} n_s^{i,k}}{\sum_{G-LP}^{G-1} ni, k_s + \sum_{G-LP}^{G-1} n_f^{n,k}} + \epsilon \tag{24}$$

and ϵ equals to 0.01 to ensure numerical stability of the algorithm in the case where $S_i^k = 0$ for all mutation schemes. The mutation scheme to create each provisional offspring is selected by Stochastic Universal Sampling [1] based on the updated probabilities.

Along with mutation probabilities each individual is also assigned with crossover rate Cr_i^k for each mutation scheme and one scale factor F_i. The value for F_i is randomly sampled from normal distribution $N(\mu, \sigma)$, where $\mu = 0.5$ and $\sigma = 0.3$. The crossover values Cr_i^k are initialized as 0.5 for each strategy k and are updated on every generation during learning period by a random normally distributed value $N(\mu, \sigma)$, where $\mu = Cr_i^k$ and $\sigma = 0.1$. Crossover rate values leading to improvements in offspring are saved for each individual and mutation scheme and at the end of learning period Cr_i^k is replaced by the median of saved values Cr_i^k. After the learning process the crossover continue to be updated as before for the remainder of the optimization process.

3.6.2 Differential Evolution Integrating Extra Components

Another category of modifications to standard DE implementations is generally adding additional components that aim to complement the limitations in stan-

dard search moves. One way of introducing new component to DE is bring in for example Local Search which can be applied on some of the individuals in the population (commonly to one with best performance). Another approach could be to introduce new mutation operation as is done in Differential Evolution with Trigonometric mutation, see [19]. This mutation scheme includes the fitness of the three randomly selected vectors involved in the mutation. The provisional offspring is computed by:

$$x_{i,off} = \frac{x_r + x_s + x_t}{3} + (p_s - p_r)(x_r - x_s) + (p_t - p_s)(x_s - x_t) + (p_r - p_t)(x_t - x_r), \tag{25}$$

where $k = r, s, t$ and

$$p_k = \frac{|f(x_k)|}{|f(x_r)| + |f(x_s)| + |f(x_t)|}. \tag{26}$$

Thus the provisional offspring generated is the centroid of the triangle formed by x_r, x_s and x_t, where the weight of each solution is a function of it's fitness and average fitness of other two solutions. Solutions generated are improved in more greedy manner than in standard DE thus accelerating convergence speed which can be beneficial especially in the case where evaluation of fitness value is a time consuming process. The algorithm uses new control parameter M_t as a probability to activate trigonometric mutation scheme which can be tuned to control the greediness of the algorithm.

3.6.2.1 Differential Evolution with Scale Factor Local Search

Differential Evolution with Scale Factor Local Search (DESFLS) has been introduced in [48] and further extended in [50] for self-adaptive DE schemes. Main feature of this modification is to consider selection of proper scale factor value F as an one dimensional optimization problem. With certain probability a local searcher is applied on scale factor when generating provisional offspring. More specifically after individuals x_t, x_s and x_r have been selected for generation of provisional offspring, the algorithm tries to generate best possible offspring by solving following minimization problem:

$$minf(F) \quad \text{, where} \quad F \in [-1.2, 1.2]. \tag{27}$$

Minus values in the problem above are for inversion of the search direction. In solving of this problem two different local search algorithms are studied and compared in [71] and []: Golden Section Search, see [33], and Hill-Climb Local Search, see [62]. Experimental results in showed that cooperative framework of both above mentioned local searchers lead to best performance.

3.6.2.2 Differential Evolution with Global and Local Neighborhoods

The Differential Evolution with Global and Local Neighborhoods (DEGL), see [10] and [14] draws inspiration from PSO algorithms by defining index based so-

lution neighborhoods. A set of population individuals $\{x_{i-k}, ..., x_i, ..., x_{i+k}\}$ form the local neighborhood with a radius k for a solution x_i. A new mutation operator is then defined as:

$$x_{i,off} = wG_i + (1 - w)L_i, \tag{28}$$

where $w \in [0, 1]$ is a weight factor balancing the weight of local and global contribution L_i and G_i correspondingly. Local contribution L_i is defined by:

$$L_i = x_i + \alpha(x_{l-best} - x_i) + \beta(x_p - x_q) \tag{29}$$

and global contribution G_i by:

$$G_i = x_i + \alpha(x_{g-best} - x_i) + \beta(x_r - x_s), \tag{30}$$

where x_{l-best} is a solution with best fitness in the neighborhood of x_i, x_p and x_q are randomly selected individuals in the same neighborhood, x_{g-best} is the best individual in the whole population and x_r and x_s are two randomly selected individuals from the population. In [10] it is suggested to set $\alpha = \beta$ equal to a constant value and weigh factor w to vary in a following way:

$$w = w_{min} + (w_{max} - w_{min})\frac{g}{g_{max}}, \tag{31}$$

where w_{min} and w_{max} are the lower and upper bounds of weight factor, respectively. The indexes g and g_{max} denote the current generation index and the maximum amount of generations, respectively. In [14] four alternative weight factor schemes have been compared which of a self-adaptive scheme proved to be the most efficient one.

3.6.2.3 Opposition Based Differential Evolution

In the Opposition Based Differential Evolution (OBDE), proposed in [57] and [58], supplementary search moves are introduced by testing for opposite points of solutions in the search space. Opposite point $\widetilde{x_i}$ for solution $x_i = (x_i[1], ..., x_i[n])$ belonging to a set $D = [a_1, b_1] \times ... \times [a_n, b_n]$ is defined as:

$$\widetilde{x_i} = (a_1 + b_1 - x_i[1], ..., a_n + b_n - x_i[n]). \tag{32}$$

Checking for opposite points is done at two steps: after initialization of the population and after survivor selection. After initialization of first solutions is done their opposite solutions are calculated forming two populations of size Np. Then these two population are combined and best Np solutions are selected to first generation. At each subsequent generation opposite solutions are calculated for the population with the probability j_r (jump rate). In this case each opposite point is calculated with:

$$\widetilde{x_i} = \big(min_i x_i[1] + max_i x_i[1] - x_i[1], ..., min_i x_i[n] + max_i x_i[n] - x_i[n]\big), \tag{33}$$

where $min_i x_i[j]$ and $max_i x_i[j]$ are respectively minimum and maximum values over the coordinate j of all solutions in the present generation. In other words, minimum and maximum values are taken from the bounding hypercube generated around current solutions. The populations for offspring and their opposite solutions are then merged and again Np best performing solutions are selected for next generation.

3.6.2.4 Differential Evolution with Population Size Reduction

Another approach to provide supplementary search moves is to vary population size during the search as modifying the amount of available solutions directly translates into change in the possible search moves generated in mutation. In Differential Evolution with Population Size Reduction (DEPSR), see[3] the population size is progressively reduced during the optimization process. The main idea behind this approach is to focus the search progressively to avoid possible stagnation condition, especially in high-dimensional problems. Population reduction strategy used requires that initial population size N_p^1, total budget T_b (i.e. total number of fitness evaluations) and number of stages N_s (i.e number of population sizes employed) are predefined.

The population reduction is carried at the end of each stage by simply halving the current population. More specifically, the population is divided into two equal size sub-populations and then one-to-one selection occurs between these sub-populations. The individual with better fitness is selected into population used in the next stage.

4 MEMETIC COMPUTING STRUCTURES

By original definition MAs are combination of a population-based meta-heuristic and a local searcher as briefly discussed in section 2.2.3. The MAs form a cornerstone for Memetic Computing (MC), which is more broad subject defined as following in [46]:

> "Memetic Computing is a broad subject which studies complex and dynamic computing structures composed of interacting modules(memes) whose evolution dynamics is inspired by the diffusion of ideas. Memes are simple strategies whose harmonic coordination allows the solution of various problems."

4.1 Coordination of the Algorithmic Components

While MAs are related to hybrid style of algorithms, MC structures are related to not only to the algorithmic components itself, but to the structures how these components coordinate. In this light, MC can be seen as a subject that studies structures composed of modules (memes) which interact and adapt to problem in order to solve it. This adaptation is often done online (during optimization run). In recent survey [45] coordination between memes is categorized to be performed one of the following ways:

- Adaptive Hyper-heuristic, where the coordination of the memes is performed by means of heuristic rules
- Meta-Lamarckian learning, where the success of the memes biases their activation probability, thus performing an on-line algorithmic design which can flexibly adapt to various optimization problems
- Self-Adaptive and Co-Evolutionary, where the memes, either directly encoded within the candidate solutions or evolving in parallel to them, take part in the evolution and undergo recombination and selection in order to select the most promising operators
- Fitness Diversity-Adaptive, where a measure of the diversity in fitness is used to select and activate the most appropriate memes.

In the first category are those implementations which include a prefixed coordination scheme. These schemes can be either randomized or deterministic. Randomized schemes include selection probability to active memes one by one. In deterministic implementations one can use prefixed amount of fitness evaluations for each meme, see [35], or choice function based on the success of the meme, e.g. in the case of success meme is subsequently reapplied. Example of a deterministic scheme used to coordinate three memes is described more in detail in the following section 38.

In meta-Lamarckian learning, see [51], active component is selected by means of it's success. The success ratios are used to adapt selection probabilities online, thus making the optimization algorithm adapt to a problem and use most successful memes in solving it.

In third category are self-adaptive and co-evolutionary schemes. In such schemes solutions consist of genetic and memetic material. Where genetic material represent the design variables of solution and memetic material are the evolution parameters, see e.g. [36], [63], [74] and [64].

In final category are the fitness diversity-adaptive schemes, which analyze the fitness diversity of the current population and decide, for example, if local search component should be activated. In a case where fitness diversity is low an alternative search logic can be applied to explore other regions of the search space or if this fails to increase diversity more exploitative meme can be selected to finalize the search. Examples of fitness diversity-adaptive schemes are studied e.g in [4] and [69].

In the following section a simple memetic structure, with a deterministic coordination scheme based on the success of memes, is described more in detail.

4.2 Single Solution Memetic Structure

Three Stage Optimal Memetic Exploration (3SOME), proposed in [29], is a single solution optimizer designed for devices plagued by a limited hardware, such as embedded systems or simple mobiles. Above mentioned algorithm was originally implemented with the concept of "Ockham's Razor" in mind: "The simpler solution is better than the more complex one given the same performance". The 3SOME algorithm is a combination of three different components: stochastic long distance exploration (L), stochastic short distance exploration (M) and deterministic short distance exploration (S) and a memetic structure framework which guides the coordination between these three components. Design wise the components were added one by one until a certain level of performance was reached without complicating the resulting structure beyond necessity following the Ockham's Razor rule. The general working principles of components and the framework of 3SOME are described below.

4.2.1 Three Stage Optimal Memetic Exploration

During stochastic long distance exploration algorithm is looking for promising basins of attraction over the whole search space **D** by sampling uniformly distributed random trial solution in **D** and then performing crossover between trial x_t and current best solution x_e. The crossover operator used is exponential crossover from Differential Evolution, see 3.3.2 on page 27. Exponential crossover is applied with high crossover rate Cr, thus most genes of the trial solution are copied making crossover highly explorative. Retaining small portion of current elite's genes seem to be beneficial compared to purely stochastic blind search (which would generate totally random solution). Long distance search is active until algorithm is able to improve on current best solution. Upon improving the best solution, the stochastic short distance search is activated to focus search around hopefully more promising area.

In stochastic short distance exploration a hyper-cube is generated around best solution to focus search to a hopefully more promising area. A number of points N (N equal to k times dimensionality of the problem) are sampled within the hyper-cube and crossover is performed with the best solution using exponential crossover with mediocre crossover rate. Crossover rate is altered (lowered) from long distance search to focus the search. Each time new solution outperforms current best the hyper-cube is centered around the new solution and process is continued until process is repeated N times. If during the N samples the M managed to improve the initial solution, it is activated again for another N repeats. On the other hand if M did not lead to improvement of the initial solution the next search logic, deterministic local search, is activated.

The purpose of deterministic local search is to exploit the current solution and finalize the optimization process, if the current solution lies in the same basin of attraction as the global optima. The description of deterministic local search is shown in section 1.2.4 on page 14. The deterministic local search is run for predefined budget of fitness evaluations and if solution is improved, within this budget, the stochastic short distance search (M) is activated subsequently. In the case of failure of improving the solution the long distance searcher (L) is activated. The reasoning behind which component is activated after S completes can be explained by following logic: in the case where local search cannot improve the current solution it is (or is close to) either local or global optima, thus long distance search is activated to escape possible local optima. In terms of fitness evaluations local search can be ineffective global optimizer due to it's exploitative nature by performing search moves limited to the coordinate axis. Thus, in the case of success on improving the initial solution the stochastic short distance search (M) is activated again to provide complementary search moves also with higher convergence rate. For the sake of clarity, the overall coordination scheme of the three components L, M and S is show in figure 5 and pseudo-codes for L,M and S can be seen in algorithms 6, 7 and 1 (see page 14) respectively.

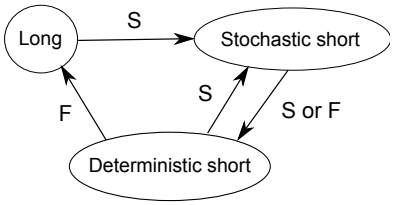


FIGURE 5 Functioning scheme of 3SOME. Arrows indicate transitions between active component and letters S and F indicate success and failure on component improving solution.

Algorithm 6 Long distance exploration

generate a random solution x_t within D
generate $i = round\,(n \cdot rand\,(0,1))$
$x_t[i] = x_e[i]$
while $rand\,(0,1) \leq Cr$ **do**
  $x_t[i] = x_e[i]$
  $i = i + 1$
  if $i == n$ **then**
    $i = 1$
  end if
end while
if $f\,(x_t) \leq f\,(x_e)$ **then**
  $x_e = x_t$
end if

Algorithm 7 Stochastic short distance exploration

generate a hypercube around x_e with side width λ
for $i = 1 : N$ **do**
  generate randomly a trial solution x_t within the hypercube
  generate $i = round(n \cdot rand(0,1))$
  while $rand(0.1) \leq Cr$ **do**
    $x_t[i] = x_e[i]$
    $i = i + 1$
    if $i == N$ **then**
      $i = N$
    end if
  end while
  if $f(x_t) \leq f(x_e)$ **then**
    $x_e = x_t$
  end if
end for

5 CONTRIBUTION OF THIS WORK

In this chapter the contribution of the each article included in this work is described. Each article included, except from article [PVII], proposes a new algorithmic structure, where resulting algorithms can be categorized into two groups: single solution optimizers based on Three Stage Optimal Memetic Exploration (3SOME) and memetic structures of Differential Evolution. More specifically, in the articles [PI] and [PII] modifications to operators of 3SOME are proposed regarding scalability and complementary search moves respectively. Article [PIII] introduces a modified structure of the 3SOME-algorithm by including concept of the Meta-Lamarckian learning as a coordination scheme, see section 4.1 on page 37, and article [PIV] is a revisited version of article [PI] extending the analysis and benchmark problems included.

Articles [PV] and [PVI] both propose a novel memetic computing algorithm by introducing modified structure of Differential Evolution integrating a local search component. Article [PVII] introduce an additional component for DE and clustering-based population initialization, which aim to improve DE-based algorithms in general. It must be noted that the local searcher described in algorithm 1 is applied (to some extent) on all of the algorithms presented in the articles.

Remainder of this chapter is organized as follows: the test framework is presented in section 5.1, articles related to memetic computing framework of 3SOME are discussed in section 5.2, articles introducing modifed structures of DE are discussed in section 5.3, and article [PVII] is discussed in a separate section 5.4.

5.1 Test Framework

Algorithms introduced in the articles are compared in artificial benchmark problems, see BBOB2010 [23], CEC2013 [39], CEC2010 [70], CEC2005 [67], and a real world benchmark problem [16] (in article [PVII]), which provide wide variety of test problems with multiple characteristics such as separability, modality, symmetry, etc. As optimization in general is a stochastic process, one run does not

necessarily reflect the performance of the algorithm. Thus, multiple runs are performed to determine average performance and related standard deviation. Also, as average is not always the best measure of performance statistical tests such as Wilcoxon Rank-sum test [72], which aim to determine if samples of two different distributions are identical giving certain confidence level, is applied on each of the articles included. Also the values of the best, worst and median results are included in the article [PVI] as required by organizers of Congress on Evolutionary Computing 2013 (CEC2013) [40]. To compare overall performance of the algorithms on a set of problems the Holm-Bonferroni method, see [25], is also applied in articles [PV], [PVI], and [PVII].

It should be mentioned that all the algorithms use toroidal transformation for bounded problems, see [53]. More specifically, when a new point x is generated if one it's components (design variables) x_i is not from the bounded interval $[a, b]$ defined by the design space, the value for x_i is computed by following formula:

$$\begin{cases} x_i = a + x_i - b & \text{if} \quad x_i > b \\ x_i = b - x_i + a & \text{if} \quad x_i < a \end{cases} \tag{34}$$

this guarantees that every new point belongs to the search space.

5.2 Modified Structures of Three Stage Optimal Memetic Exploration

Three Stage Memetic Optimal Memetic Exploration algorithm consists of three different components, each tailored to specific "role" in the optimization process. Where long distance exploration focuses on finding promising solutions from whole design space, the other two components have more exploitative role and focus the search on hopefully more promising area. While modifications can be applied on the components itself, one can also modify the behavior of co-operative framework which controls the transitions between components deciding which component is active. Instances of both of these approaches are given in the examples below.

Articles [PI] and [PIV] focus on the scalability of the 3SOME algorithm to higher dimensional problems. Goal of this approach is to have an algorithmic structure which retains it's behavior independent of problem dimensionality in terms of explorative and exploitative search moves. The article [PI] proposes a modified scheme, namely Shrinking Three Stage Optimal Memetic Exploration (S-3SOME), where the stochastic short distance exploration is modified to be scalable with dimensionality of the problem. Scalability is achieved by redefining initial hyper-cube constructed around solution such that it has fixed hyper-volume of 20 percent of search space hyper-volume. In addition if the search fails to improve the current solution the hyper-volume is halved and a new set of N solutions are sampled within the hyper-cube. In the case of success the search

is repeated within same hyper-cube. These steps are repeated until the certain threshold of hyper-volume is met and the algorithm activates the deterministic local search component. The role of the modified stochastic short distance search is more explorative at the beginning and shifts towards more exploitative while hyper-cube shrinks. Article [PIV] is a revision of article [PI] and extends the analysis and the problems included in the article [PI]. Pseudo-code for modified component is given in algorithm 8.

Algorithm 8 Modified stochastic short distance exploration

```
generate a hypercube around elite x_e with a hyper-volume 20% of that of D;
while the hyper-volume is bigger than 0.0001% of D do
  for i = 1 : n do
    generate randomly a trial solution x_t within the hypercube;
    generate i = round(n · rand(0, 1))
    while rand(0.1) ≤ Cr do
      x_t[i] = x_e[i]
      i = i + 1
      if i == N then
        i = 1
      end if
    end while
    if f(x_t) ≤ f(x_e) then
      x_e = x_t;
      center the hypercube around x_e;
    end if
  end for
  if no elite update occurred then
    halve the hyper-volume;
  end if
end while
```

One of the of weak-points in original implementation of 3SOME was that all the components perturbed current solution only along the coordinate axes, thus lacking diagonal search moves. This issue is addressed in article [PII], which proposes a rotation invariant mutation operator DE/current-to-rand/1, as described earlier in section 3.2, to be applied instead of exponential crossover in stochastic short distance search (M). The aim here is to increase the algorithms performance on non-separable problems, while not suffering too much on separable problems, and retain the simple structure of 3SOME.

While the articles discussed above focus on the operators of the 3SOME algorithm, article [PIII] proposes an adaptive coordination scheme between the three major components. Meta-Lamarckian learning is an efficient adaptive scheme defined in [51], where meta-Lamarckian learning is used in the coordination of different local search components within a Memetic Algorithm (MA) framework. At the beginning, adaptive scheme organizes each component into a pool and

assigns each an equal selection probability. Active component is selected based on the selection probability and this probability is updated during the optimization process depending on the success history of the component. While adaptive scheme performs better on some problems, it can be observed that the original 3SOME coordination scheme can provide promising results especially in the early stages of optimization. However, one observation which can be done on the basis of this study is that different coordination schemes can lead to a different performance even when the algorithmic operators remain the same. Generic pseudo-code for the Meta-Lamarckian adaptive coordination scheme can be seen in algorithm 9. In the pseudo-code, variables P_L, P_M and P_M represent the probabilities assigned to each component L, M and S respectively.

Algorithm 9 Meta-Lamarckian coordination

```
t ← 0
while termination condition is not met do
  Update probabilities P_L(t), P_M(t) and P_S(t)
  generate uniformly distributed random number U = rand(0, 1)
  Select component S, M or L by a roulette wheel selection based on U and
  P_i(t)
  t ← t + 1
end while
```

All the proposed variations of the Three Stage Optimal Memetic Exploration are simple memetic computing structures with modest hardware requirements while remaining competitive with more complex modern state-of-art algorithms. These features make them valid optimizers especially for environments where only limited hardware resources are available and/or rapid response times are required.

5.3 Differential Evolution based Memetic Structures

Micro-Evolution Algorithms (Micro-EAs) are instances of evolutionary algorithms characterized with a small population size. While originally used for an educational purposes, attempts on applying Micro-EAs for real-world problems have been proposed in the literature. One of the first implementations of micro-algorithms is micro-Genetic Algorithm (μGA) proposed in [11] and [37], in [56] micro-Differential Evolution (micro-DE or μDE) with opposition-based mechanism has been used to solve image thresholding problems, and recently in [65] for evolving bin packing problem. Having a small population usually offers a rapid convergence but with a cost of efficiency: micro-algorithms often converge into a local optima as having a small population limits exploration capabilities. However, using micro-algorithms on problems with high-dimensionality can be beneficial due to their fast convergence properties. In general, solving

high-dimensional problems is challenging as the search space grows exponentially when dimensionality increases.

Algorithm 10 μDEA

Generate an initial population of Np individuals and computer their fitness values
while termination condition is not met **do**
  for $i = 1 : Np$ **do**
    select three individuals x_r, x_s and x_t
    compute mutant individual $x_{i,off} = x_t + F(x_r - x_s)$
    compute offspring x_{off} by exponential crossover.
    compute fitness $f(x_{off})$
  end for
  perform selection
  if $rand(0,1) < \eta$ **then**
    perform deterministic local search on x_p
  end if
end while

In article [PV] μDE is integrated with a deterministic local search (S). Micro-DE component uses DE/rand/1 mutation strategy with exponential crossover. After each iteration of μDE the local search component is activated on a best performing population member with certain probability η. Deterministic local search has a large initial step-size to attempt exploratory search moves along the axis complementing limited search moves of the μDE. The resulting algorithm called micro-Differential Evolution with Extra Moves Along the Axes (μDEA) is compared against standard μDE, and more modern algorithms with a standard population size: SADE, JADE and MDE-pBX, on a variety of problems up to 1000 design variables. μDEA shows respectable performance against comparison algorithms especially on high-dimensional problems. General pseudo-code for μDEA can be seen in algorithm 10.

Building upon the structure of μDEA, a novel algorithm design has been proposed in article [PVI], namely Differential Evolution with Concurrent Fitness Based Local Search (DEcfbLS), for pseudo-code see algorithm 11. Algorithm was designed to take part into competition on real-parameter single objective optimization held in Congress on Evolutionary Computation 2013 (CEC-2013), see [40]. Similarly to μDEA, DEcfbLS is a combination of DE/rand/1/exp scheme, described in sections 13 and 3.3.2, and a deterministic local search (S) (shown in 1.2.4). However, DEcfbLS differs from μDEA as more a standard population size is used rather than micro population and also from the memetic structure which controls the coordination between components. Alternation between components follows a deterministic rule: both components should get close to equal amount of fitness evaluations (FEs) over the course of the optimization run. Thus, the number of (FEs) needs to be predefined. In addition, when the local search is activated, as S is a single-point optimizer, S is applied on solutions that have

better than average fitness of the population. At the beginning of the optimization, DEcfbLS also employs a population refining strategy where S is activated on each of the solutions in the initial population with shallow depth to perform explorative steps (using large step size) along axes. The resulting algorithm is compared against state-of-art algorithms in a benchmark provided by the organizers of the competition, see [39]. Also further comparison is done by the organizers against the other algorithms accepted into the competition. The results, performed by the organizers, are presented at [30].

Algorithm 11 DEcfbLS

```
generate randomly NP individuals of the initial population and compute their
fitness values
for i = 1 : NP do
  apply LS on i with local budget of 4 iterations.
end for
compute fitness evaluation breakpoint FEbp
while termination condition is not met do
  for i = 1 : NP do
    select three individuals x_r, x_s, and x_t
    compute mutant individual x_o = x_t + F(x_r − x_s)
    compute exponential crossover between x_i and x_o thus generating x_off
    if f(x_off) ≤ f(x_i) then
      save index
    end if
  end for
  perform survivor selection where proper on the basis of saved indexes
  if iterations i is greater or equal than the breakpoint FEbp then
    compute populations fitness average f_avg
    for i = 1 : NP do
      if f(i) < f_avg then
        apply LS on i with local budget of 40 iterations.
      end if
    end for
    update breakpoint FEbp
  end if
end while
```

5.4 Cluster-Based Population Initialization in Differential Evolution

In the final article [PVII] included in this work a new clustering-based population initialization (CBPI) strategy for DE-based algorithms is proposed. In classic

DE the initial population is usually selected by random from uniform distribution, as described earlier in 3.1. In CBPI the initialization is emphasized on more promising areas detected by a local search and a clustering method. The whole procedure of CBPI can be divided into three different stages: Optimization by local search, clustering, and Cluster-Based Population Initialization. First two stages are run before the DE framework and the last stage replaces the population initialization within the DE framework. The CBPI is tested on multiple modern DE variants, on a wide spectrum of problems from different benchmarks, and on a Lennard-Jones Potential problem. The proposed module implicitly performs an analysis on multi-modality and estimates the location of the strongest basins of attraction. Then, the module exploits this piece of information to "suggest" DE where to search for the global optimum. Description of the three stages are briefly presented below.

5.4.1 Optimization by Local Search

At the first stage a number of Np solutions are pseudo-randomly generated to form an initial population. Similar to approach used in article **VI** , where population was refined with a deterministic local search, here the initial population is optimized for prefixed amount of fitness evaluations with two local searchers: S and Rosenbrock's, see 1.2.4 and 1.2.2 on page 13. The goal here is to locate more promising areas of search space by applying local search. The selection of these two local searchers is based on that they complement each other well, as observed in [6] and [7], while S performs search moves along the axes, Rosenbrock's local search uses gradient information to detect most promising search direction. After local search, cluster analysis is performed on the current population.

5.4.2 Clustering and Cluster Evaluation

In the second stage of CBPI clustering analysis is performed to check how points are spread along search space. As an output this phase produces candidate solutions divided into k clusters. Clustering is made by means of similarity in Euclidean distance between points. The clustering algorithm used is k-means, see [41], which needs the amount of clusters k to be predefined, thus the algorithm is run with different values of k. To decide which of these k clusterings is the most suitable one, the clusters are evaluated by computing an average Silhouette of the cluster, see [61]. The goal here is to get an estimate of function's modality: if points belong into a single cluster, the problem is most likely uni-modal and if multiple clusters are detected the function is more likely multi-modal with several local optimas. While being only an estimate, this information is used in third stage to create a new population around the most promising solutions in each cluster, thus giving weight but not limiting the search to these areas (as they may be local optimas). For the sake of clarity, pseudo-code of the second stage is provided in algorithm 12.

Algorithm 12 Pseudo-code of the Second Stage.

INPUT S_{pop} candidate solutions improved by two subsequent local search algorithms
for $k = 2 : M$ **do**
    Execute k-means with k clusters
    save clustering
end for
for $k = 2 : M$ **do**
    Evaluate clustering k by average Silhouette
end for
select the best clustering scenario on the basis of the average Silhouette value
OUTPUT S_{pop} candidate solutions divided into k clusters

5.4.3 Cluster-Based Population Initialization

In the final stage of the CBPI, clustering information is used to initialize population around the detected clusters. Each cluster is characterized by their best individual x_k (*pivot individual*). First, all the k pivot individuals (one for each cluster) are copied into the population. The pivot individuals are the only solutions which get copied into population used within DE framework and all other solutions are essentially ignored. The remaining population members are generated as described below.

When initializing a new point into the population, single cluster best x_k is selected by a roulette wheel selection giving more weight on more promising clusters. Initialization of points is done by generating pseudo-random solution with normal distribution around point x_k, thus giving weight but not being limited to neighborhood of point x_k. More specifically, each design variable $x[i]$ is sampled with normal distribution with mean $x_k[i]$ and standard deviation σ, where $\sigma = 0.1 \cdot W$ and W is width of the design variable i. Illustration of 2-dimensional case is show in picture 6 and pseudo-code describing third stage more in detail is given in algorithm 13.

Each of the stages described above have a specific role: the first stage improves initial solutions locally and attempts to map promising basins of attraction. The second stage attempts to organize promising basins into clusters estimating multi-modality of the problem. The third stage utilize the information gained from the second stage to fill in population around the basins of attraction, while not completely excluding the other areas of the search space. Results show that the intelligent sampling of the search space is able to consistently improve the performance of various DE frameworks.

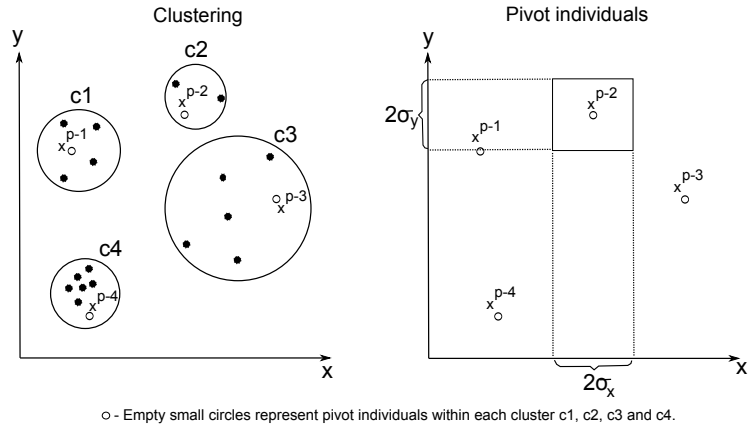


FIGURE 6 2-dimensional illustration of clusters Ck and their respective pivot individuals x^{p-k}. In stage three pivot individuals are used as mean values with standard deviations σ_x and σ_y to fill in remaining population.

Algorithm 13 Pseudo-code of the Third Stage.

INPUT S_{pop} candidate solutions divided into k clusters
for $i = 1 : k$ **do**
  select, for each cluster the individual with the best fitness $\mathbf{x^{p-i}}$
  $\mathbf{Pop^k} = \mathbf{x^{p-k}}$
end for
for $i = 1 : K$ **do**
  assign to the cluster a score sc^k on the basis of $f\left(\mathbf{x^{p-i}}\right)$
  calculate $pc^i = \frac{sc^i}{\sum_{j=1}^{K} sc^j}$
end for
for $i = K + 1 : S_{pop}$ **do**
  generate a random number $rand\,(0, 1)$
  select a cluster k by means of the roulette wheel selection and the assigned probabilities
  for $j = 1 : n$ **do**
    $Pop_j^i = \mathcal{N}\left(x_j^{p-k}, 1\right)$
  end for
end for
OUTPUT **Pop**

6 CONCLUSION

The goal of Computational Intelligence Optimization (CIO) is ultimately to be able to generate a system capable of understanding problem specific features and select appropriate algorithmic components/memes for intelligent problem solving. As this goal is very ambitious, it's not within the limits of this thesis. However, steps toward this goal can be achieved by understanding memetic structures and components (memes) involved in the optimization algorithms. Memetic Computing is a subject in computer science that studies optimization structures and memes involved from CIO perspective. A cornerstone for MC approaches are Memetic Algorithms, which by classic definition includes an evolutionary framework with one or multiple local search components.

Memetic Computing approaches presented in this work are related to both single-solution and population-based algorithms and combinations of them as memetic structures. In the first category of single-solution optimizers, modifications of the Three Stage Optimal Memetic Exploration algorithm are proposed. These modifications aim to address problems in the original implementation of the above mentioned memetic structure, more specifically, features such as dimensionality and separability. As exploration and exploitation properties are ingredients for any successful optimization algorithm, the components of an algorithm need to be able to retain their role as the problem dimension increases. This can be a very challenging task and is known as the curse of dimensionality. Scalability in Three Stage Optimal Memetic Exploration is addressed with a shrinking mechanism that scales with dimension and is more explorative at the beginning of the optimization process shifting towards more exploitative each iteration. Regarding separability, the original algorithm lacks so called rotation invariant search moves, which can limit the performance especially on rotated or non-separable problems. To address this problem rotation invariant mutation is included within one of the three components. Also an alternative memetic structure for the algorithm is presented, which employs meta-Lamarckian learning. The results show that exactly same memes can achieve different results with different memetic structures.

Two Memetic Computing approaches with Differential Evolution as the evo-

lutionary framework are included. Differential Evolution, while being efficient population-based optimization algorithm, contains an inherent flaw which can limit its performance. Because of Differential Evolution's limited search moves stagnation can occur. In this case Differential Evolution is unable to generate an offspring that outperforms its parent. Various research directions have been explored in order to avoid stagnation and premature convergence. One such approach is to include complementary search moves as an additional component. Including simple local search with an intelligent memetic structure can help Differential Evolution to avoid stagnation and increase the performance of the algorithm in general. In this work, local search is integrated with micro-Differential Evolution and a classical Differential Evolution to tackle a wide-range of optimization problems. Building upon the concept, a general component for Differential Evolution (or another population-based algorithm) is presented, which intelligently samples the design space before initialization of the population, thus, guiding the optimization algorithm. The component itself is a memetic structure consisting of two local searchers: Rosenbrock's method and Deterministic Local Search, which complement each other in terms of search moves.

A main finding in this work is that even simple algorithmic components, when selected wisely to address problem specific features and implemented with intelligent co-operative structure, can attain comparable performance on various global optimization problems. Also local search has been utilized, at least in the view of the author, with a more broad role for both explorative and exploitative purposes.

YHTEENVETO (FINNISH SUMMARY)

Perusajatus optimoinnissa on päästä parhaaseen mahdolliseen ratkaisuun tietyillä rajoitteilla. Optimointiongelmia esiintyy kaikkialla, esimerkkinä pakkausongelma, jossa annetut palaset tulisi sijoittaa rajoitettuun tilaan. Ongelma on yleinen esimerkiksi vaateteollisuudessa. Kyse voi olla myös optimaalisen muodon ja/tai materiaalin löytämisestä lentokoneen siipeä suunnitellessa tai traktorinkopin äänieristyksessä. Optimointia tarvitaan arkielämässä, liike- ja talouselämässä, sekä tekniikassa.

Yksi optimointimenetelmien keskeisistä tavoitteista on pyrkimys kehittää itsenäinen systeemi, joka kykenee analysoimaan optimointiongelmalle ominaisia piirteitä ja pystyy niiden perusteella älykkäästi valitsemaan ratkaisumenetelmän. Tavoite on kuitenkin hyvin kunnianhimoinen eikä se ole tämän väitöskirjan saavutettavissa. On kuitenkin mahdollista ottaa askel kohti tätä tavoitetta tutkimalla optimointimenetelmissä esiintyviä memeettisiä struktuureja sekä komponentteja. Memeettinen laskenta on tieteellisen laskennan ala, joka tutkii optimointimenetelmien struktuureja laskennallisen älykkyyden perspektiivistä. Yksi memeettisen laskennan tunnettu osa-alue on memeettiset algoritmit. Memeettiset algoritmit määritellään klassisesti optimointi algoritmeina, joissa on populaatiopohjainen optimointimenetelmä yhdistettynä lokaalihakumenetelmään.

Tässä työssä esitellyt memeettisen laskennan metodit käsittelevät yhden ratkaisun malleja, populaation pohjaisia metodeja sekä näiden yhdistelmiä. Yhden ratkaisun mallit keskittyvät "Three Stage Optimal Memetic Exploration-menetelmän muunnoksiin. Nämä muunnokset tähtäävät korjaamaan alkuperäisen menetelmän heikkouksia tietyissä ongelmatyypeissä. Näiden muunnoksien lisäksi esitellään vaihtoehtoinen struktuuri, joka ohjaa siirtymiä menetelmän kolmen eri vaiheen välillä. Edellä mainitun menetelmän laskennallisista tuloksista voidaan nähdä, että samoilla algoritmisilla komponenteilla struktuuria vaihtamalla voidaan päätyä erilaisiin tuloksiin ongelmatyypistä riippuen.

Lisäksi työhön liitetyissä artikkeleissa esitellään kaksi differentiaalievoluutioon (DE) perustuvaa memeettistä struktuuria. Differentiaalievoluutio on tehokas optimointimenetelmä, jossa on kuitenkin ominaisuus, joka voi rajoittaa sen suorituskykyä. Tämä rajoittava ominaisuus johtuu DE:n tavasta tuottaa ratkaisua, joita on rajoitettu määrä. Mikäli yksikään näistä ratkaisuista ei johda ratkaisuun, joka olisi edeltäjäänsä parempi, voi ratkaisujen evoluutio pysähtyä. Tältä tilanteelta voidaan välttyä esimerkiksi lisäämällä uusia tapoja tuottaa ratkaisuja, esimerkiksi yhdistämällä jokin toinen optimointimenetelmä differentiaalievoluutioon. Työssä esitellään kaksi differentiaalievoluution muunnosta, joista toinen keskittyy mikro-differentiaalievoluutioon, jossa käytetään pientä populaatiokantaa, ja toinen klassiseen differentiaalievoluutioon. Edellä mainittuihin algoritmeihin perustuen esitellään myös yleinen komponentti differentiaalievoluutioon perustuville algoritmeille. Tämä komponentti arvioi lokaaleiden optimien määrää ja keskittää alkuperäispopulaation alustuksen älykkäästi tähän informaatioon perustuen.

Työssä esitellyt algoritmit koostuvat yksinkertaisista komponenteista, siten niiden käyttötarkoitus säilyy selkeänä sekä suunnittelijalle että käyttäjälle. Lisäksi algoritmit ovat helposti toteutettavissa. Huolimatta yksinkertaisuudestaan, esitellyt menetelmät voivat tarjota varteenotettavan vaihtoehdon globaaleiden optimointiongelmien ratkaisemisessa.

REFERENCES

[1] J. E. BAKER, *Reducing Bias and Inefficiency in the Selection Algorithm*, in Proceedings of the International Conference on Genetic Algorithms, Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA, 1987, pp. 14–21.

[2] 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, 10 (2006), pp. 646–657.

[3] J. BREST AND M. S. MAUČEC, *Population size reduction for the differential evolution algorithm*, Applied Intelligence, 29 (2008), pp. 228–247.

[4] 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, 13 (2009), pp. 811–831.

[5] F. CARAFFINI, F. NERI, G. IACCA, AND A. MOL, *Parallel memetic structures*, Information Sciences, 227 (2013), pp. 60 – 82.

[6] ——, *Parallel memetic structures*, Information Sciences, 227 (2013), pp. 60 – 82.

[7] F. CARAFFINI, F. NERI, AND L. PICINALI, *An analysis on separability for memetic computing automatic design*, Information Sciences, 265 (2014), pp. 1–22.

[8] R. A. CARUANA, L. J. ESHELMAN, AND J. D. SCHAFFER, *Representation and hidden bias ii: Eliminating defining length bias in genetic search via shuffle crossover*, in Proceedings of the 11th international joint conference on Artificial intelligence-Volume 1, Morgan Kaufmann Publishers Inc., 1989, pp. 750–755.

[9] U. K. CHAKRABORTY, ed., *Advances in Differential Evolution*, vol. 143 of Studies in Computational Intelligence, Springer, 2008.

[10] U. K. CHAKRABORTY, S. DAS, AND A. KONAR, *Differential Evolution with Local Neighborhood*, in Proceedings of the IEEE Congress on Evolutionary Computation, 2006, pp. 2042–2049.

[11] C. A. C. C. COELLO AND G. T. PULIDO, *A micro-genetic algorithm for multiobjective optimization*, in Evolutionary multi-criterion optimization, Springer, 2001, pp. 126–140.

[12] A. COLORNI, M. DORIGO, V. MANIEZZO, ET AL., *An investigation of some properties of an"ant algorithm"*., in PPSN, vol. 92, 1992, pp. 509–520.

[13] CYBER DYNE SRL HOME PAGE, *Kimeme*, 2012. http://cyberdynesoft.it/.

[14] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, *Differential Evolution with a Neighborhood-based Mutation Operator*, IEEE Transactions on Evolutionary Computation, 13 (2009), pp. 526–553.

[15] S. Das, A. Konar, and U. K. Chakraborty, *Two improved differential evolution schemes for faster global search*, in Proceedings of the 2005 conference on Genetic and evolutionary computation, ACM, 2005, pp. 991–998.

[16] S. Das and P. Suganthan, *Problem definitions and evaluation criteria for cec 2011 competition on testing evolutionary algorithms on real world optimization problems*, Jadavpur Univ., Nanyang Technol. Univ., Kolkata, India, (2010).

[17] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computation*, Springer Verlag, Berlin, 2003, pp. 175–188.

[18] L. J. Eshelman, R. A. Caruana, and J. D. Schaffer, *Biases in the crossover landscape*, in Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers Inc., 1989, pp. 10–19.

[19] H.-Y. Fan and J. Lampinen, *A Trigonometric Mutation Operation to Differential Evolution*, vol. 27, 2003, pp. 105–129.

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

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

[22] D. Gross and C. M. Harris, *Fundamentals of Queueing Theory*, Wiley, NY, 1985.

[23] N. Hansen, A. Auger, S. Finck, and R. Ros, *Real-Parameter Black-Box Optimization Benchmarking 2010: Experimental Setup*, Tech. Report RR-7215, INRIA, 2010.

[24] F. Herrera, M. Lozano, and A. M. Sánchez, *A taxonomy for the Crossover Operator for Real-Coded Genetic Algorithms: An Experimental Study*, International Journal of Intelligent Systems, 18 (2003), pp. 309–338.

[25] S. Holm, *A simple sequentially rejective multiple test procedure*, Scandinavian Journal of Statistics, 6 (1979), pp. 65–70.

[26] R. Hooke and T. A. Jeeves, *Direct search solution of numerical and statistical problems*, Journal of the ACM, 8 (1961), pp. 212–229.

[27] C. Houck, J. A. Joines, M. G. Kay, and J. R. Wilson, *Empirical investigation of the benefits of partial lamarckianism*, Evolutionary Computation, 5 (1997), pp. 31–60.

[28] 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, 188 (2012), pp. 17–43.

[29] G. Iacca, F. Neri, E. Mininno, Y.-S. Ong, and M.-H. Lim, *Ockhams razor in memetic computing: three stage optimal memetic exploration*, Information Sciences, 188 (2012), pp. 17–43.

[30] T. S. Ilya Loshchilov and T. Liao, *Ranking results of cec´13 special session & competition on real-parameter single objective optimization at CEC2013*, tech. report, 2013. http://www.ntu.edu.sg/home/EPNSugan/index_files/CEC2013/results_21.pdf.

[31] H. John, *Holland, adaptation in natural and artificial systems*, 1992.

[32] J. Kennedy and R. C. Eberhart, *Particle swarm optimization*, in Proceedings of IEEE International Conference on Neural Networks, 1995, pp. 1942–1948.

[33] J. Kiefer, *Sequential minimax search for a maximum*, Proceedings of the American Mathematical Society, 4 (1953), pp. 502–506.

[34] S. Kirkpatrick, C. D. J. Gelatt, and M. P. Vecchi, *Optimization by simulated annealing*, Science, (1983), pp. 671–680.

[35] A. V. Kononova, K. J. Hughes, M. Pourkashanian, and 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.

[36] N. Krasnogor and J. Smith, *A tutorial for competent memetic algorithms: model, taxonomy, and design issues*, IEEE Transactions on Evolutionary Computation, 9 (2005), pp. 474–488.

[37] K. Krishnakumar, *Micro-genetic algorithms for stationary and non-stationary function optimization*, in 1989 Advances in Intelligent Robotics Systems Conference, International Society for Optics and Photonics, 1990, pp. 289–296.

[38] J. Lampinen and I. Zelinka, *On stagnation of the differential evolution algorithm*, in Proceedings of MENDEL, 2000, pp. 76–83.

[39] J. Liang, B. Qu, P. Suganthan, and A. G. Hernández-Díaz, *Problem definitions and evaluation criteria for the cec 2013 special session on real-parameter optimization*, Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou, China and Nanyang Technological University, Singapore, Technical Report, 201212 (2013).

[40] J. J. Liang, B. Y. Qu, P. N. Suganthan, and A. G. Hernández-Diaz, *Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session on*

Real-Parameter Optimization, Tech. Report 201212, Zhengzhou University and Nanyang Technological University, Zhengzhou China and Singapore, 2013.

[41] S. LLOYD, *Least squares quantization in pcm*, IEEE Transactions on Information Theory, 28 (1982), pp. 129–137.

[42] M. LOZANO AND C. GARCÍA-MARTÍNEZ, *Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report*, Computers & Operations Research, 37 (2010), pp. 481–497.

[43] D. MOLINA, F. HERRERA, AND M. LOZANO, *Adaptive local search parameters for real-coded memetic algorithms*, in Evolutionary Computation, 2005. The 2005 IEEE Congress on, vol. 1, IEEE, 2005, pp. 888–895.

[44] A. NELDER AND R. MEAD, *A simplex method for function optimization*, Computation Journal, Vol 7 (1965), pp. 308–313.

[45] F. NERI AND C. COTTA, *Memetic algorithms and memetic computing optimization: A literature review*, Swarm and Evolutionary Computation, 2 (2012), pp. 1–14.

[46] F. NERI, C. COTTA, AND P. MOSCATO, *Handbook of Memetic Algorithms*, vol. 379 of Studies in Computational Intelligence, Springer, 2011.

[47] F. NERI, G. IACCA, AND E. MININNO, *Disturbed Exploitation compact Differential Evolution for Limited Memory Optimization Problems*, Information Sciences, 181 (2011), pp. 2469–2487.

[48] F. NERI AND V. TIRRONEN, *Scale Factor Local Search in Differential Evolution*, Memetic Computing Journal, 1 (2009), pp. 153–171.

[49] ——, *Recent Advances in Differential Evolution: A Review and Experimental Analysis*, Artificial Intelligence Review, 33 (2010), pp. 61–106.

[50] F. NERI, V. TIRRONEN, AND T. KÄRKKÄINEN, *Enhancing Differential Evolution Frameworks by Scale Factor Local Search - Part II*, in Proceedings of the IEEE Congress on Evolutionary Computation, CEC'09, Piscataway, NJ, USA, 2009, IEEE Press, pp. 118–125.

[51] Y. S. ONG AND A. J. KEANE, *Meta-Lamarkian Learning in Memetic Algorithms*, IEEE Transactions on Evolutionary Computation, 8 (2004), pp. 99–110.

[52] K. PRICE, *An Introduction to Differential Evolution*, in New Ideas in Optimization, D. Corne, M. Dorigo, F. Glover, D. Dasgupta, P. Moscato, R. Poli, and K. V. Price, eds., McGraw-Hill, 1999, pp. 79–108.

[53] K. V. PRICE, R. STORN, AND J. LAMPINEN, *Differential Evolution: A Practical Approach to Global Optimization*, Springer, 2005.

[54] 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, 13 (2009), pp. 398–417.

[55] A. K. Qin and P. N. Suganthan, *Self-adaptive differential evolution algorithm for numerical optimization*, in Proceedings of the IEEE Congress on Evolutionary Computation, vol. 2, 2005, pp. 1785–1791.

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

[57] S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, *Opposition-Based Differential Evolution*, IEEE Transactions on Evolutionary Computation, 12 (2008), pp. 64–79.

[58] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, *Opposition-Based Differential Evolution*, in Advances in Differential Evolution, U. K. Chakraborty, ed., vol. 143 of Studies in Computational Intelligence, Springer, 2008, pp. 155–171.

[59] I. Rechenberg, *Evolutionsstrategie – Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*, PhD thesis, Technical University of Berlin, 1971.

[60] H. H. Rosenbrock, *An automatic Method for finding the greatest or least Value of a Function*, The Computer Journal, 3 (1960), pp. 175–184.

[61] P. J. Rousseeuw, *Silhouettes: A graphical aid to the interpretation and validation of cluster analysis*, Journal of Computational and Applied Mathematics, 20 (1987), pp. 53 – 65.

[62] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach (2nd ed.)*, Prentice Hall, 2003.

[63] J. E. Smith, *Coevolving Memetic Algorithms: A Review and Progress Report*, IEEE Transactions on Systems, Man, and Cybernetics, Part B, 37 (2007), pp. 6–17.

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

[65] 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, Springer, 2013, pp. 349–359.

[66] R. Storn and K. Price, *Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces*. TR-95-012, 1995.

[67] 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*, KanGAL Report, 2005005 (2005).

[68] K. TAGAWA, *A statistical study of the differential evolution based on continuous generation model*, in Evolutionary Computation, 2009. CEC'09. IEEE Congress on, IEEE, 2009, pp. 2614–2621.

[69] J. TANG, M. H. LIM, AND Y. S. ONG, *Diversity-Adaptive Parallel Memetic Algorithm for Solving Large Scale Combinatorial Optimization Problems*, Soft Computing-A Fusion of Foundations, Methodologies and Applications, 11 (2007), pp. 873–888.

[70] 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*, tech. report, University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL): Hefei, Anhui, China, 2010.

[71] V. TIRRONEN AND F. NERI, *Differential Evolution with Fitness Diversity Self-Adaptation*, in Nature-Inspired Algorithms for Optimisation, Studies in Computational Intelligence, R. Chiong, ed., vol. 193 of Studies in Computational Intelligence, Springer, 2009, pp. 199–234.

[72] F. WILCOXON, *Individual comparisons by ranking methods*, Biometrics Bulletin, 1 (1945), pp. 80–83.

[73] D. H. WOLPERT AND W. G. MACREADY, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation, 1 (1997), pp. 67–82.

[74] E. L. YU AND P. N. SUGANTHAN, *Ensemble of niching algorithms*, Information Sciences, 180 (2010), pp. 2815–2833.

[75] D. ZAHARIE, *Critical Values for Control Parameters of Differential Evolution Algorithm*, in Proceedings of 8th International Mendel Conference on Soft Computing, R. Matuŝek and P. Oŝmera, eds., 2002, pp. 62–67.

[76] A. ZAMUDA, J. BREST, B. BOŠKOVIĆ, AND V. ŽUMER, *Differential evolution for multiobjective optimization with self adaptation*, in Proceedings of the IEEE Congress on Evolutionary Computation, 2007, pp. 3617–3624.

[77] A. ZAMUDA, J. BREST, B. BOŠKOVIĆ, AND V. ŽUMER, *Large Scale Global Optimization Using Differential Evolution With Self-adaptation and Cooperative Co-evolution*, in Proceedings of the IEEE World Congress on Computational Intelligence, 2008, pp. 3719–3726.

[78] J. ZHANG AND A. SANDERSON, *Jade: Adaptive differential evolution with optional external archive*, Evolutionary Computation, IEEE Transactions on, 13 (2009), pp. 945–958.

ORIGINAL PAPERS

PI

SHRINKING THREE STAGE OPTIMAL MEMETIC EXPLORATION

by

I. Poikolainen, G. Iacca, F. Neri, E. Mininno, M. Weber 2012

Proceedings of the fifth international conference on bioinspired optimization methods and their applications, pages 61-74

SHRINKING THREE STAGE OPTIMAL MEMETIC EXPLORATION

Ilpo Poikolainen*, Giovanni Iacca †, Ferrante Neri*,
Ernesto Mininno*, Matthieu Weber*
* *Department of Mathematical Information Technology*
P.O. Box 35 (Agora), 40014 University of Jyväskylä, Finland
ilpo.poikolainen@jyu.fi, ferrante.neri@jyu.fi, ernesto.mininno@jyu.fi, matthieu.weber@jyu.fi
† *INCAS*^{3}*, Dr. Nassaulaan 9, 9401 HJ Assen, The Netherlands*
giovanniiacca@incas3.eu

Abstract The Ockham's razor in Memetic Computing states that optimization algorithms composed of few, simple, and tailored components can be very efficient, if properly designed. If the designer is aware of the role and effect of each algorithmic component an high performance can be easily obtained. Following this principle, this paper proposes a novel algorithm for numerical optimization. The proposed algorithm, namely Shrinking Three Stage Optimal Memetic Exploration (S-3SOME), performs the progressive perturbation of a candidate solution by alternating three search operators, the first is a stochastic global search, the second is random sampling within progressive narrowing hypervolume, the third is a deterministic local search. The proposed S-3SOME is an efficient scheme which outperforms, for the considered problems, a similar scheme proposed in literature and, despite its simplicity, is competitive with complex population-based algorithms which require massive overhead and memory employment.

Keywords: Memetic computing, Computational intelligence optimization, Algorithm for resource-constrained hardware

1. Introduction

During the latest years, modern research papers in numerical optimization claim to propose high performance general purpose algorithms. For practical reasons, these algorithms are usually tested on standardized sets of artificial test problems proposed in conferences and special issues and are compared with other recent algorithms. Commonly, new

algorithms are not really based on novel ideas, on the contrary, they are designed by following one of the design criteria mentioned below.

1) Starting from an existing optimization algorithm, its structure is "perturbed" by slightly modifying the structure and adding on extra components. Obviously, this approach attempts to obtain a certain performance improvement in correspondence to the proposed modifications. Successful examples of this research approaches are given in [2], where a controlled randomization on Differential Evolution (DE) parameters offers a promising alternative to the standard DE framework, and [5], where the variation operator combining the solutions of a population is modified in the context of Particle Swarm Optimization (PSO).

2) Starting from a set of algorithms, they are combined in a hybrid fashion with the trust that their combination and coordination leads to a flexible structure displaying a better performance than the various algorithms considered separately. Two examples of recently proposed algorithms which are basically the combination, by means of activation probabilities, of various meta-heuristics are given in [9] and [13].

Recently, algorithmic design has been tackled by means of the so-called Ockham Razor's principle in Memetic Computing (MC) or more generally in Computational Intelligence Optimization, see [4]. According to the Ockham's Razor, the simplest explanation of natural phenomena is likely to be the closest to the truth. In an analogous way, an optimization problem can be seen as a natural phenomenon and the optimization algorithm should be based on its understanding. This means that the optimization algorithm contains the countermeasures to handle the features of the fitness landscape. In order, on one hand, to avoid a waste of computational resources, and, on the other hand, to allow a proper algorithmic development when the fitness landscape changes (adding and removing memes/components/modules) only the strictly necessary components must be employed.

It must be remarked that we are referring to the concept of MC as reported in [7], MC is intended as "a broad subject which studies complex and dynamic computing structures composed of interacting modules (memes) whose evolution dynamics is inspired by the diffusion of ideas. Memes are simple strategies whose harmonic coordination allows the solution of various problems". In a metaphorical way, the proposed view on the subject considers the various operators as cooking ingredients. The ingredients require then to be properly selected and combined in order to address the specific taste and requirements of the guests. Thus, the algorithmic design should be performed by implementing a bottom-up procedure where the role and function of each operator should be clear. On the basis of this idea, in [4], it is shown how a very simple al-

gorithm, namely Three Stage Optimal Memetic Exploration (3SOME), which combines three perturbation mechanisms over a single solution, is competitive with complex algorithms representing the-state-of-the-art in Computational Intelligence Optimization, while requiring only little memory and computational resources (see [4] for a detailed analysis of the computational overhead and memory usage of 3SOME).

By following the *lex parsimoniae* principle of Ockham's Razor and the example of 3SOME algorithm, this paper proposes a simple algorithmic solution obtained by the sequential application of three perturbation procedures, one is a random global search which attempts to detect promising search directions, one is a random local search and the last is a deterministic local search. Two perturbation mechanisms are taken from the 3SOME algorithm while one of them has been replaced with a novel one. More specifically, while 3SOME is built-up on the idea that the search of the optimum should be reached by alternating the radius of the search and thus using long, middle, and short distance perturbation, this paper combines the long distance exploration with two short distance operators, with different roles, the first is stochastic and progressively focuses the search towards the most interesting areas of the decision space, the second is deterministic and perturbs each variable separately. The combination of these two different operators has been done on purpose to flexibly handle landscapes with diverse features. Since the stochastic operator performs the search by means of a progressive shrinking of the hypervolume, the proposed algorithm is indicated as Shrinking 3SOME (S-3SOME).

The remainder of this paper is organized in the following way. Section 2 gives a detailed description of the three operators and the coordination scheme composing the proposed algorithm. Section 3 shows the performance comparison between the proposed algorithm, 3SOME algorithm, and a set of modern algorithms. Finally, Section 4 gives the conclusive remarks of this study.

2. Shrinking Three Stage Optimal Memetic Exploration

In order to clarify the notation used, 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.

In the beginning of the optimization procedure one candidate solution is randomly sampled within the decision space D. In analogy with compact optimization, see [8], we will refer to this candidate solution as elite and indicate it with the symbol x_e. In addition to x_e, the algo-

rithm makes use of another memory slot for attempting to detect other solutions. The latter solution, namely trial, is indicated with x_t. In the following subsections the three exploratory stages and their coordination are described.

2.1 Long distance exploration

This exploration move attempts to detect a new promising solution within the entire decision space. While the elite x_e is retained, at first, a trial solution x_t is generated by randomly sampling a new set of n genes. Subsequently, the exponential crossover in the fashion of DE is applied between x_e and x_t, see [10]. More specifically, one gene from x_e is randomly selected. This gene replaces the corresponding gene within the trial solution x_t. Then, a set of random numbers between 0 and 1 are generated. As long as $rand\,(0,1) \leq Cr$, where the crossover rate Cr is a predetermined parameter, the design variables from the elite x_e are copied into the corresponding positions of the trial solution x_t. The first time that $rand\,(0,1) > Cr$, the copy process is interrupted. Thus, all the remaining design variables of the offspring are those initially sampled (belonging to the original x_t). This exploration stage performs the global stochastic search and thus attempts 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. This kind of inheritance of some genes appears to be extremely beneficial in terms of performance with respect to a stochastic blind search (which would generate a completely new solution at each step). If the trial solution outperforms the elite, a replacement occurs. A replacement has been set also if the newly generated solution has the same performance of the elite. This is to prevent the search getting trapped in some plateaus of the decision space (regions of the decision space characterized by a null gradient). The pseudo-code of this component is shown in Algorithm 1.

It can easily be observed that, for a given value of Cr, the meaning of the long distance exploration would change with the dimensionality of the problem. In order to avoid this problem and make the crossover action independent on the dimensionality of the problem, the following quantity, namely inheritance factor, is fixed: $\alpha_e \approx \frac{n_e}{n}$, where n_e is the number of genes we expect to copy from x_e into x_t in addition to the 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 we imposed that $Cr^{n\alpha_e} = 0.5$. 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: $Cr = \frac{1}{\sqrt[n\alpha_e]{2}}$. The long distance exploration 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 stochastic short distance exploration is activated.

Algorithm 1 Long distance exploration

```
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 if
end while
if f(x_t) ≤ f(x_e) then
    x_e = x_t
end if
```

2.2 Stochastic short distance exploration

This exploration move attempts to detect promising areas of the decision space by making use of a stochastic logic. In a nutshell, when the long distance exploration detects a new promising solution, the stochastic short distance exploration generates a hypercube centered in the newly detected solution x_e and having a hypervolume which is 20% of the decision space D. This exploration samples a point for n times and simply attempts to outperform solution x_e where n is dimensionality of the problem. In other words, for n times, a trial solution x_t is generated. If $f(x_t) \leq f(x_e)$, the elite solution is updated and the hypervolume is centered around the new elite solution. If, after n comparisons, at least one replacement occurred, n new samples of trial solutions and the respective comparisons are scheduled. On the contrary, if all the n comparisons led to no improvements, the hypervolume is halved and the search is repeated by sampling n solutions around the elite x_e. This shrinking mechanism is repeated until the hypervolume is smaller than 0.0001 % of the total hypervolume. The pseudo-code displaying the working principles of the stochastic short distance exploration is given in Algorithm 2. The new elite x_e is then passed to the following operator for further improvements. The size of the initial hypervolume and the number of times the hypercube volume is halved were empirically fixed to achieve good performance over the various problems considered in this paper.

Algorithm 2 Stochastic short distance exploration

```
generate a hypercube around x_e with a hypervolume 20% of that of D;
while the hypervolume is bigger than 0.0001% of D do
    for i = 1 : n do
        generate randomly a trial solution x_t within the hypercube;
        if f(x_t) ≤ f(x_e) then
            x_e = x_t;
            centre the hypercube around x_e;
        end if
    end for
    if no elite update occurred then
        halve the hypervolume;
    end if
end while
```

Algorithm 3 Deterministic short distance exploration

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

2.3 Deterministic short distance exploration

This exploration move attempts to fully exploit promising search directions. The meaning of this exploration stage is to perform the descent of promising basins of attraction and possibly finalize the search if the basin of attraction is globally optimal. De facto, the short distance exploration is a simple steepest descent deterministic local search algorithm, with an exploratory move similar to that of Hooke-Jeeves algorithm, see [3], or the first local search algorithm of the multiple trajectory search, see [12]. The short distance exploration stage requires an additional memory slot, which will be referred to as x_s (s stands for short). Starting from the elite x_e, this local search, explores each coordinate i (each gene) and samples $x_s[i] = x_e[i] - \rho$, where ρ is the exploratory radius. Subsequently, if x_s outperforms x_e, the trial solution x_t is updated (it takes the value of x_s), otherwise a half step in the opposite direction $x_s[i] = x_e[i] + \frac{\rho}{2}$ is performed. Again, x_s replaces x_t if it outperforms x_e. If there is no update, i.e. the exploration was unsuccessful, the radius ρ

is halved. This exploration is repeated for all the design variables and stopped when a prefixed budget (equal to 150 iterations as suggested in [12]) is exceeded. The pseudo-code displaying the working principles of the deterministic short distance exploration is given in Algorithm 3.

After the application of the deterministic short distance exploration, if there is an improvement in the quality of the solution, the stochastic short 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.

As a remark, a toroidal management of the bounds has been implemented for the three operators above. This means that if, along the dimension i, the design variable $x[i]$ exceeds the bounds by a value of ζ, it is reinserted from the other end of the interval at a distance of ζ from the edge, i.e. given an interval $[a, b]$, if $x[i] = b + \zeta$ it takes the value of $a + \zeta$. Finally, if we consider that each operator (meme) processes an elite x_e and returns, as an output, a fitness-wise improved elite solution, the operator can be said to "succeed" if the output is different from the input (and obviously better than it) and can be said to "fail" otherwise. In this light, S-3SOME functioning is represented by the scheme composed of interacting memes reported in Fig. 1. With the words, "Long", "Stochastic short", and "Deterministic short" the three above mentioned operators are represented. The arrows represent the interaction amongst the memes. The "S" and "F", represent success and failure, respectively, of the meme application.

Figure 1. Functioning scheme of S-3SOME

3. Numerical Results

In order to test the potential of S-3SOME, numerical experiments have been performed on the testbed in [6] (24 problems) in 10, 40, and 100 dimensions. The proposed S-3SOME has been compared with the 3SOME algorithm proposed in [4]. Both 3SOME and S-3SOME have the same parameter values for the two common components, i.e. $\alpha_e = 0.05$ and ρ equal to 40% of the total decision space width. With reference to 3SOME, it can be mentioned that middle distance exploration samples $4n$ points at each activation, as suggested in the original paper. In addition, S-3SOME has been compared with the following modern algorithms. 1) Covariance Matrix Adaptive Evolution Strategy with increasing population size (G-CMA-ES) proposed in [1] with initial population $\lambda_{start} = 10$ and factor for increasing the population size equal to 2. All the other parameters are set to standard values. 2) Self-

Adaptive Differential Evolution (SADE) proposed in [11]. SADE has been run with Learning Period $LP = 20$ and population size $N_p = 50$. The other cconstant values are the same reported in the formulas of the original paper. For each algorithm and test problem, 30 runs have been performed. Each run has been continued for $5000n$ fitness evaluations. Numerical results containing average final value and standard deviations are reported in Tables 1, 2, and 3, respectively. The best results are highlighted in bold face. In order to strengthen the statistical significance of the results, the Wilcoxon Rank-Sum test has also been applied according to the description given in [14], where the confidence level has been fixed at 0.95. The null-hypothesis being that the results of both algorithms come from the same statistical distribution, the rejection of the null-hypothesis is indicated with the symbol "+"("-") – the reference algorithm thus performs better (worse) than the algorithm labeled on the top of the column – while the symbol "=" indicates that the null-hypothesis cannot be rejected and that both algorithms have a statistically equivalent performance.

Numerical results show that the S-3SOME scheme appears to be, on a regular basis, more promising than the 3SOME scheme. According to our interpretation the shrinking mechanism is leading to an efficient algorithmic behavior. More specifically, S-3SOME, in a memetic fashion, contains two local search components that compete and cooperate tackling the problems from different perspectives. While the deterministic search is very efficient for separable problems or for non highly multivariate landscapes, the stochastic search attempts to improve upon the solutions regardless of the separability or multi-modality of problem. The combination of these two simple mechanism with a stochastic global search leads to the generation of a very efficient algorithm capable to compete with modern complex optimization algorithms. For example, it can be observed that S-3SOME tends to have, for all the dimensionality levels under examination, a similar performance with respect to the G-CMA-ES. Regarding the comparison with SADE, an interesting observation can be done: while for low dimensionality levels SADE seems to be extremely competitive, in 100 dimensions its performance tends to deteriorate and is outperformed by S-3SOME. This fact can be explained that while SADE is a complex and efficient structure designed for addressing problems with a low dimensionality (tuned for up to 30 dimensions), the proposed approach lead to simple (which employ limited resources and can be encoded in a few lines) and flexible algorithms having a respectable performance for a various set of optimization problems.

Table 1. Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = S-3SOME) for 10D case

	S-3SOME	*GCMAES*		SADE		*3SOME*	
f1	$\mathbf{7.95e+01 \pm 0.00e+00}$	$\mathbf{7.95e+01 \pm 0.00e+00}$	=	$\mathbf{7.95e+01 \pm 0.00e+00}$	=	$\mathbf{7.95e+01 \pm 0.00e+00}$	=
f2	$\mathbf{-2.10e+02 \pm 0.00e+00}$	$1.37e+03 \pm 1.16e+03$	+	$\mathbf{-2.10e+02 \pm 0.00e+00}$	=	$\mathbf{-2.10e+02 \pm 0.00e+00}$	=
f3	$-4.60e+02 \pm 1.28e+00$	$-5.32e+01 \pm 8.06e+01$	+	$-4.60e+02 \pm 2.66e+00$	=	$\mathbf{-4.61e+02 \pm 1.05e+00}$	=
f4	$-4.59e+02 \pm 1.70e+00$	$-1.59e+01 \pm 8.88e+01$	+	$-4.59e+02 \pm 2.61e+00$	=	$\mathbf{-4.60e+02 \pm 9.64e-01}$	-
f5	$5.14e+00 \pm 2.87e+01$	$1.11e+01 \pm 8.59e+00$	+	$\mathbf{-9.21e+00 \pm 0.00e+00}$	=	$2.52e+01 \pm 3.69e+01$	+
f6	$\mathbf{3.59e+01 \pm 0.00e+00}$	$\mathbf{3.59e+01 \pm 0.00e+00}$	=	$\mathbf{3.59e+01 \pm 0.00e+00}$	=	$\mathbf{3.59e+01 \pm 0.00e+00}$	=
f7	$1.06e+02 \pm 1.03e+01$	$\mathbf{9.29e+01 \pm 0.00e+00}$	-	$9.32e+01 \pm 3.52e-01$	-	$1.10e+02 \pm 1.54e+01$	=
f8	$1.49e+02 \pm 1.65e-01$	$\mathbf{1.49e+02 \pm 0.00e+00}$	-	$\mathbf{1.49e+02 \pm 0.00e+00}$	-	$1.49e+02 \pm 1.85e-01$	=
f9	$1.24e+02 \pm 1.20e+00$	$\mathbf{1.24e+02 \pm 0.00e+00}$	-	$\mathbf{1.24e+02 \pm 0.00e+00}$	-	$1.27e+02 \pm 1.31e+01$	=
f10	$5.68e+03 \pm 2.92e+04$	$5.11e+02 \pm 5.64e+02$	=	$\mathbf{-1.51e+01 \pm 4.10e+01}$	-	$2.48e+02 \pm 1.08e+02$	=
f11	$1.65e+02 \pm 2.63e+01$	$\mathbf{7.63e+01 \pm 0.00e+00}$	-	$7.74e+01 \pm 1.40e+00$	-	$1.60e+02 \pm 2.68e+01$	=
f12	$-6.13e+02 \pm 1.78e+01$	$8.90e+03 \pm 1.82e+04$	+	$\mathbf{-6.21e+02 \pm 7.60e-01}$	=	$-6.13e+02 \pm 1.43e+01$	=
f13	$3.88e+01 \pm 1.10e+01$	$\mathbf{3.00e+01 \pm 0.00e+00}$	-	$3.00e+01 \pm 0.00e+00$	-	$4.26e+01 \pm 1.20e+01$	=
f14	$\mathbf{-5.23e+01 \pm 0.00e+00}$	$5.07e+00 \pm 4.97e+01$	+	$\mathbf{-5.23e+01 \pm 0.00e+00}$	=	$\mathbf{-5.23e+01 \pm 0.00e+00}$	=
f15	$1.07e+03 \pm 3.32e+01$	$\mathbf{1.00e+03 \pm 9.74e-01}$	-	$1.01e+03 \pm 4.18e+00$	-	$1.11e+03 \pm 7.04e+01$	+
f16	$7.80e+01 \pm 3.91e+00$	$\mathbf{7.14e+01 \pm 0.00e+00}$	-	$7.23e+01 \pm 7.78e-01$	-	$7.85e+01 \pm 4.42e+00$	=
f17	$-1.41e+01 \pm 1.01e+00$	$1.31e+01 \pm 2.36e+01$	+	$\mathbf{-1.69e+01 \pm 0.00e+00}$	-	$-1.08e+01 \pm 3.74e+00$	+
f18	$-8.25e+00 \pm 4.78e+00$	$6.55e+01 \pm 6.98e+01$	+	$\mathbf{-1.69e+01 \pm 0.00e+00}$	-	$8.59e-01 \pm 1.75e+01$	+
f19	$-1.00e+02 \pm 1.35e+00$	$-5.50e+01 \pm 2.54e+01$	+	$\mathbf{-1.02e+02 \pm 6.26e-01}$	-	$-9.81e+01 \pm 2.03e+00$	+
f20	$-5.46e+02 \pm 2.98e-01$	$2.54e+04 \pm 3.78e+04$	+	$\mathbf{-5.46e+02 \pm 3.03e-01}$	=	$-5.46e+02 \pm 3.40e-01$	=
f21	$4.98e+01 \pm 6.78e+00$	$4.22e+01 \pm 1.52e+00$	-	$\mathbf{4.19e+01 \pm 9.47e-01}$	-	$5.36e+01 \pm 1.15e+01$	=
f22	$-9.89e+02 \pm 1.35e+01$	$-9.17e+02 \pm 3.24e+00$	+	$\mathbf{-9.98e+02 \pm 3.51e-01}$	-	$-9.87e+02 \pm 1.46e+01$	=
f23	$7.98e+00 \pm 4.53e-01$	$\mathbf{6.96e+00 \pm 0.00e+00}$	-	$7.23e+00 \pm 2.91e-01$	-	$7.85e+00 \pm 5.05e-01$	=
f24	$1.68e+02 \pm 1.98e+01$	$\mathbf{1.12e+02 \pm 4.33e+00}$	-	$1.21e+02 \pm 4.43e+00$	-	$1.89e+02 \pm 3.98e+01$	+

Table 2. Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = S-3SOME) for 40D case

	S-3SOME	*GCMAES*		SADE		*3SOME*	
f1	$\mathbf{7.95e+01 \pm 0.00e+00}$	$\mathbf{7.95e+01 \pm 0.00e+00}$	-	$\mathbf{7.95e+01 \pm 0.00e+00}$	-	$\mathbf{7.95e+01 \pm 0.00e+00}$	=
f2	$\mathbf{-2.10e+02 \pm 0.00e+00}$	$3.57e+02 \pm 2.30e+02$	+	$\mathbf{-2.10e+02 \pm 0.00e+00}$	-	$\mathbf{-2.10e+02 \pm 0.00e+00}$	=
f3	$-4.43e+02 \pm 5.15e+00$	$1.08e+02 \pm 8.31e+01$	+	$-4.24e+02 \pm 1.35e+01$	+	$\mathbf{-4.56e+02 \pm 2.97e+00}$	-
f4	$-4.38e+02 \pm 6.99e+00$	$1.29e+02 \pm 7.11e+01$	+	$-4.04e+02 \pm 2.86e+01$	+	$\mathbf{-4.51e+02 \pm 3.42e+00}$	-
f5	$\mathbf{-9.21e+00 \pm 0.00e+00}$	$2.16e+01 \pm 9.77e+00$	+	$\mathbf{-9.21e+00 \pm 0.00e+00}$	+	$\mathbf{-9.21e+00 \pm 0.00e+00}$	+
f6	$\mathbf{3.59e+01 \pm 0.00e+00}$	$\mathbf{3.59e+01 \pm 0.00e+00}$	-	$3.79e+01 \pm 2.92e+00$	+	$\mathbf{3.59e+01 \pm 0.00e+00}$	=
f7	$1.78e+02 \pm 3.02e+01$	$\mathbf{9.48e+01 \pm 3.13e+00}$	-	$1.30e+02 \pm 1.02e+01$	-	$2.15e+02 \pm 7.26e+01$	=
f8	$1.49e+02 \pm 3.59e-01$	$\mathbf{1.49e+02 \pm 0.00e+00}$	-	$1.82e+02 \pm 2.23e+01$	+	$1.50e+02 \pm 7.35e-01$	+
f9	$1.25e+02 \pm 1.60e+00$	$\mathbf{1.24e+02 \pm 0.00e+00}$	-	$1.58e+02 \pm 1.67e+00$	+	$1.25e+02 \pm 2.07e+00$	=
f10	$3.78e+05 \pm 2.03e+06$	$\mathbf{1.91e+02 \pm 1.12e+02}$	-	$6.09e+03 \pm 2.16e+03$	-	$9.73e+02 \pm 4.00e+02$	=
f11	$3.38e+02 \pm 4.95e+01$	$\mathbf{7.63e+01 \pm 0.00e+00}$	-	$1.16e+02 \pm 1.20e+01$	-	$3.67e+02 \pm 5.95e+01$	+
f12	$\mathbf{-6.14e+02 \pm 7.57e+00}$	$2.69e+03 \pm 2.68e+03$	+	$-6.13e+02 \pm 6.60e+00$	=	$-6.10e+02 \pm 9.48e+00$	=
f13	$4.00e+01 \pm 9.14e+00$	$\mathbf{3.00e+01 \pm 0.00e+00}$	-	$3.75e+01 \pm 6.50e+00$	=	$4.37e+01 \pm 1.28e+01$	=
f14	$\mathbf{-5.23e+01 \pm 0.00e+00}$	$5.01e+01 \pm 4.14e+01$	+	$\mathbf{-5.23e+01 \pm 0.00e+00}$	+	$\mathbf{-5.23e+01 \pm 0.00e+00}$	=
f15	$1.41e+03 \pm 1.04e+02$	$\mathbf{1.01e+03 \pm 2.40e+00}$	-	$1.06e+03 \pm 1.52e+01$	-	$1.99e+03 \pm 4.48e+02$	+
f16	$8.36e+01 \pm 5.33e+00$	$\mathbf{7.14e+01 \pm 0.00e+00}$	-	$8.31e+01 \pm 3.72e+00$	=	$8.94e+01 \pm 6.30e+00$	+
f17	$-1.02e+01 \pm 1.31e+00$	$1.47e+01 \pm 1.42e+01$	+	$\mathbf{-1.60e+01 \pm 4.13e-01}$	-	$-5.39e+00 \pm 3.51e+00$	+
f18	$7.02e+00 \pm 4.89e+00$	$2.18e+01 \pm 1.18e+01$	+	$\mathbf{-1.30e+01 \pm 1.25e+00}$	-	$2.67e+01 \pm 1.31e+01$	+
f19	$-9.67e+01 \pm 1.67e+00$	$-5.03e+01 \pm 1.29e+01$	+	$\mathbf{-1.01e+02 \pm 1.27e+00}$	-	$-9.48e+01 \pm 2.97e+00$	+
f20	$-5.46e+02 \pm 1.64e-01$	$3.66e+03 \pm 2.87e+03$	+	$-5.45e+02 \pm 1.98e-01$	+	$\mathbf{-5.46e+02 \pm 1.67e-01}$	=
f21	$5.08e+01 \pm 1.36e+01$	$\mathbf{4.16e+01 \pm 1.04e+00}$	-	$4.25e+01 \pm 2.30e+00$	-	$5.20e+01 \pm 1.71e+01$	=
f22	$-9.86e+02 \pm 1.00e+01$	$-9.14e+02 \pm 6.51e-01$	+	$-9.91e+02 \pm 8.09e+00$	-	$\mathbf{-9.91e+02 \pm 7.57e+00}$	-
f23	$8.19e+00 \pm 5.09e-01$	$\mathbf{7.62e+00 \pm 1.29e+00}$	-	$8.21e+00 \pm 7.58e-01$	=	$8.11e+00 \pm 7.06e-01$	=
f24	$5.45e+02 \pm 8.96e+01$	$\mathbf{1.50e+02 \pm 1.68e+01}$	-	$1.90e+02 \pm 1.21e+01$	-	$8.98e+02 \pm 2.53e+02$	+

Table 3. Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = S-3SOME) for 100D case

	S-3SOME	*GCMAES*		SADE		*3SOME*	
f1	$\mathbf{7.95e+01 \pm 0.00e+00}$	$\mathbf{7.95e+01 \pm 0.00e+00}$	=	$\mathbf{7.95e+01 \pm 0.00e+00}$	=	$\mathbf{7.95e+01 \pm 0.00e+00}$	=
f2	$\mathbf{-2.10e+02 \pm 0.00e+00}$	$2.56e+02 \pm 1.01e+02$	+	$\mathbf{-2.10e+02 \pm 0.00e+00}$	=	$\mathbf{-2.10e+02 \pm 0.00e+00}$	=
f3	$-4.03e+02 \pm 9.30e+00$	$3.77e+02 \pm 1.57e+02$	+	$-2.92e+02 \pm 4.41e+01$	+	$\mathbf{-4.41e+02 \pm 7.57e+00}$	-
f4	$-3.88e+02 \pm 1.28e+01$	$3.83e+02 \pm 1.56e+02$	+	$-1.59e+02 \pm 7.73e+01$	+	$\mathbf{-4.27e+02 \pm 1.20e+01}$	-
f5	$\mathbf{-9.21e+00 \pm 0.00e+00}$	$1.18e+02 \pm 1.82e+01$	+	$\mathbf{-9.21e+00 \pm 0.00e+00}$	+	$1.04e+02 \pm 4.22e+02$	=
f6	$\mathbf{3.59e+01 \pm 0.00e+00}$	$\mathbf{3.59e+01 \pm 0.00e+00}$	-	$1.18e+02 \pm 3.97e+01$	+	$\mathbf{3.59e+01 \pm 0.00e+00}$	=
f7	$3.75e+02 \pm 9.17e+01$	$\mathbf{1.33e+02 \pm 9.67e+00}$	-	$3.63e+02 \pm 7.58e+01$	=	$5.89e+02 \pm 2.63e+02$	+
f8	$1.71e+02 \pm 3.06e+01$	$\mathbf{1.50e+02 \pm 1.59e+00}$	-	$2.79e+02 \pm 4.17e+01$	+	$1.71e+02 \pm 2.61e+01$	=
f9	$1.75e+02 \pm 2.08e+01$	$\mathbf{1.24e+02 \pm 1.13e+00}$	-	$2.31e+02 \pm 2.31e+01$	+	$1.77e+02 \pm 1.12e+01$	=
f10	$3.04e+03 \pm 8.61e+02$	$\mathbf{6.31e+01 \pm 5.68e+01}$	-	$4.57e+04 \pm 1.70e+04$	+	$3.03e+03 \pm 9.61e+02$	=
f11	$6.30e+02 \pm 8.95e+01$	$\mathbf{7.63e+01 \pm 0.00e+00}$	-	$1.79e+02 \pm 2.23e+01$	-	$3.68e+02 \pm 6.38e+01$	-
f12	$\mathbf{-6.19e+02 \pm 3.74e+00}$	$1.23e+03 \pm 1.04e+03$	+	$-6.15e+02 \pm 7.44e+00$	+	$-6.13e+02 \pm 9.46e+00$	+
f13	$3.66e+01 \pm 4.13e+00$	$\mathbf{3.00e+01 \pm 0.00e+00}$	-	$3.16e+01 \pm 2.64e+00$	-	$3.27e+01 \pm 2.88e+00$	-
f14	$\mathbf{-5.23e+01 \pm 0.00e+00}$	$2.59e+01 \pm 1.58e+01$	+	$\mathbf{-5.23e+01 \pm 0.00e+00}$	+	$\mathbf{-5.23e+01 \pm 0.00e+00}$	=
f15	$2.44e+03 \pm 2.77e+02$	$\mathbf{1.03e+03 \pm 1.16e+01}$	-	$1.33e+03 \pm 4.72e+01$	-	$4.49e+03 \pm 6.17e+02$	+
f16	$8.92e+01 \pm 3.32e+00$	$\mathbf{7.15e+01 \pm 0.00e+00}$	-	$9.75e+01 \pm 4.14e+00$	+	$9.33e+01 \pm 5.37e+00$	+
f17	$-8.00e+00 \pm 1.67e+00$	$5.16e+00 \pm 4.19e+00$	+	$\mathbf{-1.39e+01 \pm 4.73e-01}$	-	$6.39e-02 \pm 3.81e+00$	+
f18	$1.57e+01 \pm 5.86e+00$	$2.43e+01 \pm 1.77e+01$	=	$\mathbf{-5.43e+00 \pm 1.90e+00}$	-	$4.63e+01 \pm 1.53e+01$	+
f19	$-9.32e+01 \pm 2.39e+00$	$6.69e+00 \pm 1.57e+01$	+	$\mathbf{-9.98e+01 \pm 4.45e-01}$	-	$-8.99e+01 \pm 4.96e+00$	+
f20	$-5.46e+02 \pm 0.00e+00$	$2.45e+03 \pm 2.25e+03$	+	$-5.45e+02 \pm 1.94e-01$	+	$\mathbf{-5.46e+02 \pm 0.00e+00}$	-
f21	$5.36e+01 \pm 1.44e+01$	$\mathbf{4.30e+01 \pm 1.06e+00}$	-	$4.68e+01 \pm 6.22e+00$	-	$5.24e+01 \pm 1.02e+01$	=
f22	$-9.83e+02 \pm 1.32e+01$	$-9.14e+02 \pm 0.00e+00$	+	$\mathbf{-9.95e+02 \pm 5.84e+00}$	-	$-9.80e+02 \pm 1.46e+01$	=
f23	$8.31e+00 \pm 6.04e-01$	$8.59e+00 \pm 2.06e+00$	=	$9.33e+00 \pm 7.66e-01$	+	$\mathbf{8.27e+00 \pm 4.79e-01}$	=
f24	$1.66e+03 \pm 3.07e+02$	$\mathbf{3.66e+02 \pm 9.61e+01}$	-	$3.74e+02 \pm 3.33e+01$	-	$2.79e+03 \pm 4.45e+02$	+

4. Conclusion

This paper proposes a simple scheme for numerical optimization. The design of the algorithm has been performed in a bottom-up fashion by following the *lex parsimoniae*. The proposed algorithm, namely Shrinking Three Stage Optimal Memetic Exploration (S-3SOME) is composed of a random optimizer, a stochastic local search which progressively shrinks the search radius, and a deterministic local search. Numerical results show that this simple scheme is flexible and performs well on a set of diverse problems and for various dimensionality levels. The comparison with the inspiring algorithm, i.e. 3SOME, shows that S-3SOME enhance upon its performance by converging to similar or better solution. The comparison with modern complex algorithms demonstrates that S-3SOME is highly competitive, especially in high dimensions, despite a much smaller requirement in terms of memory (as it perturbs only one solution) and overhead (as it does not make use of learning components or sophisticated structures).

Acknowledgments

This research is supported by the Academy of Finland, Akatemiatutkija 130600 and Tutkijatohtori 140487.

References

[1] A. Auger and N. Hansen. A restart CMA evolution strategy with increasing population size. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1769–1776, 2005.

[2] 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*, 10(6):646–657, 2006.

[3] R. Hooke and T. A. Jeeves. Direct search solution of numerical and statistical problems. *Journal of the ACM*, 8:212–229, March 1961.

[4] 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*, 188(1):17-43, 2012.

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

[6] N. Hansen, A. Auger, S. Finck, R. Ros, et al. Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2010.

[7] F. Neri, C. Cotta, and P. Moscato. *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*. Springer, 2011.

[8] F. Neri, G. Iacca, and E. Mininno. Disturbed exploitation compact differential evolution for limited memory optimization problems. *Information Sciences*, 181(12):2469–2487, 2011.

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

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

[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*, 13(2):398–417, 2009.

[12] L. Y. Tseng and C. Chen. Multiple trajectory search for large scale global optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3052–3059, 2008.

[13] 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*, 13(2):243–259, 2009.

[14] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

PII

HANDLING NON-SEPARABILITY IN THREE STAGE MEMETIC EXPLORATION

by

I. Poikolainen, F. Caraffini, F. Neri, M. Weber 2012

Proceedings of the fifth international conference on bioinspired optimization methods and their applications, pages 195-205

HANDLING NON-SEPARABILITY IN THREE STAGE OPTIMAL MEMETIC EXPLORATION

Ilpo Poikolainen*, Fabio Caraffini*, Ferrante Neri*,
Matthieu Weber*
* *Department of Mathematical Information Technology*
P.O. Box 35 (Agora), 40014 University of Jyväskylä, Finland
ilpo.poikolainen@jyu.fi, fabio.caraffini@jyu.fi, ferrante.neri@jyu.fi, matthieu.weber@jyu.fi

Abstract Three Stage Optimal Memetic Exploration (3SOME) is a powerful algorithmic framework which progressively perturbs a single solution by means of three distinct operators. This paper proposes the integration of a simple and still powerful component to enrich the 3SOME algorithm. This component, based on a special mutation scheme of Differential Evolution, replaces an existing 3SOME operator and allows 3SOME to efficiently handle non-separable problems. The resulting algorithm, namely Rotationally Invariant Three Stage Differential Evolution (RI3SOME) displays a good performance on a set of various problems. In particular, the proposed component in RI3SOME improves upon 3SOME performance for non-separable problems and without excessively deteriorating the algorithmic performance for separable problems. The comparison with modern optimization algorithms shows that RI3SOME, despite its simplicity and modest use of computational resources, displays a respectable performance.

Keywords: Memetic Computing, Ockham Razor, Computational Intelligence Optimization, Separability

1. Introduction

A function of n independent variables is said to be separable if it can be expressed as a sum of n functions, each of them depending on only one variable. From an optimization viewpoint, these functions are very easy to handle as the optimization problem in n variables can be expressed as n problems in one variable. In other words, a separable function can be efficiently optimized by perturbing separately each variable. Unfortunately, in real world applications it often (if not always) happens that the objective functions are non-separable. Nonetheless, while in

mathematics a function can be hardly classified either as separable or non-separable, the concept of separability in computational intelligence optimization is more fuzzy: a function can be fully separable, moderately separable, moderately non-separable, fully non-separable. Modern testbeds tend to classify test problems according to similar criteria, see [6]. To better understand the practical implications of this remark, a function, even if non-separable, can still be handled by perturbing separately each variable. Even though this approach will not lead to the detection of the optimum it can still spot an area of the decision space characterized by a high performance. Thus, if coupled with some other components, the resulting algorithm can turn out being an efficient optimizer.

The idea of searching optima not along preferential directions but performing search moves which simultaneously involve multiple variables (diagonal) has been studied in various contexts over the last few decades. For example, in 60's Rosenbrock and Powell algorithms, unlike the Hooke-Jeeves algorithm, were tackling optimization problems by including diagonal moves.

In modern times, with the development of computational intelligence optimization, many move operators, e.g. several kinds of recombination in Evolutionary Algorithms (EAs), naturally perform diagonal moves. On the other hand, only in a minor portion of literature the separability is explicitly handled in optimization problems. One famous example is the Covariance Matrix Adaptation (CMA) integrated within Evolution Strategy (ES) frameworks. The first algorithm employing this logic, known as CMAES, see [2]. The general algorithmic idea is that new trial solutions are generated by a distribution which progressively adapts to the fitness landscape and thus performs search moves along the most convenient direction. Other relevant algorithms based on the CMA have been proposed in literature, e.g. [3], [1], and [5].

In [14], the non-separability of the fitness landscape in handled by the employment of structured populations. Some biologically inspired algorithms, despite their efficiency, naturally perform a biased search along specific axes. This situation occurs in Differential Evolution (DE) where the crossover is executed by an inheritance mechanism of variables from a parent to an offspring solution. As a consequence, even when DE can be very efficient for some fitness landscapes, it may dramatically deteriorate its performance after a rotation operation. It must be observed that the rotation operation over a separable function jeopardizes the separability (with respect to the original axes) of the problem making it non-separable. In order to handle these conditions, several corrections to the DE variation operators have been designed. For example, in [11],

a reference rotation procedure is integrated within DE crossover and in [12] a modified DE crossover is introduced by making use of the centroid point. A classical but still efficient way to obtain a rotationally invariant DE is to combine an arithmetic crossover within the mutation scheme. This mutation scheme, namely DE/current-to-rand/1, has been presented in [10].

Recently, in [4], it has been shown that algorithms with a simple structure can be as efficient as complex algorithms (Ockham's Razor) while requiring little memory and having low computational overhead (see [4] for details). Thus, when an algorithm is designed, it is important to build it up with a bottom-up approach by including the minimum amount of components and figuring out about the role of each operator. In accordance with Memetic Computing (MC) fashion, an algorithm is a structure composed of multiple operators which interact and cooperate to tackle various optimization problems, see [8] and [7]. An algorithm composed of three operators and namely Three Stage Optimal Memetic Exploration (3SOME) has also been proposed in [4] as an example for the Ockham's Razor principle in MC. The main drawback of the 3SOME algorithm was that the non-separability was not explicitly addressed and thus the algorithm displayed a not so good performance in some cases. This paper proposes a modified algorithm based on 3SOME structure where a component for explicitly tackling non-separability is included in the algorithmic framework. The remainder of this paper is organized in the following way. Section 2 describes the proposed Rotationally Invariant Three Stage Memetic Exploration (RI3SOME). Section 3 displays the experimental testbed and numerical results related to the proposed RI3SOME with respect to 3SOME and other modern optimization algorithms. Section 4 gives the conclusion of this work.

2. Rotationally Invariant Three Stage Memetic Exploration

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.

At the beginning of the optimization problem one candidate solution is randomly sampled within the decision space D. In analogy with compact optimization, see [9], we will refer to this candidate solution as elite and indicate it with the symbol x_e. In addition to x_e, the algorithm makes use of another memory slot for attempting to detect other solutions. The latter solution, namely trial, is indicated with x_t. In the

following sub-sections the three exploratory stages and their coordination are described.

The proposed algorithm, like the 3SOME structure proposed in [4], is composed of three operators which perturb a single solution, thus exploring the decision space from complementary perspectives. The first operator, namely long distance exploration, is a randomized long distance search where the newly generated solution x_t inherits part of the elite solution x_e by means of the exponential crossover typical of DE, see [9]. This operator is continued until a new solution outperforming the original one is generated. The second operator, namely intermediate diagonal exploration, processes the elite solution and samples a trial solution x_t by means of the DE/current-to-rand/1 mutation. This operation is continued until a given budget is expired. If the final solution outperforms the initial one, a replacement occurs and a new elite is generated. Regardless of the success of the intermediate diagonal exploration, the elite solution x_e is processed by the short distance exploration. This third operator perturbs the variables separately and attempts to quickly and deterministically descend the corresponding basin of attraction. In this way a component for handling non-separable functions supports the first and third operators which are especially efficient for separable and moderately non-separable problems.

2.1 Long distance exploration

This exploration move attempts to detect a new promising solution within the entire decision space. While the elite x_e is retained, at first, a trial solution x_t is generated by randomly sampling a new set of n genes. Subsequently, the DE exponential crossover is applied between x_e and x_t, see [9]. This exploration stage performs the global stochastic search and thus attempts to detect unexplored promising areas of the decision space. 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. This kind of inheritance of some genes appears to be extremely beneficial in terms of performance with respect to a stochastic blind search (which would generate a completely new solution at each step). If the trial solution outperforms the elite, a replacement occurs. A replacement has been set also if the newly generated solution has the same performance of the elite. This is to prevent the search getting trapped in some plateaus of the decision space. The pseudo-code of this component is shown in Algorithm 1. The long distance exploration 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 stochastic short distance exploration is activated.

Algorithm 1 Long distance exploration

```
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 if
end while
if f(x_t) ≤ f(x_e) then
    x_e = x_t
end if
```

2.2 Intermediate diagonal exploration

Around the solution x_e returned by the long distance exploration a hypercube is considered. This hypercube has a volume equal to one fifth of the volume of the entire decision space D and is centred around x_e. Subsequently, three points, x_r, x_s, and x_v, respectively are sampled (from an implementation viewpoint the points are progressively sampled and allocated into the trial solution to occupy only one memory slot). These points are then combined with x_e by means of DE/current-to-rand/1 to generate x_t:

$$x_t = x_e + K\left(x_v - x_e\right) + F'\left(x_r - x_s\right) \tag{1}$$

where K is the combination coefficient, which should be chosen with a uniform random distribution from $[0, 1]$ and $F' = K \cdot F$. The trial solution replaces the elite if it displays a higher performance. This operation is repeated for a given budget.

This component replaces the middle distance search of 3SOME, see [4]. In 3SOME the middle distance exploration samples points in the same hypercube and performs the exponential crossover to increase the exploitation and thus guide the search towards the improvements. This mechanism is equivalent to keep constant some variables and move along the others. In the present paper, we propose the inheritance of x_e to x_t by means of a linear combination and a search along all the directions simultaneously. This component is supposed to tackle, in a simple and computationally inexpensive way, non-separable problems. Trial solutions can be generated outside the hypercube. The pseudo-code displaying the working principles of the short distance exploration is given in Algorithm 2.

Algorithm 2 Intermediate diagonal distance exploration

while local budget condition **do**
  generate x_r, x_s, and x_v
  $x_t = x_e + K(x_v - x_e) + F'(x_r - x_s)$
end while
if $f(x_t) \leq f(x_e)$ **then**
  $x_e = x_t$
end if

2.3 Short distance exploration

This exploration move attempts to fully exploit promising search directions. The meaning of this exploration stage is to perform the descent of promising basins of attraction and possibly finalize the search if the basin of attraction is globally optimal. De facto, the short distance exploration is a simple steepest descent deterministic local search algorithm, with an exploratory move similar to that of Hooke-Jeeves algorithm, or the first local search algorithm of the multiple trajectory search, see [13]. The short distance exploration stage requires an additional memory slot, which will be referred to as x_s (s stands for short). Starting from the elite x_e, this local search, explores each coordinate i (each gene) and samples $x_s[i] = x_e[i] - \rho$, where ρ is the exploratory radius. Subsequently, if x_s outperforms x_e, the trial solution x_t is updated (it takes the value of x_s), otherwise a half step in the opposite direction $x_s[i] = x_e[i] + \frac{\rho}{2}$ is performed. Again, x_s replaces x_t if it outperforms x_e. If there is no update, i.e. the exploration was unsuccessful, the radius ρ is halved. This exploration is repeated for all the design variables and stopped when a prefixed budget (equal to 150 iterations as suggested in [13]) is exceeded. The pseudo-code displaying the working principles of the short distance exploration is given in Algorithm 3.

After the application of the deterministic short distance exploration, if there is an improvement in the quality of the solution, the stochastic short 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.

As a remark, a toroidal management of the bounds has been implemented for the three operators above. This means that if, along the dimension i, the design variable $x[i]$ exceeds the bounds of a value ζ, it is reinserted from the other end of the interval at a distance ζ from the edge, i.e. given an interval $[a, b]$, if $x[i] = b + \zeta$ it takes the value of $a + \zeta$.

3. Numerical Results

The proposed RI3SOME has been tested on the testbed in [6] (24 problems) in 10, 40, and 100 dimensions. It must be noted that func-

Algorithm 3 Short distance exploration

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

tions f_1 to f_5 of that testbed are separable, while the remaining ones are not. In order to perform a fair comparison 3SOME and RI3SOME have been run with the same parameters, $\alpha_e = 0.05$ and ρ equal to 40% of the total decision space width. Both the budgets, for middle length and intermediate diagonal explorations, have been fixed equal to $4n$ points at each activation. Regarding the DE/current-to-rand/1 mutation in RI3SOME, the scale factor F has been set equal to 0.75 while K is a random number between 0 and 1. For an extensive discussion on the parameter setting of the 3SOME framework see [4]. In addition, RI3SOME has been compared with DE/current-to-rand/1 and with the same parameter setting of F used for RI3SOME and a population size of 50 individual. Finally, also (1+1)-CMAES proposed in [5] has been included in the comparison. The latter algorithm has been chosen since it is explicitly designed for non-separable problems and, like RI3SOME, is based on a single solution. However, it must be remarked that (1+1)-CMAES and RI3SOME do not require a similar computational overhead (having an algorithmic structure similar to 3SOME, RI3SOME is very similar to the latter with regards to memory requirements and computational overhead). More specifically, while RI3SOME requires only two memory slots (one memory slot for the elite and one for the trial solution), (1+1)-CMAES requires to store a covariance matrix. Thus, the corresponding (1+1)-CMAES memory employment grows quadratically with the dimensionality of the problem. Each algorithm has been run for $5000 \times n$ fitness evaluations for each run. For each problem 100 runs have been performed.

Tables 1, 2, and 3 display the numerical results (in terms of final value and standard deviation) for the test problems considered in this work.

The best results are highlighted in bold face. In order to strengthen the statistical significance of the results, the Wilcoxon Rank-Sum test has also been applied according to the description given in [15], where the confidence level has been fixed at 0.95. The null-hypothesis being that the results of both algorithms come from the same statistical distribution, the symbol "+"("-") indicate that the null-hypothesis is rejected (the reference algorithm thus performs better (worse) than the algorithm labeled on the top of the column) while the symbol "=" indicates that the null-hypothesis cannot be rejected and that both algorithms have a statistically equivalent performance. Note that DE/current-to-rand/1 is outperformed by RI3SOME on at least 22 test problems over dimensions 10, 40 and 100, and outperforms the latter only on $f_1 6$ in 10 dimensions. For this reason and due to lack of space, the results of that algorithm have been left out from the tables.

Table 1. Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = RI3SOME) for 10D case

	RI3SOME	3SOME		(1+1)-CMAES	
f_1	$\mathbf{7.95e+01 \pm 1.24e-14}$	$\mathbf{7.95e+01 \pm 1.14e-14}$	=	$\mathbf{7.95e+01 \pm 0.00e+00}$	=
f_2	$\mathbf{-2.10e+02 \pm 1.50e-14}$	$\mathbf{-2.10e+02 \pm 1.77e-14}$	=	$4.16e+05 \pm 2.91e+06$	+
f_3	$-4.58e+02 \pm 1.68e+00$	$\mathbf{-4.61e+02 \pm 1.28e+00}$	-	$-3.92e+02 \pm 3.25e+01$	+
f_4	$-4.57e+02 \pm 2.35e+00$	$\mathbf{-4.60e+02 \pm 1.39e+00}$	-	$-3.63e+02 \pm 5.42e+01$	+
f_5	$6.45e+00 \pm 2.95e+01$	$\mathbf{5.32e+00 \pm 2.91e+01}$	=	$1.18e+01 \pm 5.27e+01$	+
f_6	$2.26e+02 \pm 8.70e+02$	$3.59e+01 \pm 5.97e-02$	-	$\mathbf{3.59e+01 \pm 0.00e+00}$	-
f_7	$1.03e+02 \pm 5.97e+00$	$1.06e+02 \pm 1.36e+01$	=	$\mathbf{1.03e+02 \pm 6.80e+01}$	-
f_8	$\mathbf{1.49e+02 \pm 1.51e-01}$	$\mathbf{1.49e+02 \pm 1.51e-01}$	=	$1.50e+02 \pm 1.20e+00$	+
f_9	$1.26e+02 \pm 1.00e+01$	$1.25e+02 \pm 1.78e+00$	=	$\mathbf{1.25e+02 \pm 1.70e+00}$	-
f_{10}	$2.16e+02 \pm 1.43e+02$	$3.44e+03 \pm 2.25e+04$	=	$\mathbf{-5.49e+01 \pm 0.00e+00}$	-
f_{11}	$1.54e+02 \pm 2.97e+01$	$1.57e+02 \pm 3.26e+01$	=	$\mathbf{7.63e+01 \pm 0.00e+00}$	-
f_{12}	$-6.05e+02 \pm 2.13e+01$	$-6.13e+02 \pm 1.18e+01$	=	$\mathbf{-6.21e+02 \pm 0.00e+00}$	-
f_{13}	$4.14e+01 \pm 1.14e+01$	$4.36e+01 \pm 1.28e+01$	=	$\mathbf{4.05e+01 \pm 1.08e+01}$	=
f_{14}	$-5.23e+01 \pm 2.48e-05$	$-5.23e+01 \pm 3.09e-05$	=	$\mathbf{-5.24e+01 \pm 0.00e+00}$	=
f_{15}	$1.09e+03 \pm 5.13e+01$	$1.10e+03 \pm 6.32e+01$	=	$\mathbf{1.09e+03 \pm 5.90e+01}$	=
f_{16}	$7.91e+01 \pm 4.58e+00$	$7.94e+01 \pm 4.41e+00$	=	$7.89e+01 \pm 4.57e+00$	=
f_{17}	$-1.13e+01 \pm 3.23e+00$	$-9.15e+00 \pm 8.04e+00$	=	$\mathbf{-1.26e+01 \pm 2.50e+00}$	=
f_{18}	$1.01e+01 \pm 2.23e+01$	$\mathbf{4.21e+00 \pm 2.16e+01}$	-	$6.93e+00 \pm 2.58e+01$	=
f_{19}	$-9.89e+01 \pm 1.92e+00$	$-9.81e+01 \pm 2.95e+00$	=	$\mathbf{-9.97e+01 \pm 1.74e+00}$	=
f_{20}	$\mathbf{-5.46e+02 \pm 3.06e-01}$	$\mathbf{-5.46e+02 \pm 2.36e-01}$	=	$-5.45e+02 \pm 3.56e-01$	+
f_{21}	$5.36e+01 \pm 1.49e+01$	$5.22e+01 \pm 1.27e+01$	=	$\mathbf{4.81e+01 \pm 7.52e+00}$	=
f_{22}	$-9.90e+02 \pm 1.26e+01$	$-9.89e+02 \pm 1.46e+01$	=	$\mathbf{-9.91e+02 \pm 1.07e+01}$	=
f_{23}	$\mathbf{7.80e+00 \pm 4.62e-01}$	$7.88e+00 \pm 5.48e-01$	=	$8.00e+00 \pm 6.60e-01$	=
f_{24}	$1.87e+02 \pm 3.65e+01$	$1.88e+02 \pm 4.30e+01$	=	$\mathbf{1.69e+02 \pm 2.68e+01}$	-

Numerical results show that the proposed RI3SOME algorithm tends to outperform 3SOME for non-separable problems (f_6 and following) while it is sometimes outperformed for separable problems (f_1 to f_5). It must be noted that in the case where the Wilcoxon rank-sum test indicates that RI3SOME is outperformed by 3SOME, the differences in average fitness values between these two algorithms are at least one order of magnitude smaller than the difference between 3SOME and the other two algorithms. It is also shown that RI3SOME appears to be much more promising than 3SOME in relatively high dimensions. This can be interpreted that the coordination of diverse components, the first perturbing the variables separately, the second performing diagonal

Table 2. Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = RI3SOME) for 40D case

	RI3SOME	3SOME		(1+1)-CMAES	
f_1	$7.95e+01 \pm 2.70e-14$	$7.95e+01 \pm 1.98e-14$	=	$\mathbf{7.95e+01 \pm 0.00e+00}$	=
f_2	$\mathbf{-2.10e+02 \pm 3.08e-14}$	$\mathbf{-2.10e+02 \pm 3.00e-14}$	=	$-2.09e+02 \pm 6.24e-01$	+
f_3	$-4.23e+02 \pm 1.02e+01$	$\mathbf{-4.54e+02 \pm 3.52e+00}$	-	$5.45e+01 \pm 1.47e+02$	+
f_4	$-4.17e+02 \pm 1.08e+01$	$\mathbf{-4.50e+02 \pm 4.57e+00}$	-	$4.07e+02 \pm 2.18e+02$	+
f_5	$\mathbf{2.31e+01 \pm 1.28e+02}$	$4.58e+01 \pm 1.66e+02$	+	$6.57e+01 \pm 1.36e+02$	+
f_6	$3.59e+01 \pm 5.75e-07$	$3.59e+01 \pm 5.63e-07$	=	$\mathbf{3.59e+01 \pm 0.00e+00}$	=
f_7	$1.95e+02 \pm 4.14e+01$	$2.11e+02 \pm 6.18e+01$	=	$\mathbf{1.08e+02 \pm 4.77e+00}$	-
f_8	$1.52e+02 \pm 1.17e+01$	$1.55e+02 \pm 2.08e+01$	=	$\mathbf{1.50e+02 \pm 1.65e+00}$	=
f_9	$1.25e+02 \pm 1.79e+00$	$1.25e+02 \pm 1.42e+00$	=	$\mathbf{1.25e+02 \pm 1.70e+00}$	=
f_{10}	$9.04e+02 \pm 3.34e+02$	$3.89e+05 \pm 1.97e+06$	+	$\mathbf{-5.42e+01 \pm 1.03e+00}$	-
f_{11}	$\mathbf{3.59e+02 \pm 7.12e+01}$	$3.83e+02 \pm 6.35e+01$	=	$7.90e+04 \pm 5.52e+05$	+
f_{12}	$-6.12e+02 \pm 8.76e+00$	$-6.10e+02 \pm 8.87e+00$	=	$\mathbf{-6.21e+02 \pm 5.35e-04}$	-
f_{13}	$\mathbf{4.00e+01 \pm 1.07e+01}$	$4.24e+01 \pm 1.25e+01$	=	$4.18e+01 \pm 1.47e+01$	=
f_{14}	$-5.23e+01 \pm 6.56e-05$	$-5.23e+01 \pm 6.67e-05$	=	$\mathbf{-5.23e+01 \pm 7.89e-09}$	=
f_{15}	$1.76e+03 \pm 2.48e+02$	$2.04e+03 \pm 3.78e+02$	+	$\mathbf{1.69e+03 \pm 1.91e+02}$	=
f_{16}	$\mathbf{8.78e+01 \pm 6.29e+00}$	$8.80e+01 \pm 5.35e+00$	=	$9.01e+01 \pm 5.67e+00$	=
f_{17}	$-8.18e+00 \pm 1.53e+00$	$-5.39e+00 \pm 3.67e+00$	+	$\mathbf{-9.70e+00 \pm 1.44e+00}$	-
f_{18}	$1.80e+01 \pm 8.10e+00$	$2.61e+01 \pm 1.39e+01$	+	$\mathbf{1.13e+01 \pm 6.06e+00}$	-
f_{19}	$-9.51e+01 \pm 2.87e+00$	$-9.37e+01 \pm 3.34e+00$	+	$\mathbf{-9.57e+01 \pm 2.33e+00}$	=
f_{20}	$-5.45e+02 \pm 1.65e-01$	$\mathbf{-5.46e+02 \pm 1.28e-01}$	=	$-5.45e+02 \pm 1.89e-01$	=
f_{21}	$4.65e+01 \pm 6.85e+00$	$5.19e+01 \pm 1.57e+01$	=	$\mathbf{4.57e+01 \pm 8.91e+00}$	-
f_{22}	$-9.85e+02 \pm 1.53e+01$	$-9.85e+02 \pm 1.25e+01$	=	$\mathbf{-9.88e+02 \pm 8.00e+00}$	=
f_{23}	$\mathbf{7.89e+00 \pm 4.38e-01}$	$8.04e+00 \pm 5.27e-01$	=	$8.31e+00 \pm 6.33e-01$	+
f_{24}	$6.84e+02 \pm 1.66e+02$	$9.16e+02 \pm 2.89e+02$	+	$\mathbf{6.61e+02 \pm 1.21e+02}$	=

Table 3. Average Fitness ± Standard Deviation and Wilcoxon rank-sum test on the Fitness (reference = RI3SOME) for 100D case

	RI3SOME	3SOME		(1+1)-CMAES	
f_1	$\mathbf{7.95e+01 \pm 3.37e-14}$	$\mathbf{7.95e+01 \pm 3.31e-14}$	=	$\mathbf{7.95e+01 \pm 0.00e+00}$	=
f_2	$\mathbf{-2.10e+02 \pm 5.50e-14}$	$\mathbf{-2.10e+02 \pm 5.93e-14}$	=	$-3.79e+01 \pm 5.19e+01$	+
f_3	$-3.41e+02 \pm 2.11e+01$	$\mathbf{-4.38e+02 \pm 7.95e+00}$	-	$1.49e+03 \pm 3.57e+02$	+
f_4	$-3.17e+02 \pm 2.13e+01$	$\mathbf{-4.26e+02 \pm 7.83e+00}$	-	$2.67e+03 \pm 5.19e+02$	+
f_5	$\mathbf{-9.21e+00 \pm 3.64e-12}$	$2.40e+01 \pm 2.33e+01$	+	$3.52e+02 \pm 5.04e+01$	+
f_6	$\mathbf{3.59e+01 \pm 1.72e-07}$	$\mathbf{3.59e+01 \pm 3.79e-08}$	=	$\mathbf{3.59e+01 \pm 2.01e-07}$	=
f_7	$4.22e+02 \pm 1.49e+02$	$5.77e+02 \pm 2.57e+02$	+	$\mathbf{2.26e+02 \pm 4.24e+01}$	-
f_8	$1.91e+02 \pm 3.99e+01$	$\mathbf{1.85e+02 \pm 3.31e+01}$	=	$1.93e+02 \pm 1.80e+01$	=
f_9	$1.74e+02 \pm 1.01e+01$	$1.74e+02 \pm 1.41e+01$	=	$\mathbf{1.73e+02 \pm 1.94e+01}$	=
f_{10}	$2.80e+03 \pm 6.37e+02$	$2.77e+03 \pm 7.75e+02$	=	$\mathbf{1.62e+02 \pm 6.56e+01}$	-
f_{11}	$5.52e+02 \pm 1.12e+02$	$3.74e+02 \pm 8.53e+01$	-	$\mathbf{7.63e+01 \pm 0.00e+00}$	-
f_{12}	$-6.12e+02 \pm 1.18e+01$	$-6.11e+02 \pm 1.83e+01$	=	$\mathbf{-6.19e+02 \pm 2.27e+00}$	-
f_{13}	$\mathbf{3.33e+01 \pm 4.07e+00}$	$3.37e+01 \pm 5.33e+00$	=	$3.34e+01 \pm 4.91e+00$	=
f_{14}	$\mathbf{-5.23e+01 \pm 5.96e-05}$	$\mathbf{-5.23e+01 \pm 6.17e-05}$	=	$\mathbf{-5.23e+01 \pm 2.74e-07}$	-
f_{15}	$3.98e+03 \pm 6.59e+02$	$4.60e+03 \pm 5.47e+02$	+	$\mathbf{3.52e+03 \pm 4.94e+02}$	-
f_{16}	$\mathbf{9.25e+01 \pm 5.70e+00}$	$9.41e+01 \pm 6.01e+00$	=	$9.95e+01 \pm 4.99e+00$	+
f_{17}	$-3.10e+00 \pm 2.75e+00$	$-2.60e-01 \pm 3.41e+00$	+	$\mathbf{-6.32e+00 \pm 2.27e+00}$	-
f_{18}	$3.49e+01 \pm 1.09e+01$	$4.58e+01 \pm 1.62e+01$	+	$\mathbf{2.20e+01 \pm 7.05e+00}$	-
f_{19}	$\mathbf{-9.23e+01 \pm 2.87e+00}$	$-9.08e+01 \pm 3.25e+00$	+	$-8.99e+01 \pm 3.26e+00$	+
f_{20}	$-5.45e+02 \pm 1.26e-01$	$\mathbf{-5.46e+02 \pm 9.34e-02}$	=	$-5.43e+02 \pm 1.19e-01$	+
f_{21}	$5.32e+01 \pm 1.16e+01$	$5.38e+01 \pm 1.37e+01$	=	$\mathbf{4.82e+01 \pm 6.61e+00}$	-
f_{22}	$\mathbf{-9.84e+02 \pm 1.27e+01}$	$\mathbf{-9.84e+02 \pm 1.20e+01}$	=	$\mathbf{-9.84e+02 \pm 1.16e+01}$	=
f_{23}	$8.20e+00 \pm 4.53e-01$	$\mathbf{8.18e+00 \pm 4.44e-01}$	=	$8.86e+00 \pm 4.80e-01$	+
f_{24}	$2.10e+03 \pm 3.66e+02$	$2.71e+03 \pm 5.25e+02$	+	$\mathbf{1.96e+03 \pm 2.41e+02}$	=

movements is beneficial for hard to solve problems. The comparison with (1+1)-CMAES shows that RI3SOME and (1+1)-CMAES display a comparable performance. Since RI3SOME imposes lower computational requirements that (1+1)-CMAES, its implementation is preferable for those applications characterized by a limited hardware, see[4] and [9].

4. Conclusions

This paper proposes the integration, within 3SOME framework, of a rotationally invariant mutation from differential evolution as an al-

gorithmic component to handle non-separability in fitness landscapes. The resulting algorithm is simple, processes only one solution, and is characterized by modest hardware requirements. These features make RI3SOME suitable for embedded implementations. Numerical results on a set composed of diverse optimization problems shown that RI3SOME outperforms in the majority of the cases the original 3SOME structure. This effect is particularly evident for the high dimensional problems taken into account in this work. Finally, RI3SOME appears to be a competitive option also with respect to complex modern algorithms.

Acknowledgments

This research is supported by the Academy of Finland, Akatemiatutkija 130600.

References

[1] 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*, 11(1):1–18, 2003.

[2] 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*, pages 312–317, 1996.

[3] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195, 2001.

[4] 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*, 188:17–43, 2012.

[5] 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*, pages 453–460. ACM Press, 2006.

[6] N. Hansen, A. Auger, S. Finck, R. Ros, et al. Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions. Technical Report RR-6829, INRIA, 2010.

[7] F. Neri and C. Cotta. Memetic algorithms and memetic computing optimization: A literature review. *Swarm and Evolutionary Computation*, 2:1–14, 2012.

[8] F. Neri, C. Cotta, and P. Moscato. *Handbook of Memetic Algorithms*, volume 379 of *Studies in Computational Intelligence*. Springer, 2012.

[9] F. Neri, G. Iacca, and E. Mininno. Disturbed exploitation compact differential evolution for limited memory optimization problems. *Information Sciences*, 181(12):2469–2487, 2011.

[10] K. Price. An introduction to differential evolution. In David Corne, Marco Dorigo, Fred Glover, Dipankar Dasgupta, Pablo Moscato, Riccardo Poli, and Kenneth V. Price, editors, *New Ideas in Optimization*, pages 79–108. McGraw-Hill, 1999.

[11] T. Takahama and S. Sakai. Solving nonlinear optimization problems by differential evolution with a rotation-invariant crossover operation using gram-schmidt process. In *Proceedings of the World Congress on Nature and Biologically Inspired Computing*, pages 533–540, 2010.

[12] T. Takahama and S. Sakai. Efficient nonlinear optimization by differential evolution with a rotation-invariant local sampling operation. In *Proceedings of IEEE Congress on Evolutionary Computation*, pages 2215–2222, 2011.

[13] L. Y. Tseng and C. Chen. Multiple trajectory search for large scale global optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3052–3059, 2008.

[14] D. Whitley, S. Rana, and R. B. Heckendorn. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology*, 7:33–47, 1998.

[15] F. Wilcoxon. Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83, 1945.

PIII

META-LAMARCKIAN LEARNING IN THREE STAGE OPTIMAL MEMETIC EXPLORATION

by

F. Neri, M. Weber, F. Caraffini, I. Poikolainen 2012

12th UK Workshop on Computational Intelligence (UKCI), pages 1-8

Meta-Lamarckian Learning in Three Stage Optimal Memetic Exploration

Ferrante Neri*†, Matthieu Weber*, Fabio Caraffini*†, and Ilpo Poikolainen*
*Centre for Computational Intelligence, School of Computer Science and Informatics,
De Montfort University, The Gateway, Leicester LE1 9BH, England, United Kingdom
Email: {fneri, fcaraffini}@dmu.ac.uk
†Department of Mathematical Information Technology, University of Jyväskylä
P.O. Box 35 (Agora), 40014 Jyväskylä, Finland
Email: {ferrante.neri, matthieu.weber, fabio.caraffini, ilpo.poikolainen}@jyu.fi

***Abstract*—Three Stage Optimal Memetic Exploration (3SOME) is a single-solution optimization algorithm where the coordinated action of three distinct operators progressively perturb the solution in order to progress towards the problem's optimum. In the fashion of Memetic Computing, 3SOME is designed as an organized structure where the three operators interact by means of a success/failure logic. This simple sequential structure is an initial example of Memetic Computing approach generated by means of a bottom-up logic. This paper compares the 3SOME structure with a popular adaptive technique for Memetic Algorithms, namely Meta-Lamarckian learning. The resulting algorithm, Meta-Lamarckian Three Stage Optimal Memetic Exploration (ML3SOME) is thus composed of the same three 3SOME operators but makes use a different coordination logic. Numerical results show that the adaptive technique is overall efficient also in this Memetic Computing context. However, while ML3SOME appears to be clearly better than 3SOME for low dimensionality values, its performance appears to suffer from the curse of dimensionality more than that of the original 3SOME structure.**

***Index Terms*—Memetic Computing, Ockham Razor, Computational Intelligence Optimization, Automatic Algorithmic Design, Meta-Lamarckian Learning**

I. Introduction

In the past few years, the notion of Memetic Algorithm (MA) for solving optimization problems, introduced in [1], has evolved into a more general framework named Memetic Computing (MC), see e.g., [2], [3], and [4]. According to its original definition, a MA is defined as the fusion of one or more local search algorithms within an evolutionary framework, the former being activated within the generation cycle of the latter. When more than one local search algorithm are employed, the designer of the algorithm faces the problem of deciding the manner in which these algorithmic modules (referred to as memes) can be coordinated in order to improve the global performance of the algorithm; these research problems are at the core of the study of MAs. The success and diffusion of MAs is to be searched within their flexibility. The No Free Lunch Theorem [5] proves that there is no universally suitable optimization algorithm and that each optimization problem is a separate story which must be addressed by a specific algorithmic instrument. Since MAs (and MC approaches) are naturally designed each time by selecting their components, they appeared a valid alternative to tackle specific applications, see e.g. [6]. If the concept of algorithmic design is looked from a complementary perspective, most, if not all, optimization algorithms can be considered as a collection of relatively simple modules, the memes, that are in some way coordinated in order to solve optimization problems. In this sense, MC is an umbrella name to identify all the optimization algorithms. Nonetheless, the MC definition is crucially important as it allows to think about optimization algorithms no longer as paradigms but as structured collections of operators. For a given problem, the proper selection of the operators and their coordination rule are at the basis of the success of an algorithm.

The topic of algorithmic coordination in MAs has been extensively disccused over the last years. In [7] a classification is given. In [4] the classification of coordination methods has been extended and updated. The following four categories have been identified: 1) Adaptive Hyper-heuristic, where heuristic rules are employed (e.g., [8], [9], [10], [11]); 2) Meta-Lamarckian learning defined in [12], where the activation of the memes depends on their success, see also [13], [14], [15]; 3) Self-Adaptive and Co-Evolutionary, where the rules coordinating the memes are evolving in parallel with the candidate solutions of the optimization problem or encoded within the solution, see [16], [17], [18], [19]; and 4) Fitness Diversity-Adaptive, where the activation of the memes depends on a measure of the diversity (e.g., [20], [21], [6], [22], [23]). As a general idea, the algorithmic designer attempts to have a system which performs the coordination automatically. The algorithm is supposed to decide itself during runtime the manner in which the different memes are applied, adapting itself to the problem at hand and thus leading to a preliminary form of automatic design of optimization algorithms.

In this paper, we study the effect of employing a Meta-Lamarckian learning approach to coordinate the three operators composing the Three Stage Optimal Memetic Exploration (3SOME) algorithm originally presented in [24]. The 3SOME algorithm, as a choice of the authors, employs a minimalistic coordination scheme simply based on the success of each operator. The 3SOME coordination scheme constitutes the structure of the algorithm. In the present work we attempt

to study the dependency of the algorithmic performance on the coordination of the operators. More specifically, the same 3SOME operators are here tested without the 3SOME structure but by means of the Meta-Lamarckian learning coordination, thus generating the Meta-Lamarckian 3SOME (ML3SOME). The selection of this simple coordination scheme instead of modern relatively complex adaptive systems for parameter setting and component coordination, see [25], [26], [27], has been carried out as a consequence of the Oackham's Razor principle in MC formulated in [24]. It is fundamental to avoid unnecessary complexity while the algorithmic design is performed. MC structures should be constructed, in a bottom-up logic, by progressively adding complexity until the optimization aim is achieved.

The remainder of this paper is organized in the following way. Section II describes the three operators composing 3SOME, while Section III describes in details the two coordination schemes. Section IV displays the experimental test bed and numerical results produced by the two algorithms studied in this paper. Finally, Section V gives the conclusion of this work.

II. Operators of the Three Stage Optimal Memetic Exploration

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.

At the beginning of the optimization problem one candidate solution is randomly sampled within the decision space D. In analogy with compact optimization, see e;g; [28] and [29], we will refer to this candidate solution as elite and indicate it with the symbol x_e. In addition to x_e, the algorithm makes use of another memory slot for attempting to detect other solutions. The latter solution, namely trial, is indicated with x_t.

The following subsections describe the working principle of each operator composing the 3SOME algorithm and the other two variants proposed in this paper. These three operators (memes) are named long-distance, middle-distance, and short-distance exploration, respectively. Further details about the implementation of each operator are available in [24].

A. Long-distance exploration

The purpose of the long-distance operator is to explore the entire decision space and detect a new promising solution. While the elite x_e is retained, at first, a trial solution x_t is generated by randomly sampling a new set of n genes. Subsequently, the DE exponential crossover is applied between x_e and x_t, see [29]. If the trial solution outperforms the elite, a replacement occurs. A replacement has been set also if the newly generated solution has the same performance as the elite, to prevent the search getting trapped in some plateaus of the decision space. This exploration stage performs a global stochastic search and thus attempts to detect unexplored promising areas of the decision space. While this search mechanism extensively explores the decision space, the employed crossover method also promotes retention of a small section of the elite within the trial solution. This kind of inheritance of some genes appears to be extremely beneficial in terms of performance with respect to a stochastic blind search (which would generate a completely new solution at each step). The pseudo-code of this component is shown in Algorithm 1. The long-distance exploration is repeated until it detects a solution that outperforms the original elite.

Algorithm 1 Long-distance exploration

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

B. Middle-distance exploration

The middle-distance exploration operator attempts to focus the search started by the long-distance exploration in order to exploit the detected search directions. At first a a hypercube of side δ, centred around the solution x_e, is constructed. The middle-distance exploration performs the search within the hyper-volume contained in this hyper-cube of side δ. Subsequently, for $4n$ times (n is the dimensionality), a trial point x_t is generated within the hypercube. The trial point x_t is generated from the elite x_e by performing random sampling within the hyper-cube at first and then an exponential crossover between x_e and the randomly generated point. The fitness of this newly generated point is then compared with the fitness of the elite. If the new point outperforms the elite (or has the same performance), x_e is replaced by the new point, otherwise no replacement occurs. The pseudo-code displaying the working principles of this operator is given in Algorithm 2.

Algorithm 2 Middle-distance exploration

```
construct a hyper-cube with side width δ around x_e
for j = 1 : 4n do
    generate a random solution x_t within the hyper-cube
    generate i = round(n · rand(0, 1))
    count = 1
    x_t[i] = x_e[i]
    while rand(0, 1) ≤ Cr AND count < n do
        x_t[i] = x_e[i]
        i = i + 1
        if i == n then
            i = 1
        end if
        count = count + 1
    end while
    if f(x_t) ≤ f(x_e) then
        x_e = x_t
    end if
end for
```

C. Short-distance exploration

The short-distance exploration is a deterministic search that perturbs the variables of the elite one by one, behaving as a simple steepest descent deterministic local search algorithm. The perturbation is not symmetrical but is heuristically arranged in order to save budget with respect to an exhaustive exploratory step, see [30]. This exploration move attempts to fully exploit promising search directions by performing the descent of promising basins of attraction and possibly finalize the search if the basin of attraction is globally optimal. The short-distance exploration stage requires an additional memory slot, which will be referred to as x_s (s stands for short). Starting from the elite x_e, this local search, explores each coordinate i (each gene) and samples $x_s[i] = x_e[i] - \rho$, where ρ is the exploratory radius. Subsequently, if x_s outperforms x_e, the trial solution x_t is updated (it takes the value of x_s), otherwise a half step in the opposite direction $x_s[i] = x_e[i] + \frac{\rho}{2}$ is performed. Again, x_s replaces x_t if it outperforms x_e. If there is no update i.e., the exploration was unsuccessful, the radius ρ is halved. This exploration is repeated for all the design variables and stopped when a prefixed budget (equal to 150 iterations) is exceeded. The pseudo-code displaying the working principles of the short-distance exploration is given in Algorithm 3.

I should be noted that short distance exploration employs an asymmetric search step as it explores solutions, along each axis, at a ρ distance in one direction verse and $\frac{\rho}{2}$ in the opposite verse. Although a rigorous theoretical explanation of this algorithmic choice is not yet available, experimentally this logic appeared to be much more efficient than a straightforward symmetric exploration, see [30].

As a remark, a toroidal management of the bounds has been implemented for the three operators above. This means that if, along the dimension i, the design variable $x[i]$ exceeds the bounds of a value ζ, it is reinserted from the other end of the interval at a distance ζ from the edge, i.e. given an interval $[a, b]$, if $x[i] = b + \zeta$ it takes the value of $a + \zeta$.

Algorithm 3 Short-distance exploration

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

III. Coordination of the operators

Let us indicate with L, M, and S, the long-distance, middle-distance, and short-distance exploration respectively. The following subsections describe, at first, the original coordination scheme employed in [24] and then a coordination according to the Meta-Lamarckian learning proposed for the first time in this paper.

A. Original 3SOME memetic structure

In the original 3SOME algorithm, the three operators are coordinated according to a heuristically determined scheme, which is repeated until the termination criterion is met, that is the exhaustion of a budget of fitness evaluations.

The L operator is first applied until it produces a solution that outperforms the elite. This operators has thus the role of exploring the decision space to generate a new promising solution to be further exploit. The M operator is then run repeatedly, until it does not improve anymore upon the elite. This means that this second operator attempts to search within the interesting area of the decision space. If this search leads to an improvement, the research is continued. It must be appreciated that L and M are stopped by diametrically opposite criteria. This is set because while L aims to detect one new basin of attraction or a new promising search direction, M aims to subsequently improve upon the genotype detect by L and exploit the area of interest as much as possible. This explains why L is interrupted when the search succeeded (possibly after numerous failures) and M is interrupted when the exploitation turns out to be unsuccessful.

Finally, S further refines the work performed by M by performing a steepest descent deterministic search to fully exploit the basin of attraction. As a further consideration, S performs a narrow search and is a pretty computationally expensive. Thus, it is used only when the basin of attraction seems promising indeed and when M is no longer capable to perform improvements. If S detects new promising solutions, the exploitation of the area is continued by activating M again (and then S again). If S fails at detecting a new elite solution, the area is likely fully exploited and there is no use in continuing the local search within it's neighbourhood. For this reason, if S fails, L is activated anew to attempt the exploration in other areas of the decision space.

The description of the working principles of the 3 SOME structure is given Algorithm 4.

B. Meta-Lamarckian learning

Meta-Lamarckian learning is a sophisticated and efficient adaptive scheme proposed in [12] in the context of MAs. This adaptive scheme organizes the operators composing the algorithm (originally the local search components) within a pool. A selection probability is associated to each operator. The selection probability of each operator depends on its performance history during the previous activations. More specifically, the performance $\eta_p(t)$ of the operator p at iteration

Algorithm 4 3SOME structure (coordination of the operators)

```
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 L
    end while
    while x_e is updated do
        apply to x_e the middle-distance exploration M
    end while
    apply to x_e the short-distance exploration S
    if x_e has been updated then
        apply middle-distance exploration M
    else
        apply long-distance exploration L
    end if
end while
```

t is computed as

$$\eta_p(t) = \frac{fe_p^*(t)}{fe_p(t)} \tag{1}$$

where $fe_p(t)$ is the number of fitness evaluations spent by the operator p at iteration t since the algorithm was started, and $fe_p^*(t)$ is the number of fitness evaluations, at iteration t, used by operator p, that have led to an improvement of the elite. In our case, the probability $P_p(t)$ for operator p to be selected at iteration t is thus defined as

$$P_p(t) = \frac{\eta_p(t)}{\eta_L(t) + \eta_M(t) + \eta_S(t)}. \tag{2}$$

The actual choice of the next operator is performed by the mean of a roulette-wheel selection, as described in [12].

However, since the probability for an operator to be selected depends on its past success, each operator must be given a chance to accumulate some amount of success in order for its selection probability to be above zero. The operators therefore undergo at first a training period, during which their probability of being selected does not follow Equation 2, but instead is equal among all three operators. Every time t that an operator has exhausted its allocated budget (or returns, in the case of the long-distance operator), the number of fitness evaluations $fe_p(t)$ used by each of the three operators is checked. The training period thus ends when $\forall p \in \{L, M, S\}\ fe_p(t) > 0$. For the sake of clarity, this coordination scheme is represented as pseudo-code in Algorithm 5.

IV. Numerical results

The performance of the original 3SOME structure has been compared with the ML3SOME.

The algorithms in this study have been tested on the test bed defined in [31] (24 problems) in 10, 40, and 100 dimensions and on the testbed defined in [32] (20 problems) in 1000 dimensions. In order to perform a fair comparison, both the algorithms have been run with the same parameters, $\alpha_e = 0.05$, δ and ρ equal to respectively 10 % and 40 % of the total decision space width and the budget for middle length exploration has been fixed equal to $4n$ fitness evaluations at each activation. For an extensive discussion on the parameter setting of the 3SOME framework see [24]. Each algorithm has been allocated a budget of $5000 \times n$ fitness evaluations for each run and for each problem, 100 runs have been performed.

Algorithm 5 Meta-Lamarckian coordination

```
t ← 0
while termination condition is not met do
    generate U ← rand(0, 1)
    if fe_L(t) > 0 and fe_M(t) > 0 and fe_S(t) > 0 then
        if U < P_L(t) then
            apply the long-distance operator
        else if U < P_L(t) + P_M(t) then
            apply the middle-distance operator
        else
            apply the short-distance operator
        end if
    else
        if U < 1/3 then
            apply the long-distance operator
        else if U < 2/3 then
            apply the middle-distance operator
        else
            apply the short-distance operator
        end if
    end if
    update P_L(t), P_M(t) and P_S(t)
    t ← t + 1
end while
```

Tables I, II, III, and IV display the numerical results (in terms of final value and standard deviation) for the test problems considered in this work. The best results are highlighted in bold face. In order to strengthen the statistical significance of the results, the Wilcoxon Rank-Sum test has also been applied according to the description given in [33], where the confidence level has been fixed at 0.95: a "+" symbol indicates the case when ML3SOME outperforms the algorithm it is compared against, "−" indicates that ML3SOME is on the contrary outperformed, and "=" indicates that the two algorithms have indistinguishable performance.

The displayed results extend the finding in [12]. While in [12] the meta-Lamarckian learning was proven to be effective for coordinating multiple local search components within a standard MA framework, the results here presented show that the effectiveness of meta-Lamarckian schemes can be etended to algorithms which do not have a population nor an evolutionary structure. It can be observed that in 10 dimensions ML3SOME clearly outperforms 3SOME in 11 cases while it is outperformed for only 3 problems. Thus, for the testbed proposed in [31] and in 10 dimensions, the coordination of the operators by means of a meta-Lamarckian scheme appears preferable. It must be observed that the testbed in [31] is composed of 24 diverse problems which display various features in terms of multimodality, separability, ill-conditioning etc. In this sense, we can conclude that for low dimensionality values the meta-Lamarckian coordination is a robust and valid option for the meme coordination. A similar consideration can be done for the problems in 40 dimensions.

Numerical results in 100 and 1000 dimensions are much more contrasted. The comparison of the meta-Lamarckian learning with the original 3SOME structure show that, for high-dimensional values the performance of the two scheme, albeit different, is equally good. More specifically, the Wilcoxon test indicates that ML3SOME outperforms 3SOME in roughly half of the test cases, while the opposite is true in the other cases, with a small number of undecided cases.

TABLE I
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST FOR 10-DIMENSION PROBLEMS [31] (THE REFERENCE ALGORITHM IS ML3SOME)

	ML3SOME	3SOME	
f_1	$\mathbf{7.95e+01 \pm 1.22e-14}$	$\mathbf{7.95e+01 \pm 1.21e-14}$	=
f_2	$\mathbf{-2.10e+02 \pm 1.58e-14}$	$\mathbf{-2.10e+02 \pm 1.63e-14}$	=
f_3	$\mathbf{-4.61e+02 \pm 2.77e+00}$	$-4.61e+02 \pm 1.18e+00$	+
f_4	$\mathbf{-4.60e+02 \pm 4.22e+00}$	$-4.60e+02 \pm 1.39e+00$	+
f_5	$\mathbf{-9.21e+00 \pm 5.42e-14}$	$5.33e+00 \pm 2.91e+01$	+
f_6	$\mathbf{3.59e+01 \pm 3.81e-03}$	$8.25e+01 \pm 2.83e+02$	=
f_7	$\mathbf{1.03e+02 \pm 7.31e+00}$	$1.05e+02 \pm 1.23e+01$	=
f_8	$1.49e+02 \pm 1.89e-01$	$\mathbf{1.49e+02 \pm 1.86e-01}$	-
f_9	$\mathbf{1.24e+02 \pm 9.47e-01}$	$1.25e+02 \pm 1.69e+00$	+
f_{10}	$\mathbf{3.13e+02 \pm 1.64e+02}$	$3.95e+03 \pm 2.63e+04$	+
f_{11}	$1.60e+02 \pm 3.21e+01$	$\mathbf{1.57e+02 \pm 3.36e+01}$	=
f_{12}	$-6.02e+02 \pm 2.32e+01$	$\mathbf{-6.12e+02 \pm 1.33e+01}$	-
f_{13}	$\mathbf{4.08e+01 \pm 9.36e+00}$	$4.26e+01 \pm 1.28e+01$	=
f_{14}	$-5.23e+01 \pm 2.41e-05$	$\mathbf{-5.23e+01 \pm 3.05e-05}$	-
f_{15}	$\mathbf{1.07e+03 \pm 4.10e+01}$	$1.10e+03 \pm 6.38e+01$	+
f_{16}	$\mathbf{7.83e+01 \pm 4.25e+00}$	$7.97e+01 \pm 4.63e+00$	+
f_{17}	$\mathbf{-1.31e+01 \pm 2.74e+00}$	$-1.03e+01 \pm 6.57e+00$	+
f_{18}	$\mathbf{-3.60e+00 \pm 1.06e+01}$	$5.80e+00 \pm 2.56e+01$	+
f_{19}	$\mathbf{-9.93e+01 \pm 1.72e+00}$	$-9.80e+01 \pm 2.98e+00$	+
f_{20}	$-5.46e+02 \pm 2.99e-01$	$\mathbf{-5.46e+02 \pm 2.59e-01}$	=
f_{21}	$\mathbf{5.05e+01 \pm 1.14e+01}$	$5.36e+01 \pm 1.34e+01$	=
f_{22}	$\mathbf{-9.90e+02 \pm 1.33e+01}$	$-9.88e+02 \pm 1.55e+01$	=
f_{23}	$\mathbf{7.80e+00 \pm 4.53e-01}$	$7.86e+00 \pm 4.95e-01$	=
f_{24}	$\mathbf{1.71e+02 \pm 2.80e+01}$	$1.92e+02 \pm 4.46e+01$	+

TABLE II
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST FOR 40-DIMENSION PROBLEMS [31] (THE REFERENCE ALGORITHM IS ML3SOME)

	ML3SOME	3SOME	
f_1	$\mathbf{7.95e+01 \pm 1.96e-14}$	$\mathbf{7.95e+01 \pm 2.56e-14}$	=
f_2	$\mathbf{-2.10e+02 \pm 3.18e-14}$	$\mathbf{-2.10e+02 \pm 3.28e-14}$	=
f_3	$\mathbf{-4.56e+02 \pm 9.98e+00}$	$-4.54e+02 \pm 3.44e+00$	+
f_4	$\mathbf{-4.53e+02 \pm 8.17e+00}$	$-4.51e+02 \pm 4.06e+00$	+
f_5	$\mathbf{-9.21e+00 \pm 8.63e-13}$	$5.63e+01 \pm 1.78e+02$	+
f_6	$3.59e+01 \pm 3.02e-06$	$\mathbf{3.59e+01 \pm 9.31e-07}$	=
f_7	$\mathbf{1.60e+02 \pm 2.50e+01}$	$2.10e+02 \pm 6.39e+01$	+
f_8	$\mathbf{1.50e+02 \pm 8.20e+00}$	$1.53e+02 \pm 1.69e+01$	=
f_9	$1.26e+02 \pm 7.77e+00$	$\mathbf{1.25e+02 \pm 1.53e+00}$	-
f_{10}	$\mathbf{1.00e+03 \pm 3.53e+02}$	$1.95e+05 \pm 1.40e+06$	=
f_{11}	$4.20e+02 \pm 7.64e+01$	$\mathbf{3.80e+02 \pm 6.30e+01}$	-
f_{12}	$\mathbf{-6.16e+02 \pm 6.25e+00}$	$-6.11e+02 \pm 8.98e+00$	+
f_{13}	$4.37e+01 \pm 1.25e+01$	$\mathbf{4.19e+01 \pm 1.28e+01}$	=
f_{14}	$-5.23e+01 \pm 5.44e-05$	$\mathbf{-5.23e+01 \pm 7.18e-05}$	-
f_{15}	$\mathbf{1.40e+03 \pm 1.71e+02}$	$2.06e+03 \pm 4.04e+02$	+
f_{16}	$\mathbf{8.63e+01 \pm 5.03e+00}$	$8.87e+01 \pm 5.44e+00$	+
f_{17}	$\mathbf{-9.70e+00 \pm 2.00e+00}$	$-5.52e+00 \pm 3.25e+00$	+
f_{18}	$\mathbf{1.13e+01 \pm 8.67e+00}$	$2.56e+01 \pm 1.47e+01$	+
f_{19}	$\mathbf{-9.62e+01 \pm 2.43e+00}$	$-9.33e+01 \pm 3.68e+00$	+
f_{20}	$-5.45e+02 \pm 1.98e-01$	$\mathbf{-5.46e+02 \pm 1.28e-01}$	-
f_{21}	$\mathbf{5.06e+01 \pm 1.47e+01}$	$5.28e+01 \pm 1.62e+01$	=
f_{22}	$\mathbf{-9.87e+02 \pm 1.12e+01}$	$-9.85e+02 \pm 1.31e+01$	=
f_{23}	$\mathbf{8.06e+00 \pm 5.71e-01}$	$8.10e+00 \pm 5.26e-01$	=
f_{24}	$\mathbf{6.06e+02 \pm 1.98e+02}$	$9.44e+02 \pm 2.79e+02$	+

Despite the fact that ML3SOME and 3SOME appear to consistently outperform each other on a subset of the test problems across the number of dimensions, the interpretation of the results is not trivial. In 100 dimensions, the original 3SOME structure appears to offer a slightly better performance than the meta-Lamarckian scheme for separable, weakly ill-conditioned, and uni-modal problems. This tendency has anyway some exceptions such as linear slope and step ellipsoidal functions (f_5 and f_7) respectively. For these two problems ML3SOME achieves a better result with an important margin. It is relevant to observe that the meta-Lamarckian learning appears to be regularly more efficient than the 3SOME structure for all the multi-modal functions with adequate global structure ($f_{15} - f_{19}$). Regarding the multi-modal functions with weak global structure, ML3SOME and 3SOME appear to be equally good. In 1000 variables, ML3SOME outperforms 3SOME in half of the problems and is outperformed in most of the other cases. It can be observed that when 3SOME displays a better performance than ML3SOME, the difference in terms of final fitness value is usually small with repect to the total decay (see Fig.s 1 and 5) while in those problems where ML3SOME outperforms 3SOME the margin of difference in the fitness values is remarkably wide (see Fig.s 2, 3, and 4). Although the relevance of the outperformance margin width

TABLE III
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST FOR 100-DIMENSION PROBLEMS [31] (THE REFERENCE ALGORITHM IS ML3SOME)

	ML3SOME	3SOME	
f_1	$\mathbf{7.95e+01 \pm 3.75e-14}$	$\mathbf{7.95e+01 \pm 3.29e-14}$	=
f_2	$\mathbf{-2.10e+02 \pm 5.43e-14}$	$\mathbf{-2.10e+02 \pm 5.69e-14}$	=
f_3	$-4.22e+02 \pm 2.49e+01$	$\mathbf{-4.39e+02 \pm 7.28e+00}$	-
f_4	$-4.04e+02 \pm 3.73e+01$	$\mathbf{-4.27e+02 \pm 8.70e+00}$	-
f_5	$\mathbf{-9.21e+00 \pm 4.84e-12}$	$7.40e+00 \pm 1.65e+02$	+
f_6	$3.59e+01 \pm 9.81e-08$	$\mathbf{3.59e+01 \pm 8.86e-08}$	=
f_7	$\mathbf{3.67e+02 \pm 8.58e+01}$	$5.97e+02 \pm 2.83e+02$	+
f_8	$\mathbf{1.78e+02 \pm 4.10e+01}$	$1.83e+02 \pm 3.31e+01$	+
f_9	$1.94e+02 \pm 3.86e+01$	$\mathbf{1.76e+02 \pm 1.36e+01}$	-
f_{10}	$3.27e+03 \pm 7.21e+02$	$\mathbf{2.68e+03 \pm 6.96e+02}$	-
f_{11}	$7.97e+02 \pm 1.34e+02$	$\mathbf{3.83e+02 \pm 8.22e+01}$	-
f_{12}	$\mathbf{-6.17e+02 \pm 6.16e+00}$	$-6.09e+02 \pm 1.83e+01$	+
f_{13}	$3.69e+01 \pm 5.04e+00$	$\mathbf{3.35e+01 \pm 4.87e+00}$	-
f_{14}	$-5.23e+01 \pm 5.17e-05$	$\mathbf{-5.23e+01 \pm 5.47e-05}$	-
f_{15}	$\mathbf{2.44e+03 \pm 5.95e+02}$	$4.53e+03 \pm 5.89e+02$	+
f_{16}	$\mathbf{8.97e+01 \pm 4.38e+00}$	$9.51e+01 \pm 6.11e+00$	+
f_{17}	$\mathbf{-6.35e+00 \pm 3.68e+00}$	$-2.63e-02 \pm 3.97e+00$	+
f_{18}	$\mathbf{2.24e+01 \pm 1.32e+01}$	$4.55e+01 \pm 1.54e+01$	+
f_{19}	$\mathbf{-9.31e+01 \pm 2.56e+00}$	$-9.08e+01 \pm 3.39e+00$	+
f_{20}	$-5.45e+02 \pm 1.17e-01$	$\mathbf{-5.46e+02 \pm 9.61e-02}$	-
f_{21}	$\mathbf{5.18e+01 \pm 1.17e+01}$	$5.19e+01 \pm 1.21e+01$	=
f_{22}	$\mathbf{-9.84e+02 \pm 1.36e+01}$	$-9.82e+02 \pm 1.47e+01$	=
f_{23}	$8.25e+00 \pm 4.62e-01$	$\mathbf{8.21e+00 \pm 4.93e-01}$	=
f_{24}	$\mathbf{1.88e+03 \pm 4.62e+02}$	$2.79e+03 \pm 4.75e+02$	+

TABLE IV
AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST FOR 1000-DIMENSION PROBLEMS [32] (THE REFERENCE ALGORITHM IS ML3SOME)

	ML3SOME	3SOME	
f_1	$1.90e-10 \pm 1.40e-10$	$\mathbf{1.33e-11 \pm 3.43e-11}$	-
f_2	$\mathbf{9.92e-06 \pm 1.03e-05}$	$1.07e-04 \pm 1.77e-04$	+
f_3	$\mathbf{6.21e-05 \pm 1.16e-05}$	$5.42e-04 \pm 2.85e-04$	+
f_4	$1.89e+13 \pm 5.33e+12$	$\mathbf{7.14e+12 \pm 2.35e+12}$	-
f_5	$\mathbf{5.07e+08 \pm 1.50e+08}$	$7.06e+08 \pm 1.23e+08$	+
f_6	$\mathbf{1.93e+07 \pm 2.60e+06}$	$1.98e+07 \pm 1.01e+05$	=
f_7	$3.42e+09 \pm 9.45e+08$	$\mathbf{1.00e+09 \pm 2.56e+08}$	-
f_8	$8.73e+08 \pm 2.05e+09$	$\mathbf{3.29e+08 \pm 1.42e+09}$	-
f_9	$2.56e+08 \pm 6.34e+07$	$\mathbf{2.12e+08 \pm 4.13e+07}$	-
f_{10}	$\mathbf{3.47e+03 \pm 2.88e+02}$	$6.80e+03 \pm 3.43e+02$	+
f_{11}	$\mathbf{1.50e+02 \pm 5.12e+01}$	$1.98e+02 \pm 1.94e-01$	+
f_{12}	$\mathbf{5.37e+04 \pm 1.14e+04}$	$5.54e+04 \pm 1.18e+04$	=
f_{13}	$6.63e+03 \pm 4.53e+03$	$\mathbf{4.68e+03 \pm 4.77e+03}$	-
f_{14}	$6.38e+07 \pm 3.16e+06$	$\mathbf{5.62e+07 \pm 5.44e+06}$	-
f_{15}	$\mathbf{7.33e+03 \pm 5.57e+02}$	$1.38e+04 \pm 4.63e+02$	+
f_{16}	$\mathbf{8.67e+01 \pm 5.23e+01}$	$3.81e+02 \pm 6.26e+01$	+
f_{17}	$\mathbf{3.69e+04 \pm 7.04e+03}$	$4.78e+04 \pm 1.78e+04$	+
f_{18}	$\mathbf{1.40e+03 \pm 2.59e+03}$	$1.80e+04 \pm 1.24e+04$	+
f_{19}	$4.39e+05 \pm 6.23e+04$	$\mathbf{8.71e+04 \pm 9.95e+03}$	-
f_{20}	$\mathbf{9.94e+02 \pm 1.86e+02}$	$1.04e+03 \pm 1.63e+02$	+

strictly depends on the features of the fitness landscape, it can be conjectured that this result is due to the meta-Lamarckian logic which tends to select the components that mostly produce fitness enhancements.

Fig.s 1, 2, 3, 4, and 5 show some examples of performance trends.

V. CONCLUSION

This paper compares the performance of the original heuristic scheme for coordinating the operators in the 3SOME algorithm against an algorithm composed of the same operators but where the algorithmic structure is replaced by an adaptive scheme, namely meta-Lamarckian learning.

An extensive set of problems have been setup for this comparison. This set includes very diverse problems in terms of problem dimensionality, multimodality, separability, and ill-conditioning. Numerical results show that the meta-Lamarckian coordination appears to be more efficient than the original heuristic structure for low dimensional problems. On the other hand, the advantages of the adaptive coordination are not too evident in high dimensions. In the latter cases, the two coordination schemes display a different but still almost equally good performance. Nonetheless, it can be observed that the meta-Lamarckian learning is, in some cases, much more efficient than than the heuristic structure. Despite the fact that the two algorithms use the same set of operators, the meta-Lamarckian coordination allows a regular achievement of much better results on multi-modal problems with adequate global structure. Also in other isolated cases, the meta-Lamarckian learning allows the detection of final fitness values a few order of magnitude smaller than those detected

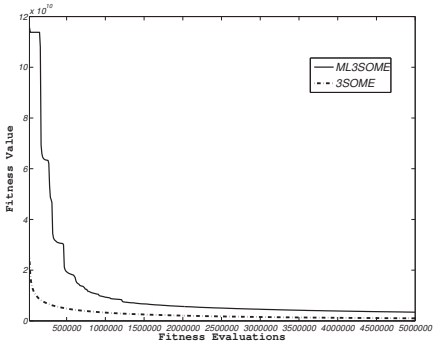


Fig. 1. Performance trends for f_7 from [32] in 1000 dimensions

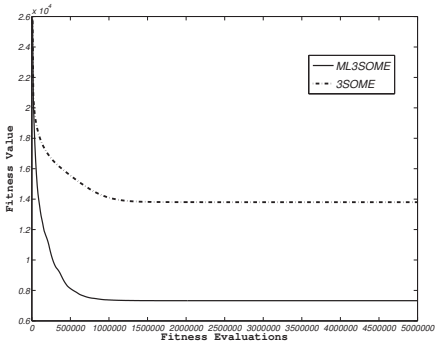


Fig. 3. Performance trends for f_{15} from [32] in 1000 dimensions

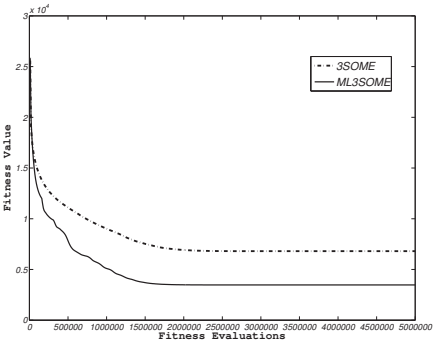


Fig. 2. Performance trends for f_{10} from [32] in 1000 dimensions

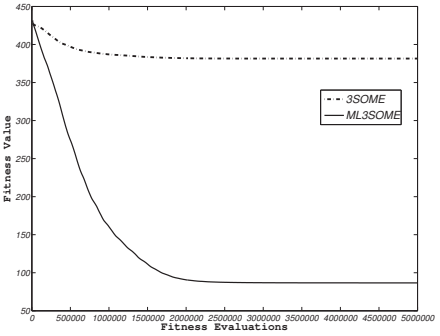


Fig. 4. Performance trends for f_{16} from [32] in 1000 dimensions

by the heuristic scheme. On the other hand, the original algorithm for a limited amount of problems, appears to be capable of detecting slightly better results compared to those detected by ML3SOME. In addition, the 3SOME structure appears, in some cases, very efficient in the early stages of the evolution and capable of quickly finding solutions with a high performance.

This study, although preliminary, has the important role of highlighting the fact that different coordination schemes of the same operators can lead to different results. Future studies focused on the bottom-up algorithmic design will attempt to combine and integrate adaptive coordination schemes within the structure of the algorithms.

ACKNOWLEDGMENTS

This research is supported by the Academy of Finland, Akatemiatutkija 130600.

REFERENCES

[1] P. Moscato, "On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms," Tech. Rep. 826, 1989.

[2] Y.-S. Ong, M.-H. Lim, and X. Chen, "Memetic computation-past, present and future," *IEEE Computational Intelligence Magazine*, vol. 5, no. 2, pp. 24–31, 2010.

[3] F. Neri, C. Cotta, and P. Moscato, *Handbook of Memetic Algorithms*, ser. Studies in Computational Intelligence. Springer, 2012, vol. 379.

[4] F. Neri and C. Cotta, "Memetic algorithms and memetic computing optimization: A literature review," *Swarm and Evolutionary Computation*, vol. 2, pp. 1–14, 2012.

[5] D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[6] F. Neri, J. 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.

[7] Y. S. Ong, M. H. Lim, N. Zhu, and K. W. Wong, "Classification of adaptive memetic algorithms: A comparative study," *IEEE Transactions*

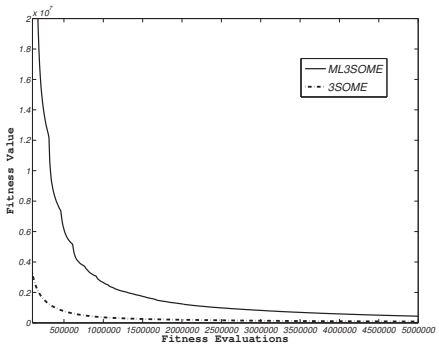


Fig. 5. Performance trends for f_{19} from [32] in 1000 dimensions

On Systems, Man and Cybernetics - Part B, vol. 36, no. 1, pp. 141–152, 2006.

[8] 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.

[9] 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. Lecture Notes in Computer Science. Springer, 2000, vol. 2079, pp. 176–190.

[10] G. Kendall, P. Cowling, and E. Soubeiga, "Choice function and random hyperheuristics," in *Proceedings of the Fourth Asia-Pacific Conference on Simulated Evolution and Learning*, 2002, pp. 667–71.

[11] A. V. Kononova, D. B. Ingham, and M. Pourkashanian, "Simple scheduled memetic algorithm for inverse problems in higher dimensions: Application to chemical kinetics," in *Proceedings of the IEEE World Congress on Computational Intelligence*, 2008, pp. 3906–3913.

[12] 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.

[13] P. Korošec, J. Šilc, and B. Filipič, "The differential ant-stigmergy algorithm," *Information Sciences*, 2011, to appear.

[14] M. N. Le, Y. S. Ong, Y. Jin, and B. Sendhoff, "Lamarckian memetic algorithms: local optimum and connectivity structure analysis," *Memetic Computing Journal*, vol. 1, no. 3, pp. 175–190, 2009.

[15] Q. C. Nguyen, Y. S. Ong, and M. H. Lim, "A probabilistic memetic framework," *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 3, pp. 604–623, 2009.

[16] 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.

[17] 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.

[18] E. L. Yu and P. N. Suganthan, "Ensemble of niching algorithms," *Information Sciences*, vol. 180, no. 15, pp. 2815–2833, 2010.

[19] J. E. Smith, "Estimating meme fitness in adaptive memetic algorithms for combinatorial problems," *Evolutionary Computation*, vol. 20, no. 2, pp. 165–188, 2012.

[20] 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, special issue on Memetic Algorithms*, vol. 37, no. 1, pp. 28–41, 2007.

[21] 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, no. 8, pp. 811–831, 2009.

[22] J. Tang, M. H. Lim, and Y. S. Ong, "Diversity-adaptive parallel memetic algorithm for solving large scale combinatorial optimization problems," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 11, no. 9, pp. 873–888, 2007.

[23] 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, no. 4, pp. 529–555, 2008.

[24] 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.

[25] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "SATzilla: Portfolio-based algorithm selection for SAT," *Journal of Artificial Intelligence Research*, vol. 32, pp. 565–606, 2008.

[26] A. E. Eiben, Z. Michalewicz, M. Schoenauer, and J. E. Smith, "Parameter control in evolutionary algorithms," in *Parameter Setting in Evolutionary Algorithms*, ser. Studies in Computational Intelligence, F. G. Lobo, C. F. Lima, and Z. Michalewicz, Eds. Springer, 2007, vol. 54, pp. 19–46.

[27] H. H. Hoos, "Automated algorithm configuration and parameter tuning," in *Autonomous Search*, Y. Hamadi, E. Monfroy, and F. Saubion, Eds. Springer, 2012, ch. 3, pp. 37–71.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

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

PIV

FOCUSING THE SEARCH: A PROGRESSIVELY SHRINKING MEMETIC COMPUTING FRAMEWORK

by

I. Poikolainen, G. Iacca,F. Caraffini, F. Neri 2013

Int. J. Innovative Computing and Applications, pages 3-16

Focusing the search: a progressively shrinking memetic computing framework

Ilpo Poikolainen*

Department of Mathematical Information Technology,
University of Jyväskylä,
Jyväskylä, Finland
E-mail: ilpo.poikolainen@jyu.fi
*Corresponding author

Giovanni Iacca

INCAS^3 - Innovation Centre for Advanced Sensors and Sensor Systems,
Assen, The Netherlands
E-mail: giovanniiacca@incas3.eu

Fabio Caraffini and Ferrante Neri

Centre for Computational Intelligence,
School of Computer Science and Informatics,
De Montfort University,
Leicester, England, United Kingdom
and
Department of Mathematical Information Technology,
University of Jyväskylä,
Jyväskylä, Finland
E-mail: fcaraffini@dmu.ac.uk and fabio.caraffini@jyu.fi
E-mail: fneri@dmu.ac.uk and ferrante.neri@jyu.fi

Abstract: An extremely natural, yet efficient design pattern in Memetic Computing optimisation is the sequential structure: algorithms composed of few simple memes executed sequentially, each one with its own specific role, have proven to be robust and versatile on various optimisation problems with diverse features and dimensionality values. This principle of non-complexity, which can be seen as an application of the Ockham's Razor in Memetic Computing, lead us to create S-3SOME (Shrinking Three Stage Optimal Memetic Exploration), a scheme which progressively perturbs a candidate solution by alternating three search operators, the first one being a stochastic global search, the second a random sampling within progressive narrowing hyper-volume, and the third a deterministic local search. Numerical results show that the proposed S-3SOME, despite its simplicity, is competitive not only with other memory-saving schemes recently proposed in literature, but also with complex state-of-the-art population-based algorithms characterised by high computational overhead and memory employment.

Keywords: algorithm for resource-constrained hardware; computational intelligence optimisation; memetic computing.

Reference to this paper should be made as follows: Poikolainen, I., Iacca, G., Caraffini, F. and Neri, F., (2013) 'Focusing the search: a progressively shrinking memetic computing framework', *Int. J. Innovative Computing and Applications*, Vol.5, No. 1, pp.3–16.

Biographical notes: Ilpo Poikolainen received his M.Sc. degree from the University of Jyväskylä, Finland, in 2011. He is currently a Ph.D. student at the University of Jyväskylä, Faculty of Information Technology. His research interests include computational intelligence optimisation and more specifically memetic computing, differential evolution, noisy and large scale optimisation, and compact and parallel algorithms.

Giovanni Iacca received his Ph.D. degree in Computer Science from the University of Jyväskylä, Finland, in 2011. He is currently a postdoctoral researcher at INCAS3, Assen, The Netherlands, in the Systems & Controls research group. His research interests include evolutionary optimisation, memetic computing, memory-saving algorithms, robotics, real-time systems, wireless sensor networks and distributed computing. He is a member of IEEE and IEEE CIS.

Fabio Caraffini received his M.Sc. degree in Computer and Electronic Engineering from the University of Perugia, Italy, in 2011. He is currently a Ph.D student in Computer Science at De Montfort University, Leicester, United Kingdom. His research interests include computational intelligence optimisation, robotics, and embedded systems.

Ferrante Neri obtained his first Ph.D in Electro-technical Engineering, from the Technical University of Bari, Italy, in Apr 2007. In Nov 2007, he obtained a second Ph.D in Computer Science from the University of Jyväskylä, Finland. In 2009 he was awarded an Academy Research Fellowship by the Academy of Finland. Dr. Neri is currently an Adjunct Professor in Computational Intelligence at the University of Jyväskylä and Reader in Computational Intelligence at the De Montfort University, United Kingdom. His current research interests include computational intelligence optimisation and more specifically memetic computing, differential evolution, noisy and large scale optimisation, and compact and parallel algorithms.

1 Introduction

Despite the very well-known No Free Lunch Theorems (Wolpert and Macready, 1997), the computational intelligence community every year still tries to propose new robust and versatile algorithms which can be applied to a broad range of optimisation problems with different dimensionalities and features (separability, multi-modality, ill-conditioning, etc.). In order to design such high performance general purpose algorithms, one might consider for example the idea of "combining" multiple algorithmic components, trying to exploiting the advantages of each of them. This concept is at the basis of Memetic Computing (MC), "a broad subject which studies complex and dynamic computing structures composed of interacting modules (memes) whose evolution dynamics is inspired by the diffusion of ideas. Memes are simple strategies whose harmonic coordination allows the solution of various problems" (Neri el al., 2012). Being MC one of the major trends in modern computational intelligence, most of the state-of-the-art algorithms can be considered truly memetic structure in the light of the previous definition. Although several design patterns inspired the creation of new memetic structures in recent years, at least two design approaches can be individuated in modern optimisation:

1 Starting from an existing optimisation algorithm, its structure is "perturbed" by slightly modifying the structure and adding on extra components. Obviously, this approach attempts to obtain a certain performance improvement in correspondence to the proposed modifications. Successful examples of this research approaches are given in (Brest el al., 2006), where a controlled randomisation on Differential Evolution (DE) parameters offers a promising alternative to the standard DE framework, and (Liang el al., 2006), where the variation operator combining the solutions of a population is modified in the context of Particle Swarm Optimisation (PSO).

2 Starting from a set of algorithms, they are combined in a hybrid fashion with the trust that their combination and coordination leads to a flexible structure displaying a better performance than the various algorithms considered separately. Two examples of recently proposed algorithms which are basically the combination, by means of activation probabilities, of various meta-heuristics are given in (Peng el al., 2010) and (Vrugt el al., 2009).

As an alternative to these two approaches, (Iacca el al., 2012) proposed the application of the Ockham's Razor in MC, and more more generally in Computational Intelligence Optimisation. According to the Ockham's Razor, the simplest explanation of natural phenomena is likely to be the closest to the truth. In an analogous way, an optimisation algorithm should be designed in the most natural and simple way, based on the understanding of its components, and following a bottom-up procedure where the role and function of each meme should be clear. Once a given "performance goal" is defined, one should include in the algorithmic structure only the strictly necessary components which guarantee, on one hand, an efficient use of computational resources, and, on the other hand, an efficient handling of different fitness landscapes. On the basis of this idea, in (Iacca el al., 2012), it is shown how a very simple algorithm, namely Three Stage Optimal Memetic Exploration (3SOME), which combines three sequential perturbation mechanisms over a single solution, is competitive

with complex algorithms representing the-state-of-the-art in Computational Intelligence Optimisation, while requiring only little memory and computational resources.

In the attempt of applying the "perturbation" design approach to the aforementioned 3SOME algorithm, while maintaining at the same time its successful simplicity, in this paper we propose a memetic framework in which one of the three original components of 3SOME is replaced with an alternative operator. More specifically, we kept the random global search (which attempts to detect promising search directions) and the deterministic local search (which refines the search in a given basin of attraction perturbing each variable separately), and we replaced the original random local search with a stochastic search operator which progressively focuses the search towards the most interesting areas of the decision space. Since the latter component performs the search by means of a progressive shrinking of the hyper-volume, the proposed approach is named Shrinking 3SOME (S-3SOME).

The remainder of this paper is organised in the following way. Section 2 gives a detailed description of the three operators and the coordination scheme composing the proposed algorithm. Section 3 shows the performance comparison between the proposed algorithm and a set of modern algorithms on two complete benchmarks which include numerous test problems with different dimensionalities and characteristics. An analysis of the algorithmic overhead is also provided. Finally, Section 4 gives the conclusion of this study.

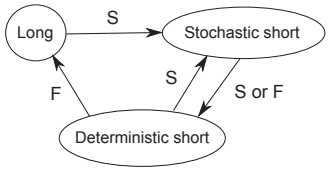
2 Shrinking three stage optimal memetic exploration

In order to clarify the notation used, we refer to the minimisation problem of an objective function $f(x)$, where the candidate solution x is a vector of n design variables (genes) in a decision space D.

In the beginning of the optimisation procedure, one candidate solution is randomly sampled within the decision space D. In analogy with compact optimisation, see (Neri el al., 2011), we will refer to this candidate solution as "elite" and indicate it with the symbol x_e. In addition to x_e, the algorithm makes use of another memory slot for attempting to detect other solutions (we here refer to "memory slot" to indicate an n-dimensional array containing a candidate solution of the optimisation problem). The latter solution, namely trial, is indicated with x_t.

Similar to 3SOME, the S-3SOME algorithm we here propose is composed of three memes, namely the "Long", "Stochastic short", and "Deterministic short" exploration operators (memes) which are activated sequentially and perturb progressively the current elite solution. If we consider that each operator processes the current x_e and returns, as an output, a (possibly) fitness-wise improved solution, the operator can be said to "succeed" if the output is different from the input (and obviously better than it) and can be said to "fail" otherwise. In this light, the S-3SOME functioning is represented by the scheme composed of interacting memes reported in Figure 1. The arrows represent the interaction amongst the memes. The "S" and "F", represent success and failure, respectively, of the meme application.

Figure 1 Functioning scheme of S-3SOME.



In the following subsections, the three aforementioned exploratory stages are described. As a remark, it should be noticed that a toroidal management of the bounds has been implemented for the all the three operators: if, along the dimension i, the perturbed design variable $x[i]$ exceeds the bounds by a value of ζ, it takes a value at a distance of ζ from the other end of the interval, i.e., given an interval $[a, b]$, if $x[i] = b + \zeta$, then $x[i] = a + \zeta$.

2.1 Long distance exploration

The goal of this operator is finding new promising solutions within the entire decision space D. While the elite x_e is retained, at first, a trial solution x_t is randomly sampled in D. Subsequently, the exponential crossover typically used in DE is applied between x_e and x_t, see (Price el al., 2005). More specifically, one gene from x_e is randomly selected. This gene replaces the corresponding gene within the trial solution x_t. Then, a set of random numbers between 0 and 1 is generated. As long as $rand(0, 1) \leq Cr$, where the crossover rate Cr is a predetermined parameter, the design variables from the elite x_e are copied into the corresponding positions of the trial solution x_t. The copy process is interrupted when $rand(0, 1) > Cr$. It can easily be observed that, for a given value of Cr, the meaning of the long distance exploration would change with the dimensionality of the problem. In order to avoid this problem and make the crossover action independent on the dimensionality of the problem, the following quantity, namely inheritance factor, is fixed: $\alpha_e \approx \frac{n_e}{n}$, where n_e is the number of genes we expect to copy from x_e into x_t in addition to the 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 we imposed that $Cr^{n\alpha_e} = 0.5$. 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: $Cr = \frac{1}{\sqrt[n\alpha_e]{2}}$.

From this description, it is clear that the long distance exploration performs a "partially" global stochastic search: while the search mechanism extensively explores the decision space, it also promotes retention of a small section of the elite within the trial solution. This kind of inheritance of some genes appears to be extremely beneficial in terms of performance with respect to a stochastic blind search (which would generate a completely new solution at each step). The long distance exploration is repeated until it does not detect a solution that outperforms the original elite: in this case, a replacement of the current elite occurs. To prevent the search getting trapped in some plateaus of the decision space (regions of the decision space characterised by a null gradient), elite replacements are performed also when the newly generated solutions have the same performance of the elite. Whenever the current elite is updated, the stochastic short distance exploration is activated. The pseudo-code of this component is shown in Algorithm 1.

Algorithm 1 Long distance exploration

```
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 if
end while
if f(x_t) ≤ f(x_e) then
  x_e = x_t
end if
```

2.2 Stochastic short distance exploration

This exploration move attempts to detect promising areas of the decision space by making use of a stochastic logic. In a nutshell, when the long distance exploration detects a new promising solution, the stochastic short distance exploration generates a hypercube centered in the newly detected solution x_e and having a hyper-volume which is 20% of the decision space D. This exploration samples a trial solution x_t for n times (being n the dimensionality of the problem), attempting to outperform the current elite x_e. If $f(x_t) \leq f(x_e)$, the elite solution is updated and the hyper-volume is centered around the new elite solution. If, after n comparisons, at least one replacement occurred, n new attempts are scheduled in the same hyper-volume, and comparisons performed accordingly. On the contrary, if all the n comparisons led to no improvements, the n new samplings are scheduled in a new hyper-volume, obtained halving the current one. This shrinking mechanism is repeated until the hyper-volume is smaller than 0.0001% of the total hyper-volume. When this condition occurs, the current x_e is passed to the following operator for further improvements. The difference between the original implementation in 3SOME is that edge length of the hypercube created is not fixed value but also shrinks over time. This modification allows scaling along the dimensionality of the problem and being more explorative at start and turning into more exploitative search operator towards the end. The size of the initial hyper-volume and the number of times the hypercube volume is halved were empirically fixed to achieve good performance over the various problems considered in this paper. The pseudo-code displaying the working principles of the stochastic short distance exploration is given in Algorithm 2.

Algorithm 2 Stochastic short distance exploration

```
generate a hypercube around x_e with a hyper-volume 20%
of that of D;
while the hyper-volume is bigger than 0.0001% of D do
  for i = 1 : n do
    generate randomly a trial solution x_t within the
    hypercube;
    if f(x_t) ≤ f(x_e) then
      x_e = x_t;
      centre the hypercube around x_e;
    end if
  end for
  if no elite update occurred then
    halve the hyper-volume;
  end if
end while
```

2.3 Deterministic short distance exploration

This operator attempts to fully exploit promising search directions, with the goal of performing the descent of promising basins of attraction, and possibly finalising the search if the basin of attraction is globally optimal. De facto, the short distance exploration is a simple steepest descent deterministic local search algorithm, with an exploratory move similar to that of Hooke-Jeeves algorithm, see (Hooke and Jeeves, 1961), or the first local search algorithm of the multiple trajectory search, see (Tseng and Chen, 2008). The short distance exploration stage requires an additional memory slot, which will be referred to as x_s (s stands for short). Starting from the elite x_e, this local search perturbs each i-th decision variable as $x_s[i] = x_e[i] - \rho$, where ρ is the exploratory radius. Subsequently, if x_s outperforms x_e, the trial solution x_t is updated (the values of x_s are copied in it), otherwise a half step in the opposite direction is performed ($x_s[i] = x_e[i] + \frac{\rho}{2}$). Again, x_s replaces x_t if it outperforms x_e. If there is no update, i.e., the exploration was unsuccessful, the radius ρ is halved. This exploration is repeated for all the design variables and stopped when a prefixed budget (equal to 150 iterations as suggested in (Tseng and Chen, 2008)) is exceeded. The pseudo-code displaying the working principles of the deterministic short distance exploration is given in Algorithm 3.

After the application of the deterministic short distance exploration, if there is a fitness improvement for the current elite, the stochastic short distance exploration is repeated subsequently. Otherwise, if no improvement is found, the long distance search is activated in order to search for new basins of attractions, as depicted in Figure 1.

Algorithm 3 Deterministic short distance exploration

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

3 Numerical results

An extensive set of numerical experiments has been performed to test the performance of S-3SOME. To evaluate the scalability of the proposed algorithm S-3SOME, two complete benchmarks have been used, namely the noiseless Black Box Optimisation Benchmark (BBOB) 2010 (Hansen el al., 2010) and the benchmark proposed for the CEC 2010 Competition on Large-Scale Global Optimisation (Tang el al., 2010). The BBOB 2010, consisting of 24 scalable functions, has been tested in 10, 40, and 100 dimensions, while the 20 functions composing the CEC 2010 testbed have been tested in 1000 dimensions. Thus, a total of $24 \times 3 + 20 = 92$ optimisation problems with various features and dimensionalities has been tested.

The proposed S-3SOME, with $\alpha_e = 0.05$ and ρ equal to 40% of the total decision space width, has been compared with three recently proposed memory-saving algorithms employing different search logics:

1 compact Differential Evolution (cDE) with rand/1 mutation and exponential crossover, proposed in (Mininno el al., 2011), with virtual population size equal to 300, scale factor $F = 0.5$, and proportion of genes undergoing exponential crossover, see (Neri el al., 2011), $\alpha_m = 0.25$.

2 real-value compact Genetic Algorithm (cGA), proposed in (Mininno el al., 2008), with virtual population size equal to 300.

3 Simplified Intelligence Single Particle Optimisation (ISPO), proposed in (Zhou el al., 2008), 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 $PartLoop = 30$.

In addition, S-3SOME has been compared with the following modern algorithms:

1 Covariance Matrix Adaptive Evolution Strategy with increasing population size and restart (G-CMAES), proposed in (Auger and Hansen, 2005), with initial population $\lambda_{start} = 10$ and factor for increasing the population size equal to 2. All the other parameters are set to standard values.

2 Self-Adaptive Differential Evolution (SADE), proposed in (Qin el al., 2009), with Learning Period $LP = 20$ and population size $N_p = 50$. The other constant values are the same reported in the formulas of the original paper.

3 Cooperatively Coevolving Particle Swarms Optimiser (CCPSO2), proposed in (Li and Yao, 2012), with population size equal to $N_p = 30$ individuals, Cauchy/Gaussian-sampling selection probability $p = 0.5$ and set of potential group sizes $S = \{2, 5, 10\}$, $S = \{2, 5, 10, 50, 100\}$, $S = \{2, 5, 10, 50, 100, 250\}$ for experiments in 10–40, 100 and 1000 dimensions, respectively.

It is important to notice that these algorithms were grouped in two categories, based on their memory employment in terms of minimum number of memory slots, as defined above. In this sense, as shown in (Iacca el al., 2012), cDE requires 4 slots, S-3SOME 3 (as well as the original 3SOME), while cGA and ISPO only 2. Thus these algorithms can be considered truly memory-saving. On the other hand, the complex modern algorithms area characterised by a much heavier memory footprint: it can be easily proved that G-CMAES requires $2^{N_{restarts}}\lambda_{start} + n$ slots (where $N_{restarts}$ indicates the number of restarts performed, and the term n accounts for the covariance matrix, defined in $\mathbb{R}^{n \times n}$), SADE $N_p + 1 + archive$ (where $archive$ is a variable-size list of solutions), and CCPSO2 $2N_p + 2$.

It should also be remarked that the original 3SOME algorithm, proposed in (Iacca el al., 2012), has been purposely excluded from comparisons on the BBOB 2010 benchmark since a detailed comparison between S-3SOME and 3SOME has already been presented in (Poikolainen el al., 2012). On the other hand, we performed a comparison of the two algorithms on the CEC 2010 testbed, as described below.

For each algorithm and test problem considered in this study, 30 runs have been performed, each

one with a fixed budget which was set to $3000 \cdot n$ and $5000 \cdot n$ fitness evaluations, being n the problem dimensionality, respectively for the BBOB 2010 and the CEC 2010 testbeds. These budgets are those ones suggested in optimisation competitions and used in most of the literature in the field. Numerical results are reported in Tables 1-9, in terms of average final value and standard deviation computed over 30 runs of each algorithm. The best results (minimum average final value) are highlighted in bold face. The output of the Wilcoxon Rank-Sum test (Wilcoxon, 1945), applied with a confidence level of 0.95 on the pairwise comparison between S-3SOME and the algorithm labelled on the top of each column, is also shown. More specifically, a symbol "=" indicates that the null-hypothesis (i.e., the final values obtained by the two algorithms come from the same statistical distribution) cannot be rejected, meaning that the two algorithms have a statistically equivalent performance. In case of null-hypothesis rejection, the symbol "+" ("−") is used to indicate that S-3SOME has a better (worse) performance than the compared algorithm, in terms of average final value.

3.1 Low and medium scale problems

As shown in Table 1 the S-3SOME scheme appears to be, on a regular basis, more promising than ISPO. Compared to cDE and cGA, S-3SOME shows instead a global similar performance, which is slightly better on some separable functions (f2–f5), particularly unimodal. On non-separable multi-modal functions, especially cGA seems however to outperform S-3SOME. The comparison with the complex algorithm, reported in Table 2, shows instead that SADE outperforms regularly S-3SOME: this fact can be explained considering that SADE is an efficient structure designed for addressing problems with a low dimensionality (tuned for up to 30 dimensions). On the other hand, no clear outperformance trend emerges from the comparisons with G-CMAES (12 "+" and 12 "-") and CCPSO2 (11 "+", 8 "-" and 5 "="): these two algorithms can be considered essentially the same as S-3SOME, from a global performance point of view.

Similar results can be seen on 40-dimensional problems, where the performance of S-3SOME is even better. As shown in Table 3, S-3SOME clearly outperforms the other three memory-saving algorithms. In addition to that, the proposed algorithm shows again a similar global performance with respect to G-CMAES and CCPSO2, see Table 4. Interesting to notice, for this dimensionality the performance of SADE tends to deteriorate, being almost equaled by S-3SOME: as we have observed previously, this behaviour was expected since the SADE structure has been designed and tuned especially for problems up to 30 dimensions.

When the problem dimensionality grows up to 100 variables, the improvements produced by S-3SOME are even more evident, as shown in Tables 5 and 6: once again it outperforms all the three memory-saving algorithms, and shows similar performances with respect to G-CMAES, SADE and CCPSO2.

Two examples of average fitness trend on 100-dimensional problems are also illustrated in Figures 2 and 3.

Figure 2 Average fitness trend of memory-saving algorithms on f16 from BBOB 2010 in 100 dimensions.

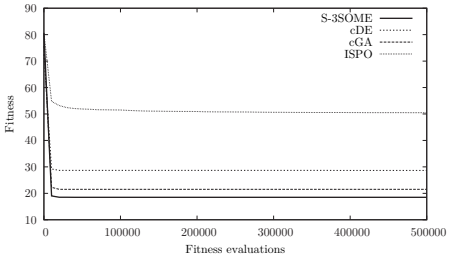
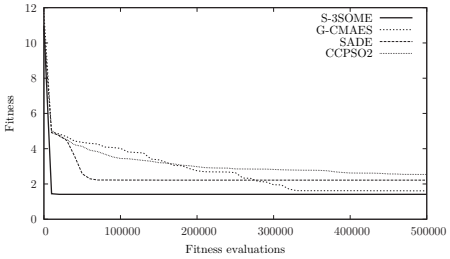


Figure 3 Average fitness trend of complex algorithms on f23 from BBOB 2010 in 100 dimensions.



3.2 Large scale problems

Tables 7 and 8 present the results obtained on the CEC 2010 benchmark, respectively with memory-saving and complex algorithms. It is clear from Table 7 that S-3SOME outperforms, also on large scale problems, all the memory-saving algorithms considered in this study. This result confirms the trend shown in the previous subsection on problems with 40 and 100 variables (Tables 3 and 4) and proves the superiority of S-3SOME, compared to other memory-saving algorithms, on problems with different features and different dimensionalities. This finding has a great importance from an application point of view: S-3SOME can be considered an excellent solution for problems that must be solved on systems endowed with limited memory.

The average fitness trend obtained by the memory-saving algorithms on the test function f10 is shown in Figure 4, where it can be seen how S-3SOME is able to improve upon the initial solution better than the other algorithms, which get stuck into different local optima.

Table 1 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with memory-saving algorithms on the BBOB 2010 in 10 dimensions

	S-3SOME	cDE		cGA		ISPO	
f1	$2.23e-14 \pm 1.23e-14$	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$1.04e-14 \pm 0.00e+00$	-	$\mathbf{0.00e+00 \pm 0.00e+00}$	-
f2	$2.56e-14 \pm 1.77e-14$	$2.27e-14 \pm 1.14e-14$	=	$8.19e-12 \pm 3.50e-11$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	-
f3	$\mathbf{6.90e-01 \pm 8.14e-01}$	$1.79e+00 \pm 1.44e+00$	+	$1.88e+01 \pm 7.12e+00$	+	$5.54e+01 \pm 3.69e+01$	+
f4	$\mathbf{1.18e+00 \pm 9.51e-01}$	$1.89e+00 \pm 1.42e+00$	+	$2.46e+01 \pm 9.44e+00$	+	$6.69e+01 \pm 3.81e+01$	+
f5	$1.07e-13 \pm 6.24e-14$	$6.22e-10 \pm 2.29e-10$	+	$2.33e-08 \pm 1.94e-08$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	-
f6	$\mathbf{1.33e-03 \pm 4.68e-03}$	$1.36e-01 \pm 2.81e-01$	+	$1.06e+00 \pm 1.11e+00$	+	$1.37e+01 \pm 1.66e+01$	+
f7	$8.04e+00 \pm 5.45e+00$	$6.49e+00 \pm 6.24e+00$	-	$\mathbf{4.74e+00 \pm 3.54e+00}$	-	$7.10e+00 \pm 6.28e+00$	=
f8	$\mathbf{1.10e-01 \pm 2.20e-01}$	$1.93e+00 \pm 1.93e+00$	+	$6.45e+00 \pm 1.15e+00$	+	$2.22e+00 \pm 1.68e+00$	+
f9	$\mathbf{1.31e+00 \pm 7.18e+00}$	$5.13e+00 \pm 1.64e+00$	+	$4.42e+01 \pm 5.16e+01$	+	$8.98e+00 \pm 2.27e+01$	+
f10	$\mathbf{3.12e+02 \pm 1.58e+02}$	$2.33e+03 \pm 2.22e+03$	+	$3.65e+03 \pm 2.54e+03$	+	$2.82e+03 \pm 1.93e+03$	+
f11	$8.80e+01 \pm 3.12e+01$	$6.98e+01 \pm 3.27e+01$	-	$\mathbf{5.53e+01 \pm 2.14e+01}$	-	$1.34e+02 \pm 4.44e+01$	+
f12	$1.09e+01 \pm 1.61e+01$	$7.40e+00 \pm 1.30e+01$	=	$\mathbf{6.01e+00 \pm 8.95e+00}$	=	$1.83e+01 \pm 2.06e+01$	+
f13	$9.49e+00 \pm 1.01e+01$	$1.08e+01 \pm 9.43e+00$	=	$\mathbf{7.52e+00 \pm 8.81e+00}$	=	$2.91e+01 \pm 1.50e+01$	+
f14	$\mathbf{1.01e-04 \pm 2.45e-05}$	$3.03e-04 \pm 6.93e-05$	+	$6.55e-04 \pm 4.46e-04$	+	$4.10e-04 \pm 1.03e-04$	+
f15	$5.72e+01 \pm 2.66e+01$	$4.49e+01 \pm 1.93e+01$	-	$\mathbf{2.11e+01 \pm 9.85e+00}$	-	$4.16e+02 \pm 1.44e+02$	+
f16	$4.07e+00 \pm 2.49e+00$	$4.04e+00 \pm 2.39e+00$	=	$\mathbf{2.32e+00 \pm 1.52e+00}$	-	$2.14e+01 \pm 1.67e+01$	+
f17	$2.58e+00 \pm 1.73e+00$	$1.46e+00 \pm 8.92e-01$	-	$\mathbf{5.70e-01 \pm 5.86e-01}$	-	$5.09e+01 \pm 5.83e+01$	+
f18	$8.60e+00 \pm 5.93e+00$	$4.75e+00 \pm 2.84e+00$	-	$\mathbf{2.05e+00 \pm 2.60e+00}$	-	$2.79e+02 \pm 4.27e+02$	+
f19	$2.62e+00 \pm 1.42e+00$	$2.08e+00 \pm 1.01e+00$	=	$\mathbf{1.54e+00 \pm 8.41e-01}$	-	$2.28e+01 \pm 1.24e+01$	+
f20	$6.63e-01 \pm 2.56e-01$	$\mathbf{5.49e-01 \pm 2.26e-01}$	-	$1.47e+00 \pm 2.97e-01$	+	$1.10e+00 \pm 2.78e-01$	+
f21	$3.38e+00 \pm 3.47e+00$	$3.93e+00 \pm 3.60e+00$	=	$\mathbf{3.00e+00 \pm 3.13e+00}$	=	$1.28e+01 \pm 1.23e+01$	+
f22	$2.75e+00 \pm 4.04e+00$	$5.73e+00 \pm 8.81e+00$	=	$\mathbf{2.21e+00 \pm 1.88e+00}$	=	$1.62e+01 \pm 1.98e+01$	+
f23	$7.56e-01 \pm 2.47e-01$	$6.50e-01 \pm 2.65e-01$	-	$\mathbf{4.64e-01 \pm 3.51e-01}$	-	$1.64e+00 \pm 5.64e-01$	+
f24	$5.14e+01 \pm 1.83e+01$	$4.09e+01 \pm 1.40e+01$	-	$\mathbf{3.95e+01 \pm 1.43e+01}$	-	$2.61e+02 \pm 8.05e+01$	+

Table 2 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with complex algorithms on the BBOB 2010 in 10 dimensions

	S-3SOME	G-CMAES		SADE		CCPSO2	
f1	$2.23e-14 \pm 1.23e-14$	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$7.25e-07 \pm 1.99e-06$	+
f2	$2.56e-14 \pm 1.77e-14$	$1.56e+03 \pm 1.27e+03$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$2.19e-01 \pm 5.23e-01$	+
f3	$\mathbf{6.90e-01 \pm 8.14e-01}$	$4.12e+02 \pm 1.13e+02$	+	$2.75e+00 \pm 2.10e+00$	+	$3.39e+00 \pm 1.69e+00$	+
f4	$\mathbf{1.18e+00 \pm 9.51e-01}$	$4.66e+02 \pm 8.11e+01$	+	$3.93e+00 \pm 2.45e+00$	+	$3.95e+00 \pm 1.64e+00$	+
f5	$1.07e-13 \pm 6.24e-14$	$2.14e+01 \pm 8.25e+00$	+	$\mathbf{9.44e-14 \pm 1.50e-13}$	-	$5.83e-01 \pm 6.57e-01$	+
f6	$1.33e-03 \pm 4.68e-03$	$\mathbf{1.18e-14 \pm 0.00e+00}$	-	$2.81e-14 \pm 1.34e-13$	-	$1.68e+00 \pm 1.15e+00$	+
f7	$8.04e+00 \pm 5.45e+00$	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$2.31e-01 \pm 3.13e-01$	-	$2.47e+00 \pm 6.63e-01$	-
f8	$1.10e-01 \pm 2.20e-01$	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$2.39e-02 \pm 3.08e-01$	-	$1.99e+00 \pm 2.19e+00$	+
f9	$1.31e+00 \pm 7.18e+00$	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$3.88e-02 \pm 3.77e-01$	-	$2.26e+00 \pm 1.44e+00$	+
f10	$3.12e+02 \pm 1.58e+02$	$8.19e+02 \pm 1.14e+03$	+	$\mathbf{4.23e+01 \pm 5.70e+01}$	-	$5.87e+03 \pm 3.56e+03$	+
f11	$8.80e+01 \pm 3.12e+01$	$\mathbf{0.00e+00 \pm 0.00e+00}$	-	$1.10e+00 \pm 1.36e+00$	-	$2.10e+01 \pm 6.57e+00$	-
f12	$1.09e+01 \pm 1.61e+01$	$1.22e+04 \pm 1.53e+04$	+	$\mathbf{4.28e-01 \pm 8.04e-01}$	-	$4.02e+00 \pm 7.84e+00$	-
f13	$9.49e+00 \pm 1.01e+01$	$\mathbf{1.98e-11 \pm 3.18e-11}$	-	$1.32e-02 \pm 3.38e-02$	-	$2.42e+00 \pm 1.97e+00$	-
f14	$1.01e-04 \pm 2.45e-05$	$6.99e+01 \pm 6.01e+01$	+	$\mathbf{1.16e-05 \pm 1.05e-05}$	-	$1.52e-03 \pm 1.08e-03$	+
f15	$5.72e+01 \pm 2.66e+01$	$\mathbf{1.45e+00 \pm 1.02e+00}$	-	$1.04e+01 \pm 4.91e+00$	-	$4.21e+01 \pm 1.20e+01$	-
f16	$4.07e+00 \pm 2.49e+00$	$\mathbf{4.37e-03 \pm 1.07e-02}$	-	$9.48e-01 \pm 8.54e-01$	-	$4.11e+00 \pm 1.16e+00$	=
f17	$2.58e+00 \pm 1.73e+00$	$4.00e+01 \pm 2.95e+01$	+	$\mathbf{9.04e-04 \pm 2.16e-03}$	-	$7.17e-01 \pm 2.44e-01$	-
f18	$8.60e+00 \pm 5.93e+00$	$1.37e+02 \pm 1.30e+02$	+	$\mathbf{2.43e-02 \pm 2.64e-02}$	-	$2.51e+00 \pm 8.58e-01$	-
f19	$2.62e+00 \pm 1.42e+00$	$3.88e+01 \pm 1.87e+01$	+	$\mathbf{5.96e-01 \pm 5.50e-01}$	-	$2.10e+00 \pm 3.86e-01$	=
f20	$6.63e-01 \pm 2.56e-01$	$1.53e+04 \pm 1.33e+04$	+	$7.59e-01 \pm 2.81e-01$	+	$\mathbf{4.68e-01 \pm 1.70e-01}$	-
f21	$3.38e+00 \pm 3.47e+00$	$1.67e+00 \pm 1.15e+00$	-	$\mathbf{1.14e+00 \pm 8.75e-01}$	-	$1.64e+00 \pm 1.41e+00$	=
f22	$2.75e+00 \pm 4.04e+00$	$8.19e+01 \pm 4.66e+00$	+	$1.92e+00 \pm 2.74e-01$	-	$\mathbf{1.23e+00 \pm 1.16e+00}$	=
f23	$7.56e-01 \pm 2.47e-01$	$\mathbf{1.06e-01 \pm 2.28e-01}$	-	$4.33e-01 \pm 4.23e-01$	-	$1.39e+00 \pm 2.39e-01$	+
f24	$5.14e+01 \pm 1.83e+01$	$\mathbf{8.72e+00 \pm 4.55e+00}$	-	$1.84e+01 \pm 4.67e+00$	-	$4.50e+01 \pm 7.31e+00$	=

The comparison with the complex state-of-the-art algorithms reported in Table 8 also suggests some interesting considerations. First of all, while G-CMAES slightly outperforms S-3SOME, which was expected considering the complexity of G-CMAES compared to S-3SOME, it's interesting to notice that SADE and CCPSO2, despite their complexity, are at least equaled, if not outperformed by S-3SOME. More specifically, S-3SOME tends to outperform G-CMAES and SADE on separable functions (f1–f3) and some non-separable multi-modal functions. The comparison with CCPSO2 is more difficult to interpret, although it seems that S-3SOME shows a better performance with functions characterised by a high level of multi-modality (second half of the benchmark).

The average fitness trend obtained by the complex algorithms on the test function f11 is shown in Figure 4, where it is possible to notice that G-CMAES gets stuck into a local optimum, while SADE and CCPSO2

Table 3 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with memory-saving algorithms on the BBOB 2010 in 40 dimensions

	S-3SOME	cDE		cGA		ISPO	
f1	$9.81e-14 \pm 2.30e-14$	$1.47e-10 \pm 7.39e-10$	=	$7.66e-01 \pm 6.03e-01$	+	$\mathbf{4.03e-14 \pm 0.00e+00}$	-
f2	$1.43e-13 \pm 2.98e-14$	$5.00e-01 \pm 2.68e+00$	+	$2.67e+03 \pm 1.69e+03$	+	$\mathbf{8.05e-14 \pm 1.29e-14}$	-
f3	$\mathbf{1.08e+01 \pm 3.10e+00}$	$7.46e+01 \pm 1.79e+01$	+	$2.71e+02 \pm 6.83e+01$	+	$1.94e+02 \pm 5.66e+01$	+
f4	$\mathbf{1.54e+01 \pm 3.14e+00}$	$8.76e+01 \pm 1.96e+01$	+	$4.56e+02 \pm 9.71e+01$	+	$2.89e+02 \pm 1.02e+02$	+
f5	$2.45e-12 \pm 9.36e-13$	$1.28e-02 \pm 6.89e-02$	+	$2.19e-04 \pm 7.91e-04$	+	$\mathbf{-3.55e-14 \pm 0.00e+00}$	-
f6	$\mathbf{8.87e-07 \pm 4.72e-06}$	$2.01e+02 \pm 7.30e+01$	+	$2.46e+02 \pm 7.45e+01$	+	$5.29e+01 \pm 5.60e+01$	+
f7	$\mathbf{8.91e+01 \pm 3.18e+01}$	$1.95e+02 \pm 5.19e+01$	+	$1.71e+02 \pm 5.57e+01$	+	$1.45e+02 \pm 5.57e+01$	+
f8	$\mathbf{3.32e-01 \pm 5.31e-01}$	$6.68e+01 \pm 6.88e+01$	+	$5.00e+02 \pm 2.75e+02$	+	$3.96e+01 \pm 3.73e+01$	+
f9	$\mathbf{1.31e+00 \pm 1.72e+00}$	$8.44e+01 \pm 7.68e+01$	+	$1.09e+03 \pm 6.88e+02$	+	$3.17e+01 \pm 9.28e+00$	+
f10	$\mathbf{9.77e+02 \pm 3.04e+02}$	$8.97e+04 \pm 3.64e+04$	+	$2.52e+05 \pm 1.27e+05$	+	$1.40e+04 \pm 4.66e+03$	+
f11	$2.96e+02 \pm 7.46e+01$	$\mathbf{2.85e+02 \pm 5.68e+01}$	=	$3.37e+02 \pm 6.24e+01$	+	$4.79e+02 \pm 9.67e+01$	+
f12	$\mathbf{6.48e+00 \pm 6.67e+00}$	$6.51e+04 \pm 2.00e+05$	+	$1.69e+06 \pm 1.66e+06$	+	$1.70e+01 \pm 1.37e+01$	+
f13	$\mathbf{9.65e+00 \pm 9.28e+00}$	$4.82e+01 \pm 2.63e+01$	+	$3.70e+02 \pm 1.04e+02$	+	$1.47e+01 \pm 1.36e+01$	=
f14	$\mathbf{3.08e-04 \pm 5.49e-05}$	$8.85e-02 \pm 1.34e-01$	+	$3.00e+00 \pm 1.37e+00$	+	$1.61e-03 \pm 3.15e-04$	+
f15	$3.88e+02 \pm 1.04e+02$	$3.92e+02 \pm 1.19e+02$	=	$\mathbf{2.96e+02 \pm 7.48e+01}$	-	$2.65e+03 \pm 6.48e+02$	+
f16	$1.42e+01 \pm 4.25e+00$	$1.64e+01 \pm 3.90e+00$	+	$\mathbf{1.28e+01 \pm 3.84e+00}$	=	$3.54e+01 \pm 7.71e+00$	+
f17	$6.91e+00 \pm 1.47e+00$	$6.08e+00 \pm 1.37e+00$	-	$\mathbf{4.87e+00 \pm 1.38e+00}$	-	$2.41e+01 \pm 1.23e+01$	+
f18	$2.22e+01 \pm 5.74e+00$	$2.27e+01 \pm 3.66e+00$	=	$\mathbf{1.86e+01 \pm 4.20e+00}$	-	$1.25e+02 \pm 8.17e+01$	+
f19	$\mathbf{6.13e+00 \pm 2.38e+00}$	$7.26e+00 \pm 1.97e+00$	+	$6.83e+00 \pm 1.74e+00$	=	$4.64e+01 \pm 1.29e+01$	+
f20	$\mathbf{8.96e-01 \pm 1.30e-01}$	$1.01e+00 \pm 1.62e-01$	+	$2.17e+00 \pm 5.25e-01$	+	$1.28e+00 \pm 1.58e-01$	+
f21	$1.19e+01 \pm 1.50e+01$	$\mathbf{3.94e+00 \pm 5.58e+00}$	-	$4.89e+00 \pm 5.52e+00$	=	$1.55e+01 \pm 1.92e+01$	=
f22	$1.49e+01 \pm 1.55e+01$	$1.48e+01 \pm 6.10e+00$	+	$\mathbf{1.28e+01 \pm 7.44e+00}$	+	$1.77e+01 \pm 1.47e+01$	=
f23	$\mathbf{1.05e+00 \pm 4.17e-01}$	$1.79e+00 \pm 5.40e-01$	+	$1.14e+00 \pm 3.99e-01$	=	$3.15e+00 \pm 9.00e-01$	+
f24	$\mathbf{3.88e+02 \pm 7.13e+01}$	$4.16e+02 \pm 6.53e+01$	=	$4.17e+02 \pm 8.56e+01$	=	$1.27e+03 \pm 1.79e+02$	+

Table 4 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with complex algorithms on the BBOB 2010 in 40 dimensions

	S-3SOME	G-CMAES		SADE		CCPSO2	
f1	$9.81e-14 \pm 2.30e-14$	$\mathbf{1.56e-14 \pm 0.00e+00}$	-	$\mathbf{1.04e-14 \pm 1.03e-14}$	-	$1.28e-13 \pm 5.81e-14$	+
f2	$1.43e-13 \pm 2.98e-14$	$5.17e+02 \pm 2.05e+02$	+	$\mathbf{6.73e-14 \pm 2.25e-13}$	-	$3.82e-13 \pm 4.32e-13$	+
f3	$1.08e+01 \pm 3.10e+00$	$5.44e+02 \pm 6.34e+01$	+	$3.57e+01 \pm 1.60e+01$	+	$\mathbf{8.16e-01 \pm 7.69e-01}$	-
f4	$1.54e+01 \pm 3.14e+00$	$5.48e+02 \pm 4.96e+01$	+	$5.39e+01 \pm 2.25e+01$	+	$\mathbf{4.47e+00 \pm 1.68e+00}$	-
f5	$\mathbf{2.45e-12 \pm 9.36e-13}$	$2.54e+01 \pm 8.99e+00$	+	$1.84e-10 \pm 3.47e-10$	+	$1.37e-10 \pm 4.95e-10$	+
f6	$8.87e-07 \pm 4.72e-06$	$\mathbf{4.41e-14 \pm 0.00e+00}$	-	$1.35e+00 \pm 2.47e+00$	+	$2.36e+01 \pm 1.77e+01$	+
f7	$8.91e+01 \pm 3.18e+01$	$\mathbf{4.11e+00 \pm 4.13e+00}$	-	$3.61e+01 \pm 1.19e+01$	-	$5.64e+01 \pm 1.23e+01$	-
f8	$3.32e-01 \pm 5.31e-01$	$\mathbf{3.03e-14 \pm 0.00e+00}$	-	$3.72e+01 \pm 2.63e+01$	+	$3.40e+01 \pm 2.63e+01$	+
f9	$1.31e+00 \pm 1.72e+00$	$\mathbf{1.33e-01 \pm 7.16e-01}$	-	$3.41e+01 \pm 1.49e+00$	+	$3.48e+01 \pm 1.06e+01$	+
f10	$9.77e+02 \pm 3.04e+02$	$\mathbf{1.34e+02 \pm 5.36e+01}$	-	$6.09e+03 \pm 2.10e+03$	+	$1.12e+04 \pm 5.32e+03$	+
f11	$2.96e+02 \pm 7.46e+01$	$\mathbf{2.40e-14 \pm 0.00e+00}$	-	$3.46e+01 \pm 1.29e+01$	-	$1.78e+02 \pm 4.92e+01$	-
f12	$\mathbf{6.48e+00 \pm 6.67e+00}$	$4.57e+03 \pm 4.68e+03$	+	$8.74e+00 \pm 6.35e+00$	=	$1.63e+01 \pm 8.10e+00$	+
f13	$9.65e+00 \pm 9.28e+00$	$\mathbf{7.65e-05 \pm 2.31e-04}$	-	$6.86e+00 \pm 5.80e+00$	=	$7.71e+00 \pm 7.51e+00$	=
f14	$\mathbf{3.08e-04 \pm 5.49e-05}$	$9.08e+01 \pm 2.74e+01$	+	$4.68e-04 \pm 1.36e-04$	+	$8.60e-04 \pm 1.69e-04$	+
f15	$3.88e+02 \pm 1.04e+02$	$\mathbf{1.10e+01 \pm 2.28e+00}$	-	$7.14e+01 \pm 2.51e+01$	-	$3.70e+02 \pm 8.74e+01$	=
f16	$1.42e+01 \pm 4.25e+00$	$\mathbf{3.03e-02 \pm 3.60e-02}$	-	$1.12e+01 \pm 3.38e+00$	-	$1.40e+01 \pm 3.11e+00$	=
f17	$6.91e+00 \pm 1.47e+00$	$2.75e+01 \pm 9.16e+00$	+	$\mathbf{8.30e-01 \pm 3.22e-01}$	-	$5.57e+00 \pm 1.78e+00$	-
f18	$2.22e+01 \pm 5.74e+00$	$4.00e+01 \pm 1.97e+01$	+	$\mathbf{3.32e+00 \pm 8.50e-01}$	-	$1.83e+01 \pm 5.42e+00$	-
f19	$6.13e+00 \pm 2.38e+00$	$5.19e+01 \pm 1.42e+01$	+	$\mathbf{1.45e+00 \pm 7.96e-01}$	-	$6.22e+00 \pm 3.69e-01$	=
f20	$8.96e-01 \pm 1.30e-01$	$4.33e+03 \pm 3.00e+03$	+	$1.84e+00 \pm 1.87e-01$	+	$\mathbf{4.77e-01 \pm 8.79e-02}$	-
f21	$1.19e+01 \pm 1.50e+01$	$1.08e+00 \pm 1.24e+00$	-	$2.93e+00 \pm 2.46e+00$	-	$\mathbf{9.98e-01 \pm 1.14e+00}$	-
f22	$1.49e+01 \pm 1.55e+01$	$8.61e+01 \pm 3.74e-01$	+	$1.07e+01 \pm 8.15e+00$	-	$\mathbf{2.74e+00 \pm 3.28e+00}$	-
f23	$1.05e+00 \pm 4.17e-01$	$\mathbf{8.92e-01 \pm 1.37e+00}$	-	$1.32e+00 \pm 5.41e-01$	=	$2.65e+00 \pm 4.42e-01$	+
f24	$3.88e+02 \pm 7.13e+01$	$\mathbf{3.97e+01 \pm 1.98e+01}$	-	$8.52e+01 \pm 1.09e+01$	-	$2.60e+02 \pm 5.16e+01$	-

converge slower than S-3SOME, and to worse final values.

To further strengthen the improvements reached with S-SOME, we also compared its performance on the CEC 2010 testbed with its inspiring memetic framework, i.e., the 3SOME algorithm proposed in (Iacca el al., 2012). 3SOME was executed with the same parameter values of α_e and ρ used for S-3SOME in the experiments described above, while the number of solutions generated during each activation of the middle distance exploration was set to $4 \cdot n$, as suggested in the original paper. Table 9 reports the numerical results of this comparison. It can be noticed that 3-3SOME seems to generally outperform the original 3SOME (13 "+", 5 "-" and 2 "="), while 3SOME outperforms S-3SOME on some unimodal (partially) non-separable functions (f4, f7 and f19). This result can be seen as a consequence of the shrinking mechanism, which tends to rapidly exploit the promising regions found by the long distance focusing the search over them. In case of large scale problems, this

Table 5 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with memory-saving algorithms on the BBOB 2010 in 100 dimensions

	S-3SOME	cDE		cGA		ISPO	
f1	$2.52e-13 \pm 2.38e-14$	$1.55e+01 \pm 9.03e+00$	+	$8.72e+01 \pm 2.74e+01$	+	$\mathbf{1.08e-13 \pm 1.25e-14}$	-
f2	$3.92e-13 \pm 4.83e-14$	$2.85e+05 \pm 4.50e+05$	+	$5.25e+05 \pm 2.71e+05$	+	$\mathbf{1.98e-13 \pm 2.26e-14}$	-
f3	$\mathbf{4.51e+01 \pm 6.25e+00}$	$6.01e+02 \pm 9.86e+01$	+	$1.69e+03 \pm 3.58e+02$	+	$5.75e+02 \pm 1.19e+02$	+
f4	$\mathbf{6.41e+01 \pm 9.26e+00}$	$1.03e+03 \pm 1.70e+02$	+	$3.22e+03 \pm 4.80e+02$	+	$6.84e+02 \pm 1.46e+02$	+
f5	$1.34e-11 \pm 4.63e-12$	$2.53e+01 \pm 9.38e+00$	+	$7.11e+01 \pm 3.60e+01$	+	$\mathbf{-3.55e-14 \pm 0.00e+00}$	-
f6	$\mathbf{8.06e-09 \pm 2.85e-08}$	$1.33e+03 \pm 1.95e+02$	+	$2.14e+03 \pm 1.89e+03$	+	$2.11e+02 \pm 9.09e+01$	+
f7	$\mathbf{3.03e+02 \pm 1.10e+02}$	$1.18e+03 \pm 2.45e+02$	+	$1.09e+03 \pm 2.33e+02$	+	$1.96e+03 \pm 5.75e+02$	+
f8	$\mathbf{3.74e+01 \pm 3.84e+01}$	$6.44e+03 \pm 3.83e+03$	+	$1.93e+05 \pm 1.01e+05$	+	$1.17e+02 \pm 3.82e+01$	+
f9	$\mathbf{5.15e+01 \pm 1.79e+01}$	$1.93e+04 \pm 1.18e+04$	+	$1.43e+05 \pm 5.01e+04$	+	$9.13e+01 \pm 1.27e+01$	+
f10	$\mathbf{3.13e+03 \pm 6.46e+02}$	$1.42e+06 \pm 3.80e+05$	+	$2.22e+06 \pm 4.99e+05$	+	$3.24e+04 \pm 6.15e+03$	+
f11	$\mathbf{5.58e+02 \pm 1.04e+02}$	$8.49e+02 \pm 1.04e+02$	+	$9.78e+02 \pm 1.04e+02$	+	$1.12e+03 \pm 1.59e+02$	+
f12	$\mathbf{3.81e+00 \pm 5.10e+00}$	$4.72e+07 \pm 2.37e+07$	+	$2.20e+08 \pm 7.38e+07$	+	$8.78e+00 \pm 1.79e+01$	=
f13	$5.81e+00 \pm 4.55e+00$	$1.23e+03 \pm 1.64e+02$	+	$2.03e+03 \pm 3.08e+02$	+	$\mathbf{3.76e+00 \pm 4.01e+00}$	-
f14	$\mathbf{3.08e-04 \pm 5.65e-05}$	$1.99e+01 \pm 3.64e+00$	+	$3.49e+01 \pm 5.28e+00$	+	$2.40e-03 \pm 5.54e-04$	+
f15	$\mathbf{1.45e+03 \pm 2.94e+02}$	$1.93e+03 \pm 3.35e+02$	+	$1.75e+03 \pm 2.86e+02$	+	$6.85e+03 \pm 1.20e+03$	+
f16	$\mathbf{1.85e+01 \pm 5.16e+00}$	$2.87e+01 \pm 4.61e+00$	+	$2.15e+01 \pm 4.01e+00$	+	$5.05e+01 \pm 8.30e+00$	+
f17	$\mathbf{8.58e+00 \pm 1.40e+00}$	$1.02e+01 \pm 1.68e+00$	+	$9.63e+00 \pm 1.13e+00$	+	$4.77e+01 \pm 4.08e+01$	+
f18	$\mathbf{3.31e+01 \pm 5.57e+00}$	$4.17e+01 \pm 6.39e+00$	+	$3.61e+01 \pm 5.90e+00$	+	$1.67e+02 \pm 9.53e+01$	+
f19	$\mathbf{9.53e+00 \pm 2.11e+00}$	$1.77e+01 \pm 2.63e+00$	+	$1.82e+01 \pm 3.14e+00$	+	$1.22e+02 \pm 2.00e+01$	+
f20	$\mathbf{9.26e-01 \pm 7.75e-02}$	$1.73e+00 \pm 1.52e-01$	+	$1.70e+04 \pm 9.71e+03$	+	$1.27e+00 \pm 8.71e-02$	+
f21	$1.25e+01 \pm 1.20e+01$	$1.58e+01 \pm 9.15e+00$	=	$2.59e+01 \pm 1.14e+01$	+	$\mathbf{1.12e+01 \pm 1.15e+01}$	=
f22	$1.58e+01 \pm 1.75e+01$	$1.78e+01 \pm 1.11e+01$	+	$2.75e+01 \pm 1.69e+01$	+	$\mathbf{1.34e+01 \pm 1.12e+01}$	=
f23	$\mathbf{1.41e+00 \pm 3.67e-01}$	$2.95e+00 \pm 5.66e-01$	+	$1.83e+00 \pm 4.29e-01$	+	$3.90e+00 \pm 6.96e-01$	+
f24	$\mathbf{1.46e+03 \pm 1.99e+02}$	$1.67e+03 \pm 1.48e+02$	+	$2.12e+03 \pm 2.33e+02$	+	$3.59e+03 \pm 3.46e+02$	+

Table 6 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with complex algorithms on the BBOB 2010 in 100 dimensions

	S-3SOME	G-CMAES		SADE		CCPSO2	
f1	$2.52e-13 \pm 2.38e-14$	$\mathbf{2.89e-14 \pm 0.00e+00}$	-	$1.92e-13 \pm 2.23e-13$	-	$3.46e-13 \pm 2.19e-13$	=
f2	$\mathbf{3.92e-13 \pm 4.83e-14}$	$3.50e+02 \pm 7.34e+01$	+	$6.12e-13 \pm 9.92e-13$	=	$8.69e-13 \pm 7.75e-13$	+
f3	$4.51e+01 \pm 6.25e+00$	$6.93e+02 \pm 1.58e+02$	+	$1.70e+02 \pm 4.68e+01$	+	$\mathbf{1.01e+01 \pm 8.65e+00}$	-
f4	$6.41e+01 \pm 9.26e+00$	$8.03e+02 \pm 1.57e+02$	+	$2.78e+02 \pm 7.73e+01$	+	$\mathbf{1.85e+01 \pm 1.24e+01}$	-
f5	$\mathbf{1.34e-11 \pm 4.63e-12}$	$1.27e+02 \pm 3.06e+01$	+	$1.69e-01 \pm 7.97e-01$	+	$4.93e-06 \pm 2.50e-05$	+
f6	$8.06e-09 \pm 2.85e-08$	$\mathbf{3.36e-13 \pm 1.14e-13}$	-	$8.18e+01 \pm 4.38e+01$	+	$7.89e+01 \pm 3.60e+01$	+
f7	$3.03e+02 \pm 1.10e+02$	$\mathbf{3.46e+01 \pm 7.29e+00}$	-	$2.77e+02 \pm 9.83e+01$	=	$3.54e+02 \pm 4.98e+01$	+
f8	$3.74e+01 \pm 3.84e+01$	$\mathbf{1.10e+00 \pm 2.02e+00}$	-	$1.38e+02 \pm 4.89e+01$	+	$1.21e+02 \pm 3.45e+01$	+
f9	$5.15e+01 \pm 1.79e+01$	$\mathbf{7.52e-01 \pm 1.36e+00}$	-	$1.01e+02 \pm 1.64e+01$	+	$1.11e+02 \pm 3.40e+01$	+
f10	$3.13e+03 \pm 6.46e+02$	$\mathbf{1.18e+02 \pm 3.69e+01}$	-	$4.88e+04 \pm 2.53e+04$	+	$2.91e+04 \pm 1.05e+04$	+
f11	$5.58e+02 \pm 1.04e+02$	$\mathbf{8.05e-14 \pm 1.34e-14}$	-	$1.03e+02 \pm 2.07e+01$	-	$5.58e+02 \pm 2.17e+02$	=
f12	$\mathbf{3.81e+00 \pm 5.10e+00}$	$1.51e+03 \pm 8.00e+02$	+	$5.90e+00 \pm 7.30e+00$	=	$9.47e+00 \pm 1.49e+01$	=
f13	$5.81e+00 \pm 4.55e+00$	$\mathbf{2.85e-02 \pm 4.17e-02}$	-	$1.53e+00 \pm 1.88e+00$	-	$2.74e+00 \pm 3.17e+00$	-
f14	$\mathbf{3.08e-04 \pm 5.65e-05}$	$8.07e+01 \pm 1.66e+01$	+	$5.64e-03 \pm 9.96e-03$	+	$1.27e-03 \pm 1.58e-04$	+
f15	$1.45e+03 \pm 2.94e+02$	$\mathbf{3.73e+01 \pm 3.18e+01}$	-	$3.49e+02 \pm 6.97e+01$	-	$1.41e+03 \pm 2.33e+02$	=
f16	$1.85e+01 \pm 5.16e+00$	$\mathbf{1.29e-01 \pm 8.03e-02}$	-	$2.53e+01 \pm 4.22e+00$	+	$2.71e+01 \pm 3.94e+00$	+
f17	$8.58e+00 \pm 1.40e+00$	$2.70e+01 \pm 5.38e+00$	+	$\mathbf{3.29e+00 \pm 5.89e-01}$	-	$8.74e+00 \pm 1.93e+00$	=
f18	$3.31e+01 \pm 5.57e+00$	$4.02e+01 \pm 1.44e+01$	=	$\mathbf{1.15e+01 \pm 1.66e+00}$	-	$3.31e+01 \pm 4.69e+00$	=
f19	$9.53e+00 \pm 2.11e+00$	$1.10e+02 \pm 1.57e+01$	+	$\mathbf{3.16e+00 \pm 9.98e-01}$	-	$7.89e+00 \pm 1.07e+00$	-
f20	$9.26e-01 \pm 7.75e-02$	$3.29e+03 \pm 1.96e+03$	+	$2.04e+00 \pm 1.63e-01$	+	$\mathbf{4.99e-01 \pm 7.67e-02}$	-
f21	$1.25e+01 \pm 1.20e+01$	$\mathbf{1.99e+00 \pm 1.56e+00}$	-	$6.18e+00 \pm 6.08e+00$	-	$3.63e+00 \pm 2.95e+00$	-
f22	$1.58e+01 \pm 1.75e+01$	$8.65e+01 \pm 7.01e-02$	+	$5.22e+00 \pm 6.53e+00$	-	$\mathbf{3.66e+00 \pm 6.54e+00}$	=
f23	$\mathbf{1.41e+00 \pm 3.67e-01}$	$1.61e+00 \pm 2.03e+00$	=	$2.22e+00 \pm 7.32e-01$	+	$2.54e+00 \pm 4.46e-01$	+
f24	$1.46e+03 \pm 1.99e+02$	$\mathbf{2.28e+02 \pm 8.75e+01}$	-	$2.74e+02 \pm 2.81e+01$	-	$1.06e+03 \pm 1.70e+02$	-

effect seems to be especially beneficial on multi-modal functions, where a strong exploitation is needed as soon as the global search component finds a promising area of the search space. On the contrary, on unimodal functions this exploitation pressure appears to be detrimental, since it can focus the search on regions relatively far from the single optimum. As a conjecture, due to the random search within fixed bounds performed by the original middle distance exploration, the 3SOME algorithm appears instead to be able to avoid suboptimal areas and, possibly, detect solutions that are closer to the optimum.

3.3 Algorithmic overhead

In order to highlight the difference, in terms of computational overhead, among the algorithms considered in this study, we recorded the execution time needed by all of them to perform an optimisation process of the test function f1 from the BBOB 2010 (sphere

Table 7 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with memory-saving algorithms on the CEC 2010 benchmark in 1000 dimensions

	S-3SOME	cDE		cGA		ISPO	
f1	$6.03e-04 \pm 4.71e-04$	$1.70e+11 \pm 1.67e+10$	+	$1.08e+11 \pm 1.60e+10$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	-
f2	$\mathbf{3.99e-02 \pm 9.13e-03}$	$1.49e+04 \pm 3.51e+02$	+	$2.02e+04 \pm 4.59e+02$	+	$1.38e+04 \pm 4.22e+02$	+
f3	$\mathbf{7.56e-03 \pm 6.83e-04}$	$2.08e+01 \pm 4.19e-02$	+	$2.12e+01 \pm 4.14e-02$	+	$1.99e+01 \pm 1.38e-02$	+
f4	$2.08e+13 \pm 6.90e+12$	$9.51e+13 \pm 4.02e+13$	+	$2.02e+14 \pm 6.92e+13$	+	$\mathbf{8.49e+12 \pm 2.60e+12}$	-
f5	$4.27e+08 \pm 1.16e+08$	$3.79e+08 \pm 7.46e+07$	=	$\mathbf{2.75e+08 \pm 6.82e+07}$	-	$8.84e+08 \pm 1.48e+08$	+
f6	$1.54e+07 \pm 5.39e+06$	$1.76e+07 \pm 2.17e+06$	=	$\mathbf{1.30e+07 \pm 3.27e+06}$	=	$1.98e+07 \pm 4.88e+04$	+
f7	$\mathbf{1.07e+10 \pm 2.83e+09}$	$3.99e+10 \pm 1.23e+10$	+	$6.69e+10 \pm 1.70e+10$	+	$3.85e+10 \pm 1.81e+10$	+
f8	$1.78e+09 \pm 2.68e+09$	$4.56e+11 \pm 6.70e+11$	+	$5.82e+13 \pm 5.83e+13$	+	$\mathbf{1.33e+09 \pm 2.12e+09}$	-
f9	$3.54e+08 \pm 7.89e+07$	$7.44e+10 \pm 7.07e+09$	+	$1.10e+11 \pm 1.54e+10$	+	$\mathbf{8.55e+07 \pm 9.80e+06}$	-
f10	$\mathbf{5.12e+03 \pm 2.63e+02}$	$1.80e+04 \pm 5.19e+02$	+	$2.02e+04 \pm 4.75e+02$	+	$1.49e+04 \pm 4.90e+02$	+
f11	$\mathbf{1.97e+02 \pm 4.27e+00}$	$2.31e+02 \pm 4.61e-01$	+	$2.31e+02 \pm 3.94e-01$	+	$2.18e+02 \pm 2.25e-01$	+
f12	$\mathbf{8.74e+04 \pm 2.12e+04}$	$6.18e+06 \pm 4.44e+05$	+	$8.88e+06 \pm 6.73e+05$	+	$2.21e+05 \pm 2.84e+04$	+
f13	$5.61e+05 \pm 7.34e+05$	$6.88e+11 \pm 7.25e+10$	+	$1.53e+12 \pm 1.93e+11$	+	$\mathbf{5.98e+03 \pm 4.10e+03}$	-
f14	$\mathbf{8.79e+07 \pm 2.66e+06}$	$6.76e+10 \pm 6.23e+09$	+	$1.05e+11 \pm 1.53e+10$	+	$1.97e+08 \pm 1.41e+07$	+
f15	$\mathbf{1.33e+04 \pm 2.40e+03}$	$1.91e+04 \pm 4.61e+02$	+	$1.99e+04 \pm 3.09e+02$	+	$1.58e+04 \pm 5.33e+02$	+
f16	$\mathbf{1.60e+02 \pm 1.14e+02}$	$4.22e+02 \pm 6.53e-01$	+	$4.23e+02 \pm 5.48e-01$	+	$3.97e+02 \pm 2.63e-01$	+
f17	$\mathbf{6.31e+04 \pm 8.74e+03}$	$1.05e+07 \pm 9.29e+05$	+	$1.46e+07 \pm 1.84e+06$	+	$4.84e+05 \pm 6.56e+04$	+
f18	$\mathbf{3.88e+03 \pm 4.66e+03}$	$2.47e+12 \pm 1.65e+11$	+	$4.65e+12 \pm 2.42e+11$	+	$1.79e+04 \pm 7.91e+03$	+
f19	$\mathbf{1.16e+06 \pm 8.91e+04}$	$1.10e+07 \pm 1.21e+06$	+	$3.54e+07 \pm 5.03e+06$	+	$5.57e+07 \pm 1.41e+07$	+
f20	$1.20e+03 \pm 1.93e+02$	$2.81e+12 \pm 1.65e+11$	+	$5.25e+12 \pm 2.10e+11$	+	$\mathbf{1.19e+03 \pm 3.07e+02}$	=

Table 8 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with complex algorithms on the CEC 2010 benchmark in 1000 dimensions

	S-3SOME	G-CMAES		SADE		CCPSO2	
f1	$6.03e-04 \pm 4.71e-04$	$4.75e+05 \pm 4.02e+04$	+	$2.52e+07 \pm 4.74e+07$	+	$\mathbf{5.12e-05 \pm 7.94e-05}$	-
f2	$\mathbf{3.99e-02 \pm 9.13e-03}$	$1.02e+04 \pm 4.45e+02$	+	$5.70e+03 \pm 3.34e+02$	+	$1.33e+02 \pm 1.26e+02$	+
f3	$7.56e-03 \pm 6.83e-04$	$1.99e+01 \pm 1.08e-02$	+	$1.90e+01 \pm 2.17e-01$	+	$\mathbf{2.13e-06 \pm 3.52e-06}$	-
f4	$2.08e+13 \pm 6.90e+12$	$\mathbf{1.54e+11 \pm 2.13e+10}$	-	$3.44e+12 \pm 2.41e+12$	-	$3.80e+12 \pm 2.29e+12$	-
f5	$4.27e+08 \pm 1.16e+08$	$6.73e+08 \pm 9.20e+07$	+	$\mathbf{1.05e+08 \pm 1.78e+07}$	-	$4.08e+08 \pm 1.10e+08$	=
f6	$1.54e+07 \pm 5.39e+06$	$1.98e+07 \pm 6.46e+04$	+	$\mathbf{5.52e+05 \pm 8.13e+05}$	-	$1.67e+07 \pm 4.94e+06$	=
f7	$1.07e+10 \pm 2.83e+09$	$\mathbf{5.46e+06 \pm 3.41e+05}$	-	$2.35e+08 \pm 4.41e+08$	-	$1.33e+10 \pm 1.33e+10$	=
f8	$1.78e+09 \pm 2.68e+09$	$\mathbf{5.62e+06 \pm 1.88e+05}$	-	$8.28e+07 \pm 3.25e+07$	-	$7.40e+07 \pm 4.88e+07$	-
f9	$3.54e+08 \pm 7.89e+07$	$\mathbf{5.04e+05 \pm 4.39e+04}$	-	$3.90e+08 \pm 3.22e+08$	-	$8.51e+07 \pm 1.25e+07$	-
f10	$5.12e+03 \pm 2.63e+02$	$1.04e+04 \pm 3.99e+02$	+	$6.37e+03 \pm 2.69e+02$	+	$\mathbf{4.55e+03 \pm 2.93e+02}$	-
f11	$\mathbf{1.97e+02 \pm 4.27e+00}$	$2.18e+02 \pm 2.14e-01$	+	$2.05e+02 \pm 3.33e+00$	+	$2.01e+02 \pm 6.51e+00$	=
f12	$8.74e+04 \pm 2.12e+04$	$\mathbf{1.04e-12 \pm 8.83e-14}$	-	$4.85e+05 \pm 1.82e+05$	+	$1.38e+05 \pm 1.27e+05$	+
f13	$5.61e+05 \pm 7.34e+05$	$\mathbf{2.20e+02 \pm 3.37e+02}$	-	$1.32e+07 \pm 3.61e+07$	-	$1.36e+03 \pm 3.96e+02$	-
f14	$8.79e+07 \pm 2.66e+06$	$\mathbf{5.52e+05 \pm 5.43e+04}$	-	$6.30e+08 \pm 2.98e+08$	+	$2.93e+08 \pm 5.53e+07$	+
f15	$1.33e+04 \pm 2.40e+03$	$1.03e+04 \pm 5.24e+02$	-	$\mathbf{6.58e+03 \pm 4.22e+02}$	-	$9.27e+03 \pm 6.90e+02$	-
f16	$\mathbf{1.60e+02 \pm 1.14e+02}$	$3.97e+02 \pm 3.14e-01$	+	$3.83e+02 \pm 1.82e+00$	+	$3.94e+02 \pm 1.40e+00$	+
f17	$6.31e+04 \pm 8.74e+03$	$\mathbf{1.97e-11 \pm 8.02e-12}$	-	$9.23e+05 \pm 1.67e+05$	+	$2.68e+05 \pm 1.38e+05$	+
f18	$3.88e+03 \pm 4.66e+03$	$\mathbf{4.47e+02 \pm 3.88e+02}$	-	$1.98e+09 \pm 4.20e+09$	+	$8.02e+03 \pm 6.53e+03$	+
f19	$1.16e+06 \pm 8.91e+04$	$\mathbf{1.48e+04 \pm 3.07e+03}$	-	$2.69e+06 \pm 1.96e+05$	+	$4.38e+06 \pm 7.34e+06$	=
f20	$1.20e+03 \pm 1.93e+02$	$\mathbf{8.33e+02 \pm 6.48e+01}$	-	$2.29e+09 \pm 2.93e+09$	+	$1.52e+03 \pm 1.36e+02$	+

Figure 4 Average fitness trend of memory-saving algorithms on f10 from CEC 2010 in 1000 dimensions.

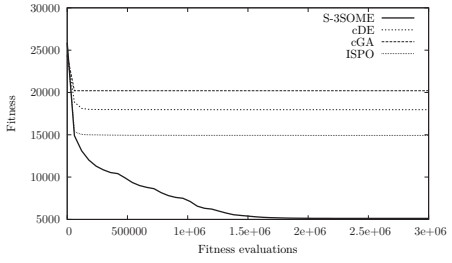


Figure 5 Average fitness trend of complex algorithms on f11 from CEC 2010 in 1000 dimensions.

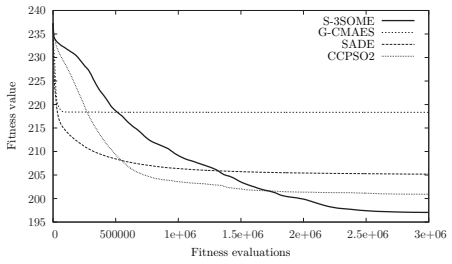


Table 9 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with 3SOME and S-3SOME on the CEC 2010 benchmark in 1000 dimensions

	S-3SOME	3SOME	
f1	$\mathbf{6.03e-04 \pm 4.71e-04}$	$2.15e-02 \pm 6.88e-02$	+
f2	$\mathbf{3.99e-02 \pm 9.13e-03}$	$1.36e+01 \pm 1.91e+01$	+
f3	$\mathbf{7.56e-03 \pm 6.83e-04}$	$4.81e-01 \pm 4.94e-01$	+
f4	$2.08e+13 \pm 6.90e+12$	$\mathbf{8.08e+12 \pm 3.04e+12}$	-
f5	$\mathbf{4.27e+08 \pm 1.16e+08}$	$7.26e+08 \pm 1.38e+08$	+
f6	$\mathbf{1.54e+07 \pm 5.39e+06}$	$1.98e+07 \pm 9.20e+04$	+
f7	$1.07e+10 \pm 2.83e+09$	$\mathbf{1.48e+09 \pm 3.73e+08}$	-
f8	$1.78e+09 \pm 2.68e+09$	$\mathbf{8.68e+08 \pm 2.62e+09}$	-
f9	$\mathbf{3.54e+08 \pm 7.89e+07}$	$4.24e+08 \pm 6.35e+07$	+
f10	$\mathbf{5.12e+03 \pm 2.63e+02}$	$6.75e+03 \pm 3.69e+02$	+
f11	$\mathbf{1.97e+02 \pm 4.27e+00}$	$1.99e+02 \pm 5.28e-01$	+
f12	$\mathbf{8.74e+04 \pm 2.12e+04}$	$1.57e+05 \pm 7.74e+04$	+
f13	$5.61e+05 \pm 7.34e+05$	$\mathbf{1.48e+04 \pm 6.61e+03}$	-
f14	$\mathbf{8.79e+07 \pm 2.66e+06}$	$1.12e+08 \pm 2.13e+07$	+
f15	$\mathbf{1.33e+04 \pm 2.40e+03}$	$1.37e+04 \pm 6.36e+02$	=
f16	$\mathbf{1.60e+02 \pm 1.14e+02}$	$3.81e+02 \pm 6.13e+01$	+
f17	$\mathbf{6.31e+04 \pm 8.74e+03}$	$2.78e+05 \pm 2.26e+05$	+
f18	$\mathbf{3.88e+03 \pm 4.66e+03}$	$2.27e+04 \pm 1.47e+04$	+
f19	$1.16e+06 \pm 8.91e+04$	$\mathbf{1.44e+05 \pm 1.83e+04}$	-
f20	$1.20e+03 \pm 1.93e+02$	$\mathbf{1.14e+03 \pm 1.51e+02}$	=

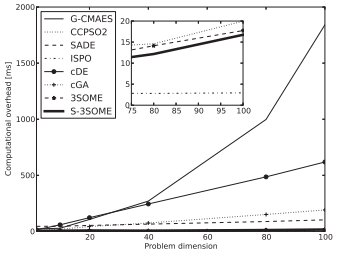
function), with a budget of 10000 fitness evaluations. Due to the scalability of the sphere function, we tested all the algorithms for solving the same problem in 2, 10, 20, 40, 80 and 100 dimensions. For each algorithm and problem dimensionality, 30 independent runs were performed, and for each run the computational overhead, i.e., the time employed by the algorithm to perform its operations, was computed subtracting the calculation time of 10000 fitness evaluations (at the selected dimensionality) from the total execution time of the run. The average computational overhead, in dependence on the dimensionality of the problem, is shown for all the algorithms in Figure 6 and Table 10.

It can be observed that, among the selected algorithms, ISPO shows the minimum overhead, which does not even depend on the problem dimensionality. On the other hand, the other two memory-saving algorithms considered, namely cGA and cDE, are characterised by an overhead which grows linearly with the dimensions, being cDE more expensive than cGA (due to the multiple sampling mechanism performed at each step of the algorithm).

Amongst the complex algorithms, G-CMAES is by far the most expensive in terms of computational overhead: it's easy to prove that, due to the covariance matrix update mechanism, its complexity grows with the square of the problem dimensionality ($\mathcal{O}\left(n^2\right)$). On the other hand, the complexity of SADE and CCPSO2 grows linearly with the dimensionality, being CCPSO2 computationally much cheaper than SADE.

Finally, 3SOME and S-3SOME are also characterised by a linear complexity, being S-3SOME slightly cheaper than the original 3SOME framework. However, the most important observation is that S-3SOME is the second computationally cheapest algorithm of the entire experimental setup, after ISPO (whose performance, except for 10-dimensional problems, is generally worse, as we have seen before). Together with its low memory requirement and its respectable performance, this fact makes the S-3SOME scheme an appealing candidate for real-time optimisation on embedded systems.

Figure 6 Overhead of the selected algorithms over 10000 fitness evaluations of f1 from BBOB 2010.



3.4 Dynamic behaviour of memory-saving algorithms

We here conclude the discussion of our experiments with some remarks on the dynamic behaviour of S-3SOME compared to the other memory-saving algorithms in the experimental setup described above, namely cDE, cGA and ISPO. Our goal is to analyse how those algorithms perform when a limited budget is allotted: this can be the case, for example, of a real-time optimisation process performed on board of an embedded system. In this context a large budget is often not available and the optimisation must be completed within a reasonably

Table 10 Average overhead of the selected algorithms (ms) over 10000 fitness evaluations of f1 from BBOB 2010.

Algorithm	Problem Dimension					
	2	10	20	40	80	100
G-CMAES	22.60	28.27	106.83	270.53	997.90	1841.27
CCPSO2	10.60	10.27	10.13	12.30	14.60	19.97
SADE	45.37	46.50	53.30	65.07	88.03	103.07
ISPO	3.63	2.73	3.30	2.97	2.80	2.93
cDE	14.53	58.70	122.60	244.70	486.17	618.73
cGA	6.57	18.93	43.43	75.03	151.10	191.43
3SOME	10.00	6.50	10.00	6.10	14.13	17.73
S-3SOME	4.50	6.40	8.13	6.33	12.17	16.73

short amount of time. Due to hardware limitations and a higher algorithmic complexity, modern population-based meta-heuristics cannot be used, in general, in this kind of application. For this reason we purposely focus this analysis only on memory-saving algorithms.

Tables 11 and 12 show the final values and Wilcoxon Rank-Sum Test results obtained, at different budget levels, by the four aforementioned algorithms on a selected subset of test functions taken from the BBOB 2010 and CEC 2010 testbeds. More specifically, Table 11 shows the results obtained on f5, f8 and f19 from BBOB 2010 in 40 dimensions, while Table 12 shows the results obtained on f3, f6 and f19 from CEC 2010 in 1000 dimensions. These functions have been chosen looking at the benchmark definitions (Hansen el al., 2010) and (Tang el al., 2010) and selecting, for each of the two testbeds, a separable problem (f5 from BBOB 2010 and f3 from CEC 2010), a moderately non-separable problem (f8 from BBOB 2010 and f6 from CEC 2010), and a fully non-separable problem (f19 from both BBOB 2010 and CEC 2010). Although this choice might not be completely exhaustive, it gives at least some clues about the behaviour of the selected algorithms on different classes of problems.

In order to highlight the converge trend of the four algorithms, three stopping criteria were used, namely $50 \cdot n$, $500 \cdot n$ and $5000 \cdot n$, where n is usual the problem dimensionality. Intermediate results were collected at each budget level and statistic analyses were performed accordingly.

From Table 11 it can be seen that, while ISPO, due to its variable-wise perturbation, is extremely good at handling separable medium scale functions (f5), on (at least partially) non-separable problems S-3SOME is able to improve upon the initial solution faster than the other memory-saving algorithms, and to better final values. This trend is confirmed on large scale problems, see Table 12, where the performance of S-3SOME is improved also on separable functions (f3). This result confirms how S-3SOME is an excellent candidate for real-time optimisation, where a good solution has to be found as quickly as possible, especially on large scale problems.

4 Conclusion

In this paper a simple memetic scheme for numerical optimisation has been proposed. Combining a random global search, a stochastic local search which progressively shrinks the search radius, and a deterministic local search, the resulting algorithm, called Shrinking Three Stage Optimal Memetic Exploration (S-3SOME), has proven to be extremely flexible at handling efficiently a set of diverse problems at various dimensionality levels. Numerical results obtained on an extensive experimental setup composed of two entire benchmarks in 10, 40, 100 and 1000 dimensions showed that, despite much smaller requirements in terms of computational resources (memory and CPU operations), the proposed approach is highly competitive with some of the state-of-the-art global optimisers, particularly in high dimensions. In addition to that, from the comparison with other algorithms requiring limited computational resources, it emerged that S-3SOME is generally able to obtain the best results in the shortest amount of time (fitness evaluations). Thus the S-3SOME scheme can be considered a suitable solution for solving especially (semi) large scale problems in contexts where limited hardware resources are available and a rapid (even real-time) optimised response is needed.

Acknowledgments

This research is supported by the Academy of Finland, Akatemiatutkija 130600 and Tutkijatohtori 140487. 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 paper is a revised and expanded version of a paper entitled 'Shrinking Three Stage Optimal Memetic Exploration' presented at the 5th International Conference on Bioinspired Optimization Methods and their Applications, 24–25 May 2012, Bohinj, Slovenia.

Table 11 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with memory-saving algorithms on f5, f8 and f19 from BBOB 2010 in 40 dimensions and multiple budget levels.

	stop	S-3SOME	cDE		cGA		ISPO	
f5	$50 \cdot n$	$6.57e+01 \pm 1.16e+01$	$8.19e+01 \pm 1.66e+01$	+	$1.89e+02 \pm 1.49e+01$	+	$\mathbf{2.85e+01 \pm 4.14e+01}$	-
	$500 \cdot n$	$3.42e-06 \pm 1.78e-06$	$6.61e-02 \pm 1.77e-01$	+	$1.51e-01 \pm 8.13e-01$	+	$\mathbf{-3.55e-14 \pm 0.00e+00}$	-
	$5000 \cdot n$	$2.45e-12 \pm 9.36e-13$	$1.28e-02 \pm 6.89e-02$	+	$2.19e-04 \pm 7.91e-04$	+	$\mathbf{-3.55e-14 \pm 0.00e+00}$	-
f8	$50 \cdot n$	$\mathbf{8.87e+02 \pm 3.08e+02}$	$4.08e+03 \pm 1.64e+03$	+	$6.46e+04 \pm 2.05e+04$	+	$4.21e+04 \pm 6.84e+04$	+
	$500 \cdot n$	$\mathbf{5.64e+01 \pm 3.46e+01}$	$9.24e+01 \pm 8.25e+01$	=	$5.96e+02 \pm 3.27e+02$	+	$1.30e+02 \pm 4.69e+01$	+
	$5000 \cdot n$	$\mathbf{3.32e-01 \pm 5.31e-01}$	$6.68e+01 \pm 6.88e+01$	+	$5.00e+02 \pm 2.75e+02$	+	$3.96e+01 \pm 3.73e+01$	+
f19	$50 \cdot n$	$\mathbf{1.04e+01 \pm 2.22e+00}$	$1.17e+01 \pm 1.74e+00$	+	$1.61e+01 \pm 2.16e+00$	+	$5.17e+01 \pm 1.61e+01$	+
	$500 \cdot n$	$6.91e+00 \pm 2.38e+00$	$7.61e+00 \pm 1.98e+00$	=	$\mathbf{6.88e+00 \pm 1.74e+00}$	=	$4.84e+01 \pm 1.46e+01$	+
	$5000 \cdot n$	$\mathbf{6.13e+00 \pm 2.38e+00}$	$7.26e+00 \pm 1.97e+00$	+	$6.83e+00 \pm 1.74e+00$	=	$4.64e+01 \pm 1.29e+01$	+

Table 12 Average Fitness ± Standard Deviation and Wilcoxon Rank-Sum Test (reference = S-3SOME) obtained with memory-saving algorithms on f3, f6 and f19 from CEC 2010 in 1000 dimensions and multiple budget levels.

	stop	S-3SOME	cDE		cGA		ISPO	
f3	$50 \times n$	$\mathbf{1.90e+01 \pm 9.58e-02}$	$2.10e+01 \pm 1.65e-01$	+	$2.12e+01 \pm 4.14e-02$	+	$1.99e+01 \pm 1.54e-02$	+
	$500 \times n$	$\mathbf{9.53e+00 \pm 4.03e-01}$	$2.08e+01 \pm 4.24e-02$	+	$2.12e+01 \pm 4.14e-02$	+	$1.99e+01 \pm 1.41e-02$	+
	$5000 \times n$	$\mathbf{7.56e-03 \pm 6.83e-04}$	$2.08e+01 \pm 4.19e-02$	+	$2.12e+01 \pm 4.14e-02$	+	$1.99e+01 \pm 1.38e-02$	+
f6	$50 \times n$	$1.62e+07 \pm 5.32e+06$	$1.76e+07 \pm 2.19e+06$	=	$\mathbf{1.30e+07 \pm 3.27e+06}$	-	$2.01e+07 \pm 8.34e+04$	=
	$500 \times n$	$1.57e+07 \pm 5.39e+06$	$1.76e+07 \pm 2.17e+06$	=	$\mathbf{1.30e+07 \pm 3.27e+06}$	=	$1.98e+07 \pm 4.88e+04$	=
	$5000 \times n$	$1.54e+07 \pm 5.39e+06$	$1.76e+07 \pm 2.17e+06$	=	$\mathbf{1.30e+07 \pm 3.27e+06}$	=	$1.98e+07 \pm 4.88e+04$	+
f19	$50 \times n$	$\mathbf{5.79e+06 \pm 3.93e+05}$	$5.74e+07 \pm 1.19e+08$	+	$3.54e+07 \pm 5.03e+06$	+	$8.74e+08 \pm 4.93e+08$	+
	$500 \times n$	$\mathbf{3.10e+06 \pm 1.32e+05}$	$1.10e+07 \pm 1.21e+06$	+	$3.54e+07 \pm 5.03e+06$	+	$1.75e+08 \pm 5.49e+07$	+
	$5000 \times n$	$\mathbf{1.16e+06 \pm 8.91e+04}$	$1.10e+07 \pm 1.21e+06$	+	$3.54e+07 \pm 5.03e+06$	+	$5.57e+07 \pm 1.41e+07$	+

References

Auger, A. and Hansen, N. (2005) 'A restart CMA evolution strategy with increasing population size', *Proceedings of the IEEE Congress on Evolutionary Computation*, 2–4 September, Edinburgh, UK, pp.1769–1776.

Brest, J., Greiner, S., Bošković, B., Mernik, M. and Žumer, V. (2006) 'Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems', *IEEE T. Evolut. Comput.*, Vol. 10, No. 6, pp.646–657.

Hooke, R. and Jeeves, T.A. (1961) 'Direct search solution of numerical and statistical problems', *J. ACM*, Vol. 8, No. 2, pp.212–229.

Iacca, G., Neri, F., Mininno, E., Ong, Y.S. and Lim, M.H. (2012) 'Ockham's razor in memetic computing: Three stage optimal memetic exploration', *Inform. Sciences*, Vol. 188, No. 1, pp.17-43.

Li, X. and Yao, X. (2012) 'Cooperatively coevolving particle swarms for large scale optimization', *IEEE T. Evolut. Comput.*, Vol. 16, No. 2, pp.210–224.

Liang, J.J., Qin, A.K., Suganthan, P.N. and Baskar, S. (2006) 'Comprehensive learning particle swarm optimizer for global optimization of multimodal functions', *IEEE T. Evolut. Comput.*, Vol. 10, No. 3, pp.281–295.

Hansen, N., Finck, S., Ros, R. and Auger, A. (2010) 'Real-parameter black-box optimization benchmarking 2010: Noiseless functions definitions', Technical Report RR-6829, INRIA.

Mininno, E., Cupertino, F. and Naso, D. (2008) 'Real-valued compactgenetic algorithms for embedded microcontroller mptimization', *IEEE T. Evolut. Comput.*, Vol. 12, No. 2, pp.203–219.

Mininno, E., Neri, F., Cupertino, F. and Naso, D. (2011) 'Compact differential evolution', *IEEE T. Evolut. Comput.*, Vol. 15, No. 1, pp.32–54.

Neri, F., Cotta, C. and Moscato, P. (2012) *Handbook of Memetic Algorithms*, Vol. 379 of *Studies in Computational Intelligence*, Springer.

Neri, F., Iacca, G. and Mininno, E. (2011) 'Disturbed exploitation compact differential evolution for limited memory optimization problems', *Inform. Sciences*, Vol. 181, No. 12, pp.2469–2487.

Peng, F., Tang, K., Chen, G. and Yao, X. (2010) 'Population-based algorithm portfolios for numerical optimization', *IEEE T. Evolut. Comput.*, Vol. 14, No. 5, pp.782–800.

Poikolainen, I., Iacca, G., Neri, F., Mininno, E. and Weber, M. (2012) 'Shrinking three stage optimal memetic exploration', *Proceedings of the Fifth International Conference on Bioinspired Optimization Methods and their Applications*, 24–25 May, Bohinj, Slovenia, pp.61–74.

Price, K.V., Storn, R.M. and Lampinen, J.A. (2005) *Differential Evolution: A Practical Approach to Global Optimization*, Springer.

Qin, A.K., Huang, V.L. and Suganthan, P.N. (2009) 'Differential evolution algorithm with strategy adaptation for global numerical optimization', *IEEE T. Evolut. Comput.*, Vol. 13, No. 2, pp.398–417.

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

Tseng, L.Y. and Chen, C. (2008) 'Multiple trajectory search for large scale global optimization', *Proceedings of the IEEE Congress on Evolutionary Computation*, 1–6 June, Hong Kong, China, pp.3052–3059.

Vrugt, J.A., Robinson, B.A. and Hyman, J.M. (2009) 'Self-adaptive multimethod search for global optimization in real-parameter spaces', *IEEE T. Evolut. Comput.*, Vol. 13, No. 2, pp.243–259.

Wilcoxon, F. (1945) 'Individual comparisons by ranking methods', *Biometrics Bull.*, Vol. 1, No. 6, pp.80–83.

Wolpert, D.H. and Macready, W.G. (1997) 'No free lunch theorems for optimization', *IEEE T. Evolut. Comput.*, Vol. 1, No. 1, pp.67–82.

Zhou, J., Ji, Z. and Shen, L. (2008) 'Simplified intelligence single particle optimization based neural network for digit recognition', *Proceedings of the Chinese Conference on Pattern Recognition*, 22–24 October, Beijing, China, pp.1–5.

PV

MICRO-DIFFERENTIAL EVOLUTION WITH EXTRA MOVES ALONG THE AXES

by

F. Caraffini, F. Neri, I. 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

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, al 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 or 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 arount 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 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 III 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}	$\mathbf{1.49e+04 \pm 1.10e+03}$	$1.99e+08 \pm 7.66e+08$	+

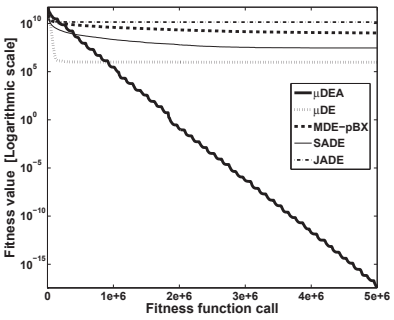


Fig. 3. Performance trend for f_1 of CEC2010 [40] in 1000 dimensions.

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 RIS is shown in parenthesis. Moreover, it is indicated whether the null-hypothesis (that the two algorithms have indistinguishable performances) is "Rejected", i.e. RIS 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 be preferable to the large ones.

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 of the computer network.

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 Cyberneticspart 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. Ošmera, 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/.

PVI

DIFFERENTIAL EVOLUTION WITH CONCURRENT FITNESS BASED LOCAL SEARCH

by

I. Poikolainen, F. Neri 2013

IEEE Congress on Evolutionary Computation (CEC), pages 384-391

Differential Evolution with Concurrent Fitness Based Local Search

Ilpo Poikolainen
Department of Mathematical Information Technology
University of Jyväskylä
Jyväskylä, Finland
Email: ilpo.poikolainen@jyu.fi

Ferrante Neri
Department of Mathematical Information Technology
University of Jyväskylä
Jyväskylä, Finland
Email: ferrante.neri@jyu.fi

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

***Abstract*—This paper proposes a novel implementation of memetic structure for continuous optimization problems. The proposed algorithm, namely Differential Evolution with Concurrent Fitness Based Local Search (DEcfbLS), enhances the DE performance by including a local search concurrently applied on multiple individuals of the population. The selection of the individuals undergoing local search is based on a fitness-based adaptive rule. The most promising individuals are rewarded with a local search operator that moves along the axes and complements the normal search moves of DE structure. The application of local search is performed with a shallow termination rule. This design has been performed in order to overcome the limitations within the search logic on the original DE algorithm. The proposed algorithm has been tested on various problems in multiple dimensions. Numerical results show that the proposed algorithm is promising candidate to take part to competition on Real-Parameter Single Objective Optimization at CEC-2013. A comparison against modern meta-heuristics confirms that the proposed algorithm robustly displays a good performance on the testbed under consideration.**

I. Introduction

Differential Evolution (DE), see [1], is a popular and reliable optimizer that shows a robust performance across various continuous optimization problems. DE employs a population of solutions and generates its offspring by means of the scaled difference of two (or more) individuals of the population. The survivor selection is made between the parent individual and offspring by following the so called one-to-one spawning: the offspring directly replaces the parent that generated it when the offspring outperforms the parent. Thanks to its simplicity features and performance, after its original definition, DE has become popular among both researches and practitioners. Some examples of successful DE applications can be found in [1] and [2]. Practical applications include multisensor fusion problem, see [3], aerodynamic design, see [4], filter design, see [5] and many other engineering applications. Survey papers on DE framework and its variants are reported in [6] and [7]. The reasons behind the DE success are listed and analysed by the following points.

1) DE has rather simple implementation with respect to other computational intelligence optimization algorithms while still having very high performance on complex fitness landscapes including uni-modal, multi-modal, separable, non-separable etc, see [8].
2) Control parameters of DE are relatively few (only three classical DE), see [9]. Although a proper selection of these parameters is essential for guaranteeing a good performance on a given problem, a reasonably quick parameter tuning appears to often allow a good performance, see [10]. However, due to the dynamic nature of evolution/optimization, modern DE schemes often propose a dynamic/adaptive parameter setting or employment of multiple search rules. Some popular examples are given in the self-adaptive (and randomized) DE version proposed in [11], and in the controlled parameter randomization proposed in [12].
3) The computational complexity of DE is relatively low, thus making DE schemes flexibly applicable in large scale optimization problem. This advantage is especially evident when we consider the features of algorithms based on a covariance matrix adaptation, see e.g. [13]. Although the latter algorithms are very efficient (thanks to their theoretical foundations) in low dimensions, they might lead to an unacceptable computational cost (due to calculations involving square matrices with size equal to number of variables) for large scale problems.

Although DE is capable at displaying a high performance, this algorithm is characterized by some limitations, see [7]. If the algorithm does not manage to generate an offspring which outperforms the corresponding parent solution, the search is likely to be jeopardized and result into the undesired stagnation condition. If the search is repeated over and over again with similar step sizes without succeeding to improve upon the candidate solutions, DE can display a poor performance, see [14]. As shown in the analysis reported in [7], DE scheme is characterized by a limited amount of search moves. Some countermeasure to mitigate this issue include a randomization of parameters, see e.g. [12], [11], and [15], progressive population size reduction, see [16], and explicit inclusion of local search operators, see e.g. [17], [18], and [19].

Modern DE-based algorithms can thus be divided into the two following categories, see [7]:

1) DE integrating an extra component This class includes those algorithms that use DE as an evolutionary framework assisted by other algorithmic components, e.g., local searchers or extra operators (see [20] [17] and [19]). The algorithms belonging to this category can be decomposed as DE framework and additional components
2) Modified strutures of DE This class includes those algorithms which modify the DE structure, search logic, selection etc. Some examples of such modifications are given in articles [11], [21], and [22]

In this paper we use integration of local searcher as extra component with DE to overcome possible problems of stagnation and to improve exploitative capabilities of DE algorithm. The proposed algorithm, at first, samples a population of candidate solutions within a decision space. Then, each population individual is refined with a a local search by using a small local budget. Small local budget appears very important for achieving some initial improvements without excessively biasing the search towards local optima. After this initialization, the main components a DE framework cooperates with a local search to enhance upon the performance of the candidate solutions. While DE is a population based component, deterministic Local Search (LS) is a single point optimizer which is performed on multiple population members on each activation.

The remaining of this paper is organized as follows. Section II describes the algorithmic structure of DEcfbLS and its components. Section III shows the experimental setup and numerical results of the study. Section IV gives the conclusions of this work.

II. Differential Evolution with Concurrent Fitness Based Local Search

In order to clarify the notation used in this paper, we refer to the minimization problem of an objective function $f(x)$, where a candidate solution x is a vector of n design variables in decision space D. This section describes the algorithmic framework of DEcfbLS by analysing the three different components that compose it:

1) Local Search Initialization
2) Differential Evolution
3) Deterministic Local Search

A. Local Search Initialization

At the beginning of optimization process, the population is randomly initialized (with uniform distribution) within the search space D. In our implementation we have set population size NP equal to 30 individuals. After the initialization, we perform a LS with very small local budget (4 iterations) over all the individuals of the initial population, see Algorithm 2.

B. Differential Evolution

After initialization and refining, DE is activated over the individuals of the population, see e.g. [23] and [1]. The working principles of this framework are here described. At each generation, for each solution x_i, three other solutions x_t, x_s and x_t are randomly selected from population. Then, a provisional offspring x_o is generated by mutation:

$$x_o = x_t + F(x_r - x_s), \tag{1}$$

where F is a parameter namely scale factor. Scale factor is a positive value tat usually is selected to be greater than 1. In our implementation we have set the scale factor F to be equal to 0.7. The mutation scheme shown in ed. (1) is known as DE/rand/1. Many other mutation schemes have been proposed in the literature as well as self-adaptive schemes that make use of multiple mutation strategies, for example see [11] and [6].

When the provisional offspring has been generated by mutation, the so-called exponential crossover is applied to combine the genes of the parent solution x_i and provisional offspring x_o. At first a copy of the parent solution x_i is performed. Then, one design variable is chosen at random and copied from the provisional offspring (also referred as mutant) to the corresponding position of x_i (its copy more precisely) to make sure atleast one parameter is exchanged. The source of subsequent trial parameters is determined by comparing predefined crossover rate Cr to uniformly distributed random number between 0 and 1 that is generated anew for each parameter, i.e, $rand_j(0,1)$. As long as $rand_j(0.1) \leq Cr$, parameters continue to be taken from mutant x_o. As soon as $rand_j(0,1) > Cr$, the copy process is interrupted. Thus, the current and all remaining parameters of the final offspring x_{off} are those from the parent solution x_i. The pseudo-code of the exponential crossover is shown in Algorithm 1.

Algorithm 1 Exponential crossover

```
x_off = x_i
generate j = round(n · rand(0,1))
x_off[j] = x_o[j]
k = 1
while rand(0,1) ≤ Cr AND k < n do
    x_off[j] = x_o[j]
    j = j + 1
    if j == n then
        j = 1
    end if
    k = k + 1
end while
```

In other words, this crossover operator generates offspring composed of the parent x_i and contains, within it, a section of the chromosome of the mutant vector x_o. According to its original definition, see e.g. [1], for a fixed Cr value the exploration feature of the crossover operator is dependant on the dimensionality of the problem. For example, if Cr is prearranged in a low dimensional problem, the offspring is composed mainly of the mutant vector (provisional offspring), while for a high dimensional problem, the offspring is mainly composed of the parent. In this study, we propose to slightly modify the definition of exponential crossover by fixing, instead of Cr, the approximate proportion of mutant genes within the offspring. Let us define this proportion, namely inheritance factor, as

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

where n_e is the number of mutant genes we expect to copy from x_o into x_{off} in addition to the gene deterministically copied. In order to achieve that on average n_e are copied into the offspring we need to impose 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 in the following way, see [24] for details:

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

C. Deterministic local search

The deterministic local search is a single solution algorithm that attempts to exploit promising solutions by performing steps along the axis. This LS has been effectively used in [25], [26], and [27]. The search logic is similar to that of Hooke-Jeeves algorithm, see [28]. The purpose of this search algorithm is to exploit promising basins of attraction. The algorithm searches each design variable by performing steps along the axis and if the solution does not improve the search step-size is halved for the following iteration. More precisely, for each dimension j, the algorithm samples $x[j] - \rho$, where $x[j]$ is the j^{th} design variable of the selected point x on which LS is performed. If the perturbed solution is better than x, it replaces the old solution and moves to the next dimension $j+1$. If there is no out-performance, the search looks towards the opposite direction with a step-size $\rho/2$ before moving towards the following dimension. After having gone through each dimension if there was no improvement the step-size ρ is halved before subsequent iteration. For the sake of clarity, the pseudo-code of the proposed LS is given in Algorithm 2.

Algorithm 2 Determinisitic local search

```
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 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
```

D. Coordination of the local search

The functioning of this LS, albeit efficient, is characterized by a steepest descent pivot rule and thus requires up to $2 \times n$ Fitness Evaluations (FEs) per iteration. In order to prevent an excess use of LS budget, especially in the early stages of optimization process, each time LS is activated we allocate a local budget of $Iter = 40$ iterations (how many times step size is halved at maximum) for LS. In addition, both algorithmic components are given a similar budget and are alternatively activated. Both DE and LS component are given equal budget of FEs, thus we can activate LS K times, where K is calculated as follows:

$$K = \frac{maxFEs}{(LSFE * NP * 2)}, \tag{5}$$

where $LSFE$ is fitness evaluations used at most by LS and NP is population size. These K breakpoints are evenly spread along optimization process:

$$FEbp = \frac{maxFEs}{K} \tag{6}$$

and LS is activated every $FEbp$ fitness evaluations. It can be observed that if the LS component improves upon the starting solution it uses less FEs than when it fails to succeed also not all population members get selected for LS. Thus, in general, the number of FEs used by DE is greater or equal than that used by LS used over the whole optimization process.

At each breakpoint, the LS is performed on multiple individuals selected from the population. This selection is based on the individual performance (fitness of individuals) with respect to the average performance of the population. More specifically, the LS is performed over all those individuals whose fitness is better than the average fitness over all the individuals of the population. For the sake of clarity pseudo-code for selection is shown in Algorithm 3.

Algorithm 3 Selection for LS

```
for i = 1 : NP do
    if f(x_i) ≤ f_avg then
        activate LS on x_i
    end if
end for
```

The flowchart of the entire DEcfbLS is presented in Fig. 1 while the pseudocode displaying the implementation details of the proposed algorithm is given in Algorithm 4.

III. Numerical Results

In this study, the proposed algorithm has been run with following parameters $Np = 30$, $F = 0.7$, $\alpha_e = 0.5$, $Iter = 40$, and $\rho = 0.4$ times the width of decision space D along axis.

All experiments have been performed on CEC2013 test suite [29] in 10, 30 and 50 dimensions over 28 test problems. For each problem, 51 runs have been performed for $10000 \times n$ fitness evaluations, where n is the dimensionality of the problem. For each test problem and each algorithm, the best, worst, median, mean and standard deviation over the 51 runs has been computed as function error value ($f(x) - f^*(x)$), where $f^*(x)$ is the global minimum of function. Also algorithm complexity has been considered by calculating three different runtime values $T0$, $T1$ and $T2$ as follows:

- $T0$ is calculated by running test program shown in table I.
- $T1$ is calculated by evaluating computation time of function 14 for 200000 evaluations of certain dimension D.

Algorithm 4 DEcfbLS

generate randomly NP individuals of the initial population and compute their fitness values
for $i = 1 : NP$ **do**
  apply LS on i with local budget of 4 iterations.
end for
while iterations is less than predefined budget **do**
  for $i = 1 : NP$ **do**
    select three individuals x_r, x_s, and x_t
    compute mutant individual $x_o = x_t + F(x_r - x_s)$
    compute exponential crossover between x_i and x_o thus generating x_{off}
    if $f(x_{off}) \leq f(x_i)$ **then**
      save index
    end if
  end for
  perform survivor selection where proper on the basis of saved indexes
  compute $FEbp$ according to eq. (6)
  if iterations is greater or equal than the breakpoint $FEbp$ **then**
    compute populations fitness average f_{avg}
    for $i = 1 : NP$ **do**
      if $f(i) < f_{avg}$ **then**
        apply LS on i with local budget of 40 iterations.
      end if
    end for
    update fitness evaluations breakpoint
  end if
end while

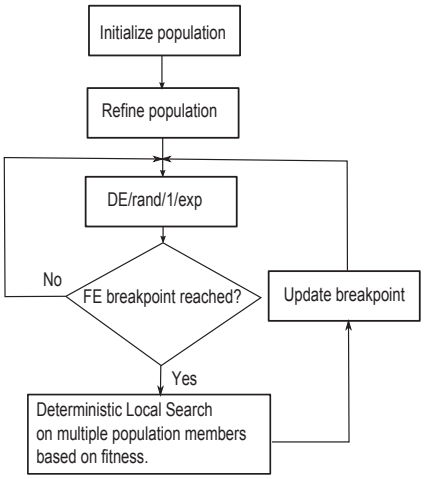


Fig. 1. Flowchart

- $T2$ is calculated by running function 14 and computing total computation time with 200000 evalutations of the same D dimensional benchmark function 14.

The procedure for the computation of $T0$ is reported in Algorithm 5.

The complexity of algorithm is reflected by $\hat{T}2, T1, T0$ and $(\hat{T}2 - T1)/T0$. The machine used for complexity computations

Algorithm 5 compute T0

for $i = 1 : 1000000$ **do**
  $x = 0.55 + (double)i$;
  $x = x + x; x = x./2; x = x * x;$
  $x = sqrt(x); x = log(x); x = exp(x); y = x/x;$
end for

TABLE I. COMPUTATIONAL COMPLEXITY

	$\mathbf{T0}$	$\mathbf{T1}$	$\widehat{\mathbf{T2}}$	$(\widehat{\mathbf{T2}} - \mathbf{T1})/\mathbf{T0}$
$\mathbf{D = 10}$	66.0	797.0	740.6	-0.9
$\mathbf{D = 30}$		1976.0	3167.0	18
$\mathbf{D = 50}$		3266.0	7523.8	64.5

is PC with Intel(R) Core(TM)2 Quad CPU Q9650 3.00GHz, 8GB RAM.

Table I shows the computational complexity of the proposed algorithm while Tables II, III, and IV shows the numerical results in the competition format for the proposed DEcfbLS in 10, 30, and 50, respectively.

A. Comparison against state-of-art algorithms

In order to proved a better understanding of the DEcfbLS performance, the proposed algorithm has been compared with the following modern metaheuristics:

- Covariance Matrix Adaptive Evolution Strategy with increasing population size and restart (G-CMAES), proposed in [30], with initial population $\lambda_{start} = 10$ and factor for increasing the population size equal to 2. All the other parameters are set to standard values.
- Comprehensive Learning Particle Swarm Optimizer (CLPSO), proposed in [31], with population size 60.

TABLE II. RESULTS IN 10 DIMENSIONS

	DEcfbLS				
	Best	Worst	Median	Mean	Std
f1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
f2	$0.00e+00$	$4.38e+03$	$1.77e-05$	$1.01e+02$	$6.09e+02$
f3	$6.09e-06$	$6.37e+00$	$2.88e-01$	$1.14e+00$	$2.00e+00$
f4	$1.91e-07$	$3.21e+01$	$8.82e-03$	$8.19e-01$	$4.57e+00$
f5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
f6	$0.00e+00$	$9.81e+00$	$0.00e+00$	$1.35e+00$	$3.38e+00$
f7	$7.30e-02$	$4.32e+00$	$6.23e-01$	$8.71e-01$	$8.21e-01$
f8	$2.00e+01$	$2.06e+01$	$2.03e+01$	$2.03e+01$	$1.12e-01$
f9	$5.95e-01$	$5.40e+00$	$3.82e+00$	$3.54e+00$	$1.09e+00$
f10	$7.40e-03$	$7.63e-02$	$2.95e-02$	$3.29e-02$	$1.71e-02$
f11	$0.00e+00$	$9.95e-01$	$0.00e+00$	$1.95e-02$	$1.38e-01$
f12	$2.98e+00$	$1.09e+01$	$5.97e+00$	$6.15e+00$	$1.88e+00$
f13	$4.00e+00$	$2.18e+01$	$1.10e+01$	$1.19e+01$	$4.39e+00$
f14	$0.00e+00$	$1.25e-01$	$0.00e+00$	$1.84e-02$	$3.10e-02$
f15	$2.36e+02$	$8.32e+02$	$5.20e+02$	$5.27e+02$	$1.38e+02$
f16	$2.00e-04$	$1.34e+00$	$2.51e-01$	$2.79e-01$	$1.99e-01$
f17	$1.61e+00$	$1.05e+01$	$1.01e+01$	$9.80e+00$	$1.51e+00$
f18	$9.03e+00$	$2.24e+01$	$1.63e+01$	$1.65e+01$	$2.66e+00$
f19	$1.58e-01$	$4.14e-01$	$2.82e-01$	$2.90e-01$	$6.21e-02$
f20	$1.82e+00$	$3.43e+00$	$2.50e+00$	$2.56e+00$	$4.01e-01$
f21	$4.00e+02$	$4.00e+02$	$4.00e+02$	$4.00e+02$	$0.00e+00$
f22	$9.31e+00$	$1.17e+02$	$2.73e+01$	$3.08e+01$	$1.89e+01$
f23	$3.47e+02$	$1.18e+03$	$6.28e+02$	$6.56e+02$	$1.59e+02$
f24	$1.03e+02$	$2.00e+02$	$1.08e+02$	$1.13e+02$	$2.19e+01$
f25	$1.09e+02$	$2.16e+02$	$2.00e+02$	$1.82e+02$	$3.70e+01$
f26	$1.03e+02$	$2.00e+02$	$1.08e+02$	$1.10e+02$	$1.31e+01$
f27	$3.00e+02$	$4.00e+02$	$4.00e+02$	$3.90e+02$	$2.97e+01$
f28	$1.00e+02$	$3.00e+02$	$3.00e+02$	$2.41e+02$	$9.11e+01$

TABLE III. RESULTS IN 30 DIMENSIONS

	DEcfbLS				
	Best	Worst	Median	Mean	Std
f1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
f2	$4.77e+04$	$5.48e+05$	$1.89e+05$	$1.99e+05$	$1.07e+05$
f3	$6.94e+00$	$2.86e+07$	$2.77e+05$	$2.11e+06$	$4.64e+06$
f4	$2.14e+01$	$2.49e+03$	$2.36e+02$	$3.82e+02$	$5.12e+02$
f5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
f6	$2.96e-02$	$2.64e+01$	$6.61e+00$	$7.08e+00$	$4.17e+00$
f7	$2.33e+01$	$8.93e+01$	$5.29e+01$	$5.68e+01$	$1.66e+01$
f8	$2.07e+01$	$2.11e+01$	$2.09e+01$	$2.09e+01$	$9.44e-02$
f9	$1.48e+01$	$3.12e+01$	$2.45e+01$	$2.40e+01$	$2.86e+00$
f10	$0.00e+00$	$6.89e-02$	$1.23e-02$	$2.01e-02$	$1.73e-02$
f11	$0.00e+00$	$9.95e-01$	$0.00e+00$	$5.85e-02$	$2.34e-01$
f12	$3.68e+01$	$6.87e+01$	$5.47e+01$	$5.42e+01$	$8.80e+00$
f13	$5.57e+01$	$1.37e+02$	$1.02e+02$	$1.01e+02$	$1.86e+01$
f14	$8.14e+00$	$7.62e+01$	$3.08e+01$	$3.29e+01$	$1.35e+01$
f15	$1.94e+03$	$7.35e+03$	$3.10e+03$	$3.43e+03$	$1.08e+03$
f16	$9.05e-02$	$3.90e+00$	$3.57e-01$	$7.27e-01$	$8.93e-01$
f17	$3.28e+01$	$3.75e+01$	$3.52e+01$	$3.53e+01$	$1.14e+00$
f18	$5.26e+01$	$1.02e+02$	$8.03e+01$	$7.94e+01$	$1.10e+01$
f19	$1.22e+00$	$1.93e+00$	$1.47e+00$	$1.50e+00$	$1.74e-01$
f20	$9.77e+00$	$1.36e+01$	$1.18e+01$	$1.17e+01$	$6.52e-01$
f21	$2.00e+02$	$4.44e+02$	$3.00e+02$	$3.36e+02$	$9.78e+01$
f22	$9.83e+01$	$4.66e+02$	$2.46e+02$	$2.56e+02$	$9.13e+01$
f23	$2.12e+03$	$4.77e+03$	$3.59e+03$	$3.59e+03$	$4.99e+02$
f24	$2.29e+02$	$2.77e+02$	$2.66e+02$	$2.64e+02$	$9.15e+00$
f25	$2.60e+02$	$2.93e+02$	$2.85e+02$	$2.83e+02$	$5.79e+00$
f26	$2.00e+02$	$2.00e+02$	$2.00e+02$	$2.00e+02$	$6.62e-03$
f27	$7.73e+02$	$1.05e+03$	$9.38e+02$	$9.38e+02$	$5.75e+01$
f28	$3.00e+02$	$3.00e+02$	$3.00e+02$	$3.00e+02$	$0.00e+00$

TABLE IV. RESULTS IN 50 DIMENSIONS

	DEcfbLS				
	Best	Worst	Median	Mean	Std
f1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
f2	$3.16e+05$	$2.42e+06$	$5.73e+05$	$6.55e+05$	$3.74e+05$
f3	$2.04e+07$	$1.39e+09$	$1.87e+08$	$2.20e+08$	$2.14e+08$
f4	$7.86e+01$	$1.25e+04$	$6.16e+02$	$1.21e+03$	$1.94e+03$
f5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
f6	$4.34e+01$	$4.34e+01$	$4.34e+01$	$4.34e+01$	$0.00e+00$
f7	$7.74e+01$	$1.25e+02$	$1.07e+02$	$1.05e+02$	$9.53e+00$
f8	$2.09e+01$	$2.13e+01$	$2.11e+01$	$2.11e+01$	$9.52e-02$
f9	$4.00e+01$	$5.37e+01$	$4.73e+01$	$4.71e+01$	$3.17e+00$
f10	$0.00e+00$	$6.64e-02$	$3.20e-02$	$3.21e-02$	$1.67e-02$
f11	$4.20e-03$	$1.79e+01$	$4.02e+00$	$4.64e+00$	$4.38e+00$
f12	$6.37e+01$	$2.06e+02$	$1.31e+02$	$1.34e+02$	$3.55e+01$
f13	$1.63e+02$	$3.67e+02$	$2.39e+02$	$2.47e+02$	$4.87e+01$
f14	$4.72e+01$	$4.41e+02$	$2.23e+02$	$2.31e+02$	$8.64e+01$
f15	$4.99e+03$	$1.45e+04$	$5.96e+03$	$6.25e+03$	$1.40e+03$
f16	$2.00e-01$	$4.15e+00$	$8.09e-01$	$1.63e+00$	$1.37e+00$
f17	$5.95e+01$	$7.04e+01$	$6.62e+01$	$6.58e+01$	$2.31e+00$
f18	$1.12e+02$	$1.98e+02$	$1.55e+02$	$1.57e+02$	$2.09e+01$
f19	$2.46e+00$	$3.68e+00$	$2.98e+00$	$2.95e+00$	$2.89e-01$
f20	$1.89e+01$	$2.30e+01$	$2.20e+01$	$2.17e+01$	$8.51e-01$
f21	$2.00e+02$	$1.12e+03$	$2.00e+02$	$5.24e+02$	$3.98e+02$
f22	$3.58e+02$	$1.04e+03$	$6.63e+02$	$6.89e+02$	$1.50e+02$
f23	$5.89e+03$	$1.56e+04$	$7.04e+03$	$7.77e+03$	$2.18e+03$
f24	$3.06e+02$	$3.43e+02$	$3.31e+02$	$3.31e+02$	$7.40e+00$
f25	$3.18e+02$	$3.82e+02$	$3.61e+02$	$3.60e+02$	$9.94e+00$
f26	$2.00e+02$	$2.00e+02$	$2.00e+02$	$2.00e+02$	$3.37e-02$
f27	$1.29e+03$	$1.71e+03$	$1.54e+03$	$1.55e+03$	$9.50e+01$
f28	$4.00e+02$	$4.00e+02$	$4.00e+02$	$4.00e+02$	$0.00e+00$

- Modified Differential Evolution with p-best crossover, proposed in [32], with population size 100 and $q = 0.15$ as suggested in original paper.

Each algorithm has been run 51 times for $10000 \times n$ fitness evaluations. The same problems and dimensionality values mentioned above have been considered in this section. Tables V, VI, and VII show the average final fitness $\pm$ the standard deviation and the associated statistical significance according to the Wicloxon test with confidence level 0.95, see [33]. For each pair-wise comparison on the final values obtained by DEcfbLS against GCMAES, MDE-pBX and CCPSO2 ("+", "-" and "=" indicate, respectively, a better, worse or equivalent performance of DEcfbLS against the other algorithms).

TABLE VIII. HOLM-BONFERRONI TEST ON THE FITNESS (REFERENCE = DECFBLS)

i	Optimizer	z	p	α/i	Hypothesis
3	GCMAES	-7.77e+00	3.96e-15	1.67e-02	Rejected
2	CLPSO	-4.24e+00	1.10e-05	2.50e-02	Rejected
1	MDEpBX	-3.35e+00	4.09e-04	5.00e-02	Rejected

Numerical results show that for all the considered dimensionality values, the proposed DEcfbLS tends to outperform the other algorithms. The most challenging competitor appeared to be the MDE-pBX. However, the latter scheme is outperformed of the vast majority of the problems by the proposed memetic structure. This result is confirmed by the Holm-Bonferroni prodedure [34], which we performed as described in [35], with confidence level 0.05. As shown in Table VIII, DEcfbLS is ranked first among the four algorithms, with the null-hypothesis rejected in all the pair-wise comparisons. In other words, DEcfbLS is significantly the best algorithm amongst the four considered in this study and for the problems under investigation.

IV. CONCLUSION

This paper presents a memetic approach based on a DE framework and a simple local search that moves along the axes. The proposed algorithm, namely Differential Evolution with concurrent fitness based Local Search (DEcfbLS) is composed of a DE/rand/1/exp framework and a local search operator. After first step of population initialization, this local search is applied for one time and with a small local budget to all the individuals of the population. This operation appears to have a great impact on the final performance. Subsequently, the local search activations are scheduled by means of criterion based on the exhausted budget. More specifically, DE is performed until a breakpoint is met and LS is activated on multiple population members. The individuals undergoing LS are selected on the basis of their fitness with respect to the fitness of the other population members (the most promising individuals are those undergoing LS). Thus, the total DE and LS budget is made comparable. This scheme, although simple, appears to effectively balance global and local search necessities and offer a respectable performance on a range of various problems. Numerical results demonstrate that DEcfbLS displays a very good performance with respect to modern metaheuristics representing the-state-of-the-art in computational intelligence optimization. The proposed DEcfbLS is potentially a promising candidate to take part to competition on Real-Parameter Single Objective Optimization at CEC-2013.

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 University of Jyväskylä by means of the software for distributed optimization Kimeme [36].

TABLE V. AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST ON THE FITNESS IN 10D (REFERENCE = DECFBLS)

	DEcfbLS	GCMAES		MDEpBX		CLPSO	
f1	$\mathbf{0.00e+00 \pm 0.00e+00}$	$7.95e+03 \pm 7.28e+03$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$\mathbf{0.00e+00 \pm 0.00e+00}$	=
f2	$\mathbf{1.01e+02 \pm 6.09e+02}$	$2.28e+04 \pm 2.99e+04$	+	$4.47e+02 \pm 8.73e+02$	+	$1.22e+06 \pm 6.11e+05$	+
f3	$\mathbf{1.14e+00 \pm 2.00e+00}$	$6.58e+06 \pm 1.73e+07$	+	$9.56e+03 \pm 4.38e+04$	-	$9.67e+06 \pm 7.61e+06$	+
f4	$8.19e-01 \pm 4.57e+00$	$1.19e+04 \pm 8.61e+03$	+	$\mathbf{8.25e-04 \pm 5.54e-03}$	-	$6.81e+03 \pm 2.37e+03$	+
f5	$\mathbf{0.00e+00 \pm 0.00e+00}$	$1.09e+04 \pm 1.74e+04$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$\mathbf{0.00e+00 \pm 0.00e+00}$	=
f6	$\mathbf{1.35e+00 \pm 3.38e+00}$	$7.06e+03 \pm 5.57e+03$	+	$6.35e+00 \pm 4.69e+00$	+	$3.62e+00 \pm 3.40e+00$	+
f7	$\mathbf{8.71e-01 \pm 8.21e-01}$	$1.42e+13 \pm 4.70e+13$	+	$7.24e+00 \pm 9.11e+00$	+	$1.69e+01 \pm 4.76e+00$	+
f8	$\mathbf{2.03e+01 \pm 1.12e-01}$	$2.13e+01 \pm 1.82e-01$	+	$2.05e+01 \pm 8.15e-02$	+	$2.03e+01 \pm 6.42e-02$	+
f9	$3.54e+00 \pm 1.09e+00$	$1.90e+01 \pm 2.45e+00$	+	$\mathbf{2.20e+00 \pm 1.43e+00}$	-	$4.47e+00 \pm 6.22e-01$	+
f10	$\mathbf{3.29e-02 \pm 1.71e-02}$	$1.96e+03 \pm 1.87e+03$	+	$1.36e-01 \pm 1.24e-01$	+	$1.14e+00 \pm 2.24e-01$	+
f11	$1.95e-02 \pm 1.38e-01$	$8.28e+02 \pm 4.02e+02$	+	$2.48e+00 \pm 1.70e+00$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=
f12	$\mathbf{6.15e+00 \pm 1.88e+00}$	$6.60e+02 \pm 3.49e+02$	+	$1.09e+01 \pm 5.21e+00$	+	$1.39e+01 \pm 3.52e+00$	+
f13	$\mathbf{1.19e+01 \pm 4.39e+00}$	$8.41e+02 \pm 7.26e+02$	+	$2.03e+01 \pm 9.51e+00$	+	$1.74e+01 \pm 5.51e+00$	+
f14	$\mathbf{1.84e-02 \pm 3.10e-02}$	$1.02e+03 \pm 3.02e+02$	+	$1.14e+02 \pm 9.80e+01$	+	$2.11e-01 \pm 8.61e-02$	+
f15	$\mathbf{5.27e+02 \pm 1.38e+02}$	$9.81e+02 \pm 2.53e+02$	+	$7.99e+02 \pm 2.85e+02$	+	$7.62e+02 \pm 1.37e+02$	+
f16	$2.79e-01 \pm 1.99e-01$	$\mathbf{6.38e-04 \pm 4.51e-03}$	-	$6.01e-01 \pm 4.43e-01$	+	$9.83e-01 \pm 2.00e-01$	+
f17	$\mathbf{9.80e+00 \pm 1.51e+00}$	$1.14e+01 \pm 8.32e-01$	+	$1.29e+01 \pm 2.05e+00$	+	$1.02e+01 \pm 5.16e-02$	+
f18	$\mathbf{1.65e+01 \pm 2.66e+00}$	$2.61e+02 \pm 3.58e+02$	=	$1.97e+01 \pm 4.99e+00$	+	$3.12e+01 \pm 4.35e+00$	+
f19	$\mathbf{2.90e-01 \pm 6.21e-02}$	$6.49e-01 \pm 1.15e-01$	+	$6.48e-01 \pm 2.22e-01$	+	$3.43e-01 \pm 1.17e-01$	+
f20	$\mathbf{2.56e+00 \pm 4.01e-01}$	$2.93e+00 \pm 6.26e-01$	+	$2.89e+00 \pm 5.42e-01$	+	$2.94e+00 \pm 2.53e-01$	+
f21	$4.00e+02 \pm 0.00e+00$	$3.24e+02 \pm 1.21e+02$	-	$4.00e+02 \pm 0.00e+00$	=	$\mathbf{2.83e+02 \pm 6.63e+01}$	-
f22	$3.08e+01 \pm 1.89e+01$	$1.70e+03 \pm 4.35e+02$	+	$1.19e+02 \pm 1.00e+02$	+	$\mathbf{1.86e+01 \pm 1.45e+01}$	-
f23	$\mathbf{6.56e+02 \pm 1.59e+02}$	$1.77e+03 \pm 3.37e+02$	+	$8.83e+02 \pm 3.33e+02$	+	$1.09e+03 \pm 1.70e+02$	+
f24	$\mathbf{1.13e+02 \pm 2.19e+01}$	$1.65e+02 \pm 5.82e+01$	=	$2.04e+02 \pm 4.82e+00$	+	$1.56e+02 \pm 1.39e+01$	+
f25	$1.82e+02 \pm 3.70e+01$	$1.88e+02 \pm 3.35e+01$	-	$2.01e+02 \pm 2.12e+00$	=	$\mathbf{1.77e+02 \pm 1.79e+01}$	-
f26	$\mathbf{1.10e+02 \pm 1.31e+01}$	$1.23e+02 \pm 4.38e+01$	-	$1.41e+02 \pm 4.23e+01$	+	$1.18e+02 \pm 5.06e+00$	+
f27	$3.90e+02 \pm 2.97e+01$	$\mathbf{3.00e+02 \pm 1.10e-03}$	-	$3.01e+02 \pm 3.92e+00$	-	$3.45e+02 \pm 1.95e+01$	-
f28	$\mathbf{2.41e+02 \pm 9.11e+01}$	$5.48e+02 \pm 7.65e+02$	+	$3.09e+02 \pm 6.90e+01$	=	$2.50e+02 \pm 6.13e+01$	=

TABLE VI. AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST ON THE FITNESS IN 30D (REFERENCE = DECFBLS)

	DEcfbLS	GCMAES		MDEpBX		CLPSO	
f1	$\mathbf{0.00e+00 \pm 0.00e+00}$	$4.42e+03 \pm 2.94e+03$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$\mathbf{0.00e+00 \pm 0.00e+00}$	=
f2	$1.99e+05 \pm 1.07e+05$	$\mathbf{7.65e+03 \pm 4.66e+03}$	-	$8.57e+04 \pm 5.00e+04$	-	$1.84e+07 \pm 4.43e+06$	+
f3	$\mathbf{2.11e+06 \pm 4.64e+06}$	$8.56e+06 \pm 1.25e+07$	+	$1.48e+07 \pm 2.46e+07$	+	$1.40e+09 \pm 5.54e+08$	+
f4	$3.82e+02 \pm 5.12e+02$	$5.28e+03 \pm 3.44e+03$	+	$\mathbf{1.33e+01 \pm 3.92e+01}$	-	$2.79e+04 \pm 3.15e+03$	+
f5	$\mathbf{0.00e+00 \pm 0.00e+00}$	$2.62e+03 \pm 1.18e+03$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$\mathbf{0.00e+00 \pm 0.00e+00}$	=
f6	$\mathbf{7.08e+00 \pm 4.17e+00}$	$2.15e+03 \pm 1.09e+03$	+	$3.34e+01 \pm 3.00e+01$	+	$6.76e+01 \pm 9.13e+00$	+
f7	$\mathbf{5.68e+01 \pm 1.66e+01}$	$1.64e+13 \pm 8.15e+13$	+	$6.09e+01 \pm 1.73e+01$	=	$7.21e+01 \pm 7.80e+00$	+
f8	$\mathbf{2.09e+01 \pm 9.44e-02}$	$2.15e+01 \pm 7.45e-02$	+	$2.10e+01 \pm 5.23e-02$	+	$2.10e+01 \pm 4.75e-02$	+
f9	$2.40e+01 \pm 2.86e+00$	$5.97e+01 \pm 4.92e+00$	+	$\mathbf{2.27e+01 \pm 4.25e+00}$	=	$2.81e+01 \pm 1.73e+00$	+
f10	$\mathbf{2.01e-02 \pm 1.73e-02}$	$1.52e+03 \pm 6.51e+02$	+	$1.44e-01 \pm 8.45e-02$	+	$3.57e+00 \pm 6.63e-01$	+
f11	$5.85e-02 \pm 2.34e-01$	$6.27e+02 \pm 2.94e+02$	+	$4.70e+01 \pm 1.32e+01$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	-
f12	$\mathbf{5.42e+01 \pm 8.80e+00}$	$1.05e+03 \pm 8.28e+02$	+	$7.18e+01 \pm 1.85e+01$	+	$1.43e+02 \pm 1.71e+01$	+
f13	$\mathbf{1.01e+02 \pm 1.86e+01}$	$1.60e+03 \pm 1.28e+03$	+	$1.46e+02 \pm 3.27e+01$	+	$1.80e+02 \pm 1.42e+01$	+
f14	$3.29e+01 \pm 1.35e+01$	$3.57e+03 \pm 1.06e+03$	+	$1.16e+03 \pm 4.50e+02$	+	$\mathbf{1.74e+01 \pm 2.37e+01}$	-
f15	$\mathbf{3.43e+03 \pm 1.08e+03}$	$4.50e+03 \pm 7.47e+02$	+	$3.92e+03 \pm 6.42e+02$	+	$4.62e+03 \pm 3.64e+02$	+
f16	$7.27e-01 \pm 8.93e-01$	$\mathbf{6.24e-03 \pm 6.80e-03}$	-	$1.26e+00 \pm 7.53e-01$	+	$1.97e+00 \pm 2.77e-01$	+
f17	$3.53e+01 \pm 1.14e+00$	$3.78e+03 \pm 1.19e+03$	+	$7.09e+01 \pm 1.28e+01$	+	$\mathbf{3.15e+01 \pm 2.73e-01}$	-
f18	$\mathbf{7.94e+01 \pm 1.10e+01}$	$3.94e+03 \pm 8.53e+02$	+	$8.15e+01 \pm 1.34e+01$	=	$1.99e+02 \pm 1.41e+01$	+
f19	$1.50e+00 \pm 1.74e-01$	$2.45e+00 \pm 4.67e-01$	+	$9.86e+00 \pm 7.61e+00$	+	$\mathbf{1.39e+00 \pm 3.12e-01}$	-
f20	$1.17e+01 \pm 6.52e-01$	$1.41e+01 \pm 2.04e+00$	+	$\mathbf{1.08e+01 \pm 7.21e-01}$	-	$1.30e+01 \pm 4.56e-01$	+
f21	$3.36e+02 \pm 9.78e+01$	$\mathbf{2.24e+02 \pm 4.24e+01}$	-	$3.23e+02 \pm 8.67e+01$	=	$3.02e+02 \pm 5.03e+00$	=
f22	$2.56e+02 \pm 9.13e+01$	$4.50e+03 \pm 1.57e+03$	+	$1.24e+03 \pm 4.50e+02$	+	$\mathbf{1.21e+02 \pm 7.91e+00}$	-
f23	$\mathbf{3.59e+03 \pm 4.99e+02}$	$5.70e+03 \pm 9.43e+02$	+	$4.48e+03 \pm 7.47e+02$	+	$5.59e+03 \pm 3.59e+02$	+
f24	$2.64e+02 \pm 9.15e+00$	$3.01e+02 \pm 3.53e+02$	-	$\mathbf{2.32e+02 \pm 1.10e+01}$	-	$2.47e+02 \pm 4.51e+00$	-
f25	$2.83e+02 \pm 5.79e+00$	$\mathbf{2.54e+02 \pm 1.71e+01}$	-	$2.78e+02 \pm 1.12e+01$	-	$2.97e+02 \pm 1.07e+01$	+
f26	$\mathbf{2.00e+02 \pm 6.62e-03}$	$2.62e+02 \pm 2.45e+02$	-	$2.26e+02 \pm 5.21e+01$	-	$2.01e+02 \pm 3.45e-01$	+
f27	$9.38e+02 \pm 5.75e+01$	$\mathbf{4.01e+02 \pm 1.28e+02}$	-	$6.48e+02 \pm 1.18e+02$	-	$8.19e+02 \pm 1.92e+02$	-
f28	$3.00e+02 \pm 0.00e+00$	$5.76e+02 \pm 1.11e+03$	+	$\mathbf{2.88e+02 \pm 4.71e+01}$	+	$3.00e+02 \pm 8.00e-02$	+

REFERENCES

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

[2] V. P. Plagianakos, D. K. Tasoulis, and M. N. Vrahatis, "A Review of Major Application Areas of Differential Evolution," in *Advances in Differential Evolution*, ser. Studies in Computational Intelligence, U. K. Chakraborty, Ed. Springer, 2008, vol. 143, pp. 197–238.

[3] 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.

[4] T. Rogalsky and R. W. Derksen, "Hybridization of Differential Evolution for Aerodynamic Design," in *Proceedings of the 8th Annual Conference of the Computational Fluid Dynamics Society of Canada*, June 2000, pp. 729–736.

[5] N. Karaboga and B. Cetinkaya, "Design of Digital FIR Filters Using Differential Evolution Algorithm," *Circuits, Systems, and Signal Processing*, vol. 25, no. 5, pp. 649–660, October 2006.

[6] 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.

[7] 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.

[8] 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.

[9] J. Brest, A. Zamuda, B. Bošković, and V. Žumer, "An Analysis of the Control Parameters'Adaptation in DE," in *Advances in Differential Evo-*

TABLE VII. AVERAGE FITNESS ± STANDARD DEVIATION AND WILCOXON RANK-SUM TEST ON THE FITNESS IN 50D (REFERENCE = DECFBLS)

	DEcfbLS	GCMAES		MDEpBX		CLPSO	
f1	$\mathbf{0.00e+00 \pm 0.00e+00}$	$2.67e+03 \pm 1.07e+03$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$\mathbf{0.00e+00 \pm 0.00e+00}$	=
f2	$6.55e+05 \pm 3.74e+05$	$\mathbf{7.96e+03 \pm 4.11e+03}$	-	$4.59e+05 \pm 2.43e+05$	-	$3.58e+07 \pm 5.81e+06$	+
f3	$2.20e+08 \pm 2.14e+08$	$\mathbf{1.01e+06 \pm 1.78e+06}$	-	$9.84e+07 \pm 1.60e+08$	-	$2.77e+09 \pm 6.96e+08$	+
f4	$1.21e+03 \pm 1.94e+03$	$4.58e+03 \pm 2.72e+03$	+	$\mathbf{2.36e+01 \pm 2.36e+01}$	-	$3.42e+04 \pm 3.57e+03$	+
f5	$\mathbf{0.00e+00 \pm 0.00e+00}$	$2.13e+03 \pm 6.58e+02$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	=	$\mathbf{0.00e+00 \pm 0.00e+00}$	=
f6	$\mathbf{4.34e+01 \pm 0.00e+00}$	$1.98e+03 \pm 9.49e+02$	+	$5.57e+01 \pm 2.33e+01$	+	$4.67e+01 \pm 5.09e-01$	+
f7	$1.05e+02 \pm 9.53e+00$	$6.65e+14 \pm 3.78e+15$	+	$\mathbf{6.46e+01 \pm 1.37e+01}$	-	$8.88e+01 \pm 6.24e+00$	-
f8	$\mathbf{2.11e+01 \pm 9.52e-02}$	$2.15e+01 \pm 5.98e-02$	+	$2.12e+01 \pm 4.37e-02$	+	$2.11e+01 \pm 3.28e-02$	=
f9	$4.71e+01 \pm 3.17e+00$	$9.78e+01 \pm 5.23e+00$	+	$\mathbf{4.43e+01 \pm 7.26e+00}$	-	$5.40e+01 \pm 2.41e+00$	+
f10	$\mathbf{3.21e-02 \pm 1.67e-02}$	$1.35e+03 \pm 4.56e+02$	+	$1.53e-01 \pm 2.67e-01$	+	$1.44e+01 \pm 2.11e+00$	+
f11	$4.64e+00 \pm 4.38e+00$	$5.13e+02 \pm 2.36e+02$	+	$1.19e+02 \pm 2.73e+01$	+	$\mathbf{0.00e+00 \pm 0.00e+00}$	-
f12	$\mathbf{1.34e+02 \pm 3.55e+01}$	$2.33e+03 \pm 1.34e+03$	+	$1.59e+02 \pm 3.07e+01$	+	$3.25e+02 \pm 2.65e+01$	+
f13	$\mathbf{2.47e+02 \pm 4.87e+01}$	$3.64e+03 \pm 1.05e+03$	+	$3.20e+02 \pm 4.24e+01$	+	$3.92e+02 \pm 2.01e+01$	+
f14	$2.31e+02 \pm 8.64e+01$	$7.46e+03 \pm 9.56e+02$	+	$2.76e+03 \pm 7.65e+02$	+	$\mathbf{2.97e+01 \pm 2.13e+01}$	-
f15	$\mathbf{6.25e+03 \pm 1.40e+03}$	$8.56e+03 \pm 8.83e+02$	+	$7.61e+03 \pm 8.52e+02$	+	$9.57e+03 \pm 4.86e+02$	+
f16	$1.63e+00 \pm 1.37e+00$	$\mathbf{3.03e-03 \pm 2.72e-03}$	-	$1.85e+00 \pm 8.83e-01$	+	$2.60e+00 \pm 2.92e-01$	+
f17	$6.58e+01 \pm 2.31e+00$	$7.07e+03 \pm 1.20e+03$	+	$1.74e+02 \pm 3.90e+01$	+	$\mathbf{5.35e+01 \pm 5.13e-01}$	-
f18	$\mathbf{1.57e+02 \pm 2.09e+01}$	$6.96e+03 \pm 1.09e+03$	+	$1.84e+02 \pm 2.96e+01$	+	$4.11e+02 \pm 2.07e+01$	+
f19	$2.95e+00 \pm 2.89e-01$	$4.42e+00 \pm 6.69e-01$	+	$3.61e+01 \pm 1.68e+01$	+	$\mathbf{2.68e+00 \pm 3.69e-01}$	-
f20	$2.17e+01 \pm 8.51e-01$	$2.01e+01 \pm 3.02e+00$	-	$\mathbf{1.98e+01 \pm 8.35e-01}$	-	$2.22e+01 \pm 3.52e-01$	+
f21	$\mathbf{5.24e+02 \pm 3.98e+02}$	$6.85e+02 \pm 4.33e+02$	=	$8.67e+02 \pm 3.67e+02$	+	$5.32e+02 \pm 2.15e+02$	+
f22	$6.89e+02 \pm 1.50e+02$	$1.05e+04 \pm 1.33e+03$	+	$3.24e+03 \pm 1.15e+03$	+	$\mathbf{5.55e+01 \pm 2.84e+01}$	-
f23	$\mathbf{7.77e+03 \pm 2.18e+03}$	$1.15e+04 \pm 1.05e+03$	+	$9.00e+03 \pm 1.16e+03$	+	$1.14e+04 \pm 6.12e+02$	+
f24	$3.31e+02 \pm 7.40e+00$	$3.79e+02 \pm 5.21e+02$	-	$\mathbf{2.84e+02 \pm 1.42e+01}$	-	$3.01e+02 \pm 7.53e+00$	-
f25	$3.60e+02 \pm 9.94e+00$	$\mathbf{3.16e+02 \pm 6.31e+01}$	-	$3.67e+02 \pm 1.44e+01$	+	$4.22e+02 \pm 6.94e+00$	+
f26	$\mathbf{2.00e+02 \pm 3.37e-02}$	$3.58e+02 \pm 5.17e+02$	=	$3.46e+02 \pm 8.23e+01$	+	$2.04e+02 \pm 8.57e-01$	+
f27	$1.55e+03 \pm 9.50e+01$	$\mathbf{8.00e+02 \pm 2.42e+02}$	-	$1.28e+03 \pm 1.63e+02$	-	$1.67e+03 \pm 1.86e+02$	+
f28	$\mathbf{4.00e+02 \pm 0.00e+00}$	$1.35e+03 \pm 2.06e+03$	=	$5.42e+02 \pm 7.05e+02$	-	$4.00e+02 \pm 1.02e-03$	+

lution, ser. Studies in Computational Intelligence, U. K. Chakraborty, Ed. Springer, 2008, vol. 143, pp. 89–110.

[10] N. Salvatore, A. Caponio, F. Neri, S. Stasi, and G. L. Cascella, "Optimization of Delayed-State Kalman Filter-based Algorithm via Differential Evolution for Sensorless Control of Induction Motors," *IEEE Transactions on Industrial Electronics*, vol. 57, no. 1, pp. 385–394, 2010.

[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] 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.

[13] N. Hansen and A. Ostermeier, "Completely derandomized self-adaptation in evolution strategies," *Evolutionary Computation*, vol. 9, no. 2, pp. 159–195, 2001.

[14] 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.

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

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

[17] 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.

[18] 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.

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

[20] Z. Z. Liu, F. L. Luo, and M. A. Rahman, "Robust and precision motion control system of linear-motor direct drive for high-speed X-Y table positioning mechanism," *IEEE Transactions on Industrial Electronics*, vol. 52, no. 5, pp. 1357–1363, Oct. 2005.

[21] 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.

[22] 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.

[23] R. Storn, "Differential evolution design of an IIR-filter," in *Proceedings of IEEE International Conference on Evolutionary Computation*, 1996, pp. 268–273.

[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] 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.

[26] F. Caraffini, F. Neri, G. Iacca, and A. Mol, "Parallel Memetic Structures," *Information Sciences*, vol. 227, pp. 60–82, 2013.

[27] I. Poikolainen, G. Iacca, F. Neri, E. Mininno, and M. Weber, "Shrinking three stage optimal memetic exploration," in *Proceedings of the Fifth International Conference on Bioinspired Optimization Methods and their Applications*, 2012, pp. 61–74.

[28] R. Hooke and T. A. Jeeves, "Direct search solution of numerical and statistical problems," *Journal of the ACM*, vol. 8, pp. 212–229, Mar. 1961.

[29] 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.

[30] 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.

[31] J. J. Liang, A. K. Qin, P. N. Suganthan, and S. Baskar, "Comprehensive learning particle swarm optimizer for global optimization of multimodal functions," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 3, pp. 281–295, 2006.

[32] S. G. S. R. S. S. P. Islam, S.M.; Das, "An adaptive differential evolution algorithm with novel mutation and crossover strategies for global numerical optimization," *IEEE Transactions on System Man and Cybernetics-part B*, vol. 42, pp. 482–500, 2012.

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

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

[35] 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.

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

PVII

CLUSTER-BASED POPULATION INITIALIZATION FOR DIFFERENTIAL EVOLUTION FRAMEWORKS

by

I. Poikolainen, F. Neri, F.Caraffini 2014

submitted in April