JYVÄSKYLÄ STUDIES IN COMPUTING
121

Matthieu Weber

Parallel Global Optimization

Structuring Populations in Differential Evolution

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 121

Matthieu Weber

Parallel Global Optimization

Structuring Populations in Differential Evolution

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston Agora-rakennuksen auditoriossa 2
marraskuun 13. päivänä 2010 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in the building Agora, auditorium 2, on November 13, 2010 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2010

Parallel Global Optimization

Structuring Populations in Differential Evolution

JYVÄSKYLÄ STUDIES IN COMPUTING 121

Matthieu Weber

Parallel Global Optimization

Structuring Populations in Differential Evolution

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2010

Editor
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä
Pekka Olsbo, Sini Rainivaara
Publishing Unit, University Library of Jyväskylä

The cover picture represents the author's
great-grandfather. Montage by Rudi Klos.

URN:ISBN:978-951-39-4076-8
ISBN 978-951-39-4076-8 (PDF)

ISBN 978-951-39-4057-7 (nid.)
ISSN 1456-5390

Jyväskylä University Printing House, Jyväskylä 2010

ABSTRACT

Weber, Matthieu
Parallel Global Optimization. Structuring Populations in Differential Evolution.
Jyväskylä: University of Jyväskylä, 2010, 70 p.(+included articles)
(Jyväskylä Studies in Computing
ISSN 1456-5390; 121)
ISBN 978-951-39-4057-7 (nid.), 978-951-39-4076-8 (PDF)
Finnish summary
Diss.

Differential Evolution is a versatile and powerful optimization algorithm that can be applied to a wide range of continuous problems. Under some conditions however, the algorithm may suffer from stagnation, where it stops improving upon its current solutions, especially when applied to medium and large scale problems. The replacement of the usual panmictic population by a structured population composed of interacting subpopulations has been found to improve the performance of the Differential Evolution. This work presents a few algorithmic components build upon such algorithms, aimed at preventing stagnation by acting on the subpopulations themselves or by modifying the search logic of the algorithm. One important finding in this work is that the introduction of small amounts of randomness in structured population algorithm, controlled by using very simple rules leads to significant improvements in performance.

Keywords: global optimization, evolutionary computing, differential evolution, structured population, algorithmic enhancements

Author

Matthieu Weber
Department of Mathematical Information Technology
University of Jyväskylä
Finland
matthieu.weber@jyu.fi

Supervisors

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

Dr. Ville Tirronen
Department of Mathematical Information Technology
University of Jyväskylä
Finland
ville.tirronen@jyu.fi

Reviewers

Dr. Dimitris K. Tasoulis
Mathematics Department
Imperial College London
United Kingdom
d.tasoulis@imperial.ac.uk

Dr. Leonardo Vanneschi
Department of Informatics, Systems and Communication (D.I.S.Co.)
University of Milano-Bicocca
Italy
vanneschi@disco.unimib.it

Opponent

Dr. Janez Brest
Institute of Computer Science
University of Maribor
Slovenia
janez.brest@uni-mb.si

PRÉFACE

Bien que la majeure partie de ce mémoire de doctorat soit écrite en anglais, j'ai décidé d'en rédiger la préface en français afin d'expliquer en termes simples le sujet de mon travail à mon entourage francophone, que la lecture du contenu scientifique de cet ouvrage ne motive pas forcément beaucoup.

Le sujet de ce mémoire se traduit en français par Optimisation globale parallèle : structuration des populations dans l'évolution différentielle. *En termes mathématiques concis, le but est de développer des algorithmes permettant de calculer une valeur approximative de l'optimum d'une fonction de plusieurs variables sans recourir à son expression analytique (qui n'est même pas forcément connue). Pour prendre une comparaison accessible au plus grand nombre, imaginons que l'on veuille préparer un gâteau mais que l'on n'aie pas de recette sous la main. Tout ce que l'on sait c'est qu'il faut des œufs, du sucre, de la farine, du beurre et qu'il faut le faire cuire au four un temps donné à une température donnée.*[1] *On suppose que l'on est aussi capable de goûter un gâteau et de lui donner une note, ceci afin de pouvoir comparer deux gâteaux et d'en choisir le meilleur. L'optimisation de la recette consiste donc à trouver les valeurs des six paramètres de la recette (quantités d'œufs, de sucre, de farine et de beurre, durée et température de cuisson) qui permettent de préparer le meilleur gâteau possible. L'*ensemble des règles dictant les étapes *qui conduisent à la découverte de cette recette optimale s'appelle un algorithme d'optimisation.*

L'idée fondamentale de la méthode utilisée pour optimiser la recette d'un gâteau est déconcertante de simplicité : essayer des recettes complètement aléatoires, goûter les gâteaux ainsi produits et conserver la meilleure recette. Cette manière de procéder a cependant l'inconvénient d'être extrêmement lente[2] *car elle va explorer un grand nombre de recettes absurdes (par exemple des quantités d'ingrédients complètement disproportionnées, une durée de cuisson démesurément longue ou courte, etc.) L'idée des algorithmes dits* metaheuristiques *est alors de « guider » cette exploration aléatoire en se concentrant sur les recettes qui sont proches de celles ayant donné de bons résultats (et qui ont donc eu une bonne note) et en évitant les recettes semblables à celles ayant donné de mauvais gâteaux.*

*Une des approches utilisées pour guider cette exploration est inspirée directement des mécanismes naturels qui permettent l'évolution des êtres vivants. On qualifie ces algorithmes d'*évolutionnistes, *car ils font appel à des concepts tels qu'une* population d'individus, *des mécanismes de* sélection, *de* mutation *et de* recombinaison *que l'on trouve en biologie et en génétique. Le principe des algorithmes évolutionnistes appliqué à nos gâteaux est relativement simple : on commence avec un ensemble de recettes aléatoires (chaque recette se compose donc de quantités aléatoires d'œufs, de sucre, de farine et de beurre, et d'un temps et d'une température de cuisson aléatoires). Chaque recette de cet ensemble est considérée comme un* individu, *et l'ensemble forme donc notre* population.

---

1 On devrait aussi prendre en compte l'ordre dans lequel les ingrédients sont mélangés, mais cela compliquerait inutilement l'exemple.

2 On ne tient ici même pas compte du temps qu'il faut pour préparer le gâteau et le faire cuire ; on mesure en fait la vitesse de l'algorithme en *nombre de gâteaux* qu'il faut préparer avant d'obtenir un résultat satisfaisant.

On va alors essayer chaque recette de la population et lui donner une note. Puis on choisit au hasard des paires de recettes que l'on combine pour produire de nouvelles recettes, de la même manière que dans la nature, deux parents combinent leurs gènes pour produire un descendant.[3] *Des gâteaux sont ensuite préparés en suivant ces nouvelles recettes, et des notes leur sont attribuées. Enfin, on sélectionne les meilleures recettes, c'est-à-dire celles ayant obtenu les meilleures notes, et qui vont composer la génération suivante de la population : on se retrouve donc avec une population composée d'individus un peu meilleurs que ceux de la génération précédente. On répète alors le même processus que ci-dessus (recombinaison, sélection), autant de fois que nécessaire jusqu'à obtention d'une recette produisant un gâteau que l'on juge suffisamment bon.*

L'algorithme décrit ci-dessus (que l'on nomme, selon les variantes, algorithme génétique *ou* stratégie d'évolution*) n'est pas celui sur lequel j'ai travaillé. Mon travail a porté sur un algorithme nommé* évolution différentielle*, où l'on crée une recette* mutante *à partir d'une recette choisie au hasard dans la population à laquelle on ajoute la* différence *entre deux autres recettes, elles aussi choisies au hasard dans la population : ainsi si l'on a par exemple une première recette qui utilise 200 g de farine, une autre 150 g et une troisième 210 g, la recette mutante utilisera* $200 + (150 - 210) = 140$ *g de farine ; on répète ce ensuite calcul*[4] *pour les cinq autres paramètres de la recette, puis on recombine la recette mutante avec une recette parente pour produire une recette descendante. La descendante remplace alors la parente si elle permet de préparer un meilleur gâteau que sa parente ; sinon, on conserve la parente et on rejette la descendante. L'évolution différentielle ne cherche plus à reproduire un mécanisme existant dans la nature, mais son mécanisme un peu abscons la rend, pour certain types de problèmes, plus efficace que les algorithmes décrits plus haut.*

L'idée d'optimisation parallèle *consiste alors à diviser la population en plusieurs sous-populations que l'on va optimiser séparément et en même temps. Certains individus sont alors parfois autorisés à migrer d'une sous-population à l'autre et apportent ainsi les améliorations issues de leur propre évolution aux individus d'une autre sous-population, qui s'est elle-même améliorée mais d'une manière éventuellement différente. La combinaison de ces deux améliorations peut alors conduire à une amélioration encore supérieure, et ainsi de suite.*

Le but de mon travail a été de perfectionner des algorithmes d'optimisation parallèle pour les rendre plus performants, c'est à dire de les rendre capables de produire de meilleures recettes avec comme contrainte un nombre maximum de gâteaux que l'on peut préparer. La comparaison avec les gâteaux s'arrête là, car je me suis intéressé à des problèmes qui n'ont non pas six paramètres comme nos recettes, mais cinq cents ou mille, et qui nécessitent

---

[3] Une telle recombinaison peut se faire dans la pratique en prenant par exemple les quantités d'œufs et de sucre de l'une des recettes parentes, et les autres quantités de l'autre recette parente. On peut aussi introduire une *mutation* en changeant un peu la quantité d'un des ingrédients (ou la durée, ou la température de cuisson) de manière aléatoire : par exemple en augmentant la quantité de sucre de 30 grammes, ou en diminuant le temps de cuisson de 5 minutes. Ces deux transformations sont inspirées directement de mécanismes génétiques qui ont lieu lors des divisions cellulaires dans tous les êtres vivants.

[4] Cette description est simplifiée : la formule exacte est $r_1 + F \times (r_2 - r_3)$ où F est un nombre entre 0 et 1, et une bonne partie de mes recherches a consisté à étudier comment faire varier F afin d'améliorer les performances de l'algorithme.

un million de « préparations de gâteaux », voire plus. Heureusement, au lieu de devoir préparer des gâteaux afin de leur donner des notes, j'ai utilisé de très courts programmes informatiques qui mettent moins d'un millième de seconde à produire une « note », et de nombreux ordinateurs effectuant tous ces calculs en même temps.

Quant à la question « À quoi tout cela sert-il ? » que certains ne manqueront pas de poser, il est très facile de répondre : ça sert à trouver l'optimum d'une fonction. Dans la pratique, bon nombre de problèmes du monde réel peuvent être traduits sous la forme d'une fonction mathématique : faire évoluer un robot dans une pièce sans se cogner aux objets qui s'y trouvent, faire trier par un ordinateur des photos en fonction des personnes ou des objets qui y sont représentés, placer des composants électroniques sur un circuit imprimé de sorte que ce dernier soit le plus compact possible, prédire la structure d'une protéine afin de concevoir de nouveaux médicaments, definir l'emploi du temps de toutes les classes d'une école en fonction des salles de classe disponibles, planifier le déploiement d'antennes-relais pour un réseau de téléphonie mobile, faire jouer un ordinateur à un jeu de société… Ces problèms présentent souvent plusieurs solutions possibles, mais une de ces solutions est meilleure que toutes les autres[5] *: c'est la solution correspondant à l'optimum de la fonction mathématique. Les algorithmes d'optimisation, en trouvant cet optimum, permettent alors de trouver la meilleure solution au problème, et donc de le résoudre.*

[5] Dans un but de simplification, on ne prend pas en compte l'existence possible de plusieurs optima d'égales valeurs.

ACKNOWLEDGEMENTS

My first and very warm thanks go to Ferrante Neri and Ville Tirronen for their guidance, without whom this work would not exist. Ville introduced me to global optimization through its applicative side, while Ferrante lead the way to the algorithmic aspect of the field, eventually resulting in this work. Ferrante and Ville provided numerous algorithmic ideas (some of which even worked), and Ferrante's guidance when writing many of the articles included in this thesis was very much appreciated. Moreover, I probably would never have completed this thesis without their continuous, yet gentle pressure.

Additional thanks go to Jonne Itkonen for enlightening me in my initiation of the Python programming language, and Petteri Olkinuora and Tapani Tarvainen for their help with the computing resources that were needed in producing the hundreds of gigabytes of data resulting from this work. More thanks go to the Faculty of Information Technology for the financial support of this work.

Further acknowledgements must be given to the anonymous reviewers of various journals and conferences who sometimes ask pertinent questions and are the source of new ideas.

Last but not least, my love goes to my wife, my daughter, my parents and my friends for general moral support.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

CONTENTS

LIST OF INCLUDED ARTICLES

PI Matthieu Weber, Ville Tirronen and Ferrante Neri. Fitness Diversity Parallel Evolution Algorithms in the Turtle Race Game. In *Applications of Evolutionary Computing*, volume 5484/2009 of *Lecture Notes in Computer Science*, pages 303–312, 2009.

PII Matthieu Weber, Ferrante Neri and Ville Tirronen. Distributed differential evolution with explorative-exploitative population families. In *Genetic Programming and Evolvable Machines*, volume 10, issue 4, pages 343–371, 2009.

PIII Matthieu Weber, Ville Tirronen and Ferrante Neri. Scale factor inheritance mechanism in distributed differential evolution. In *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, volume 14, number 11, pages 1187–1207, 2010.

PIV Matthieu Weber, Ferrante Neri and Ville Tirronen. Parallel Random Injection Differential Evolution. In *Applications of Evolutionary Computation*, volume 6024/2010 of *Lecture Notes in Computer Science*, pages 471-480, 2010.

PV Matthieu Weber, Ferrante Neri and Ville Tirronen. Parallel Differential Evolution with Endemic Randomized Control Parameters. In *Proceedings of the Fourth International Conference on Bioinspired Optimization Methods and their Applications*, pages 19–29, 2010.

PVI Matthieu Weber, Ferrante Neri and Ville Tirronen. Component Decomposition in Parallel Differential Evolution. In *Proceedings of the Fourth International Conference on Bioinspired Optimization Methods and their Applications*, pages 43–53, 2010.

PVII Matthieu Weber, Ferrante Neri and Ville Tirronen. Shuffle Or Update Parallel Differential Evolution for Large Scale Optimization. In *Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems*, to appear, 2010.

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

The article which introduced the author to global optimization and more particularly parallel global optimization is Article **PI**, where an artificial player for a board game is presented. The author implemented the software modeling the game as well as the four optimization algorithms used for training the player and the distributed framework used for running those software. The author also compiled the results produced by the various algorithms and took part in the writing of the article (description of the game and of the artificial player; production of the figures and result tables).

Articles **PII** to **PVI** form together a study of distributed variants of the Differential Evolution and of algorithmic components aimed at improving the

former. In all these articles, the author implemented all the test functions forming the benchmark on which the algorithms are tested, and based on which they are compared. He also implemented the various algorithms described in those articles and compiled the results produced by the algorithms into the tables and figures shown in the articles.

In addition to the contribution described above which apply to all the articles, the author wrote the descriptions of the various reference algorithms and the section on numerical results in Articles **PII** and **PIII**, while Articles **PV** and **PVI** were integrally written by the author.

Article **PVII** uses a different benchmark than the previous articles, which was imposed by the guest editor of the journal it was published in. The author designed the new algorithm introduced in this article and implemented it as well as the benchmark. He also compiled the results presented therein and wrote the section on numerical results.

This progression in the participation to the preparation of these articles eventually led to the author's scientific independence.

More details on the scientific content of Article **PI** can be found in Section 4.1.4 on page 54 while Articles **PII** to **PVII** are described in Chapter 5 on page 55.

1 INTRODUCTION

The goal of optimization is, for a given problem formalized as an objective function $f : D \rightarrow E$, to find the element of D which gives the "best" solution in E, for a given definition of what "best" means. One can distinguish two categories of such problems: combinatorial problems and continuous ones. In combinatorial problems, there is a finite, albeit potentially immensely large set of discrete solutions to the problem. One famous such problem is the Traveling Salesman Problem, where a salesman must find the shortest path allowing him to visit a given number of cities without passing more than once through each city. Continuous problems on the contrary are represented by functions of the type $f : D \subset \mathbb{R}^n \rightarrow E \subset \mathbb{R}^p$, where there is an infinite number of potential solutions in the continuous space D. In the remainder of this thesis, only continuous problems with $p = 1$ will be considered. Moreover the notion of "best" solution will be reduced to the solution x such that $f(x)$ is minimum. Finally, while D can take any shape in $\mathbb{R}^n$, we will limit ourselves to hyperrectangles. More formally, we will focus on objective functions of the type

$$f : D \subset \mathbb{R}^n \rightarrow \mathbb{R}$$

and trying to find

$$\min_{x \in D} f(x)$$

where D can be expressed as a tuple of intervals of $\mathbb{R}$, thus

$$D = \left([a_1, b_1], \ldots, [a_n, b_n] \right).$$

Throughout the literature on the subject of optimization, multiple names have been used to qualify given concepts. Thus, *function* and *problem* are often used interchangeably, and are sometimes referred to as *test function* or *test problem*. The space D can be called the *search space*, the *problem space* or the function's *domain*. The element $x \in D$ may be named a *vector*, a *solution* or possibly a *point* in D. The *fitness* of x, $f(x)$, is sometimes called the *performance* of x, especially when comparing the fitness of x to that of $y \in D$. Finally, when $n > 1$, x is composed of multiple *design variables*, sometimes also referred to as *components*.

1.1 Derivative-based Optimization

If f has an analytical expression, and under the conditions that f is unimodal and twice differentiable, one can attempt to find its minimum using an analytical method, based on the decomposition of f into a Taylor series. We then can write

$$f(x) = f(x_0) + g(x) \cdot (x - x_0) + (x - x_0)^T \cdot \frac{1}{2} G(x_0) \cdot (x - x_0) + \dots$$

where $g(x)$ is the gradient of f and $G(x)$ is the Hessian matrix of f. Since the optimum x^* of f is a stationary point where $g(x^*) = 0$, we can derive that

$$x^* = -g(x_0) \cdot G^{-1}(x_0) + x_0.$$

In situations where computing $G^{-1}(x_0)$ is impractical, one can use the *steepest descent* method, which consists in replacing $G^{-1}(x_0)$ with the identity matrix and compute

$$x_1 = x_0 - g(x0).$$

While x_1 is not the sought extremum, it is closer to it than x_0 if the $g(x_0)$ step that lead from x_0 to $x1$ was not too big. The size of the step can be controlled by the adjunction of a step-size parameter γ, leading to the recursive equation

$$x_{n+1} = x_n - \gamma \cdot g(x_n).$$

The choice of the proper step size however becomes yet another problem, more so since it depends on the objective function. Moreover, this approximation of the above-described analytical solution causes new problems to appear, calling for more elaborate techniques which can fall into two categories: *quasi-newton* methods attempt to approximate the inverse Hessian matrix in various ways which often require extensive matrix computations, while *conjugate gradient* methods forgo the Hessian matrix entirely and repose on line optimization in conjugate directions. But even if these methods show fast convergence properties on quadratic, once- or twice-differentiable, unimodal functions, their efficiency is not guaranteed on arbitrary functions.

1.2 Derivative-Free Optimization

Arbitrary functions are not necessarily differentiable, in some cases they do not even have an analytical expression. In the case the function is expressed as a computer program the source code of which is available and can be modified, automatic differentiation (see for example Rall, 1981) can be applied to generate a computer program able to compute the function's derivative; the derivative-based methods described above can then be applied as well. Otherwise, one has to turn to other techniques.

Derivative-free optimization methods have thus been developed to allow the optimization of arbitrary functions. Also known as *direct search* methods, they consist in essence in generating a solution x and testing its fitness by computing $f(x)$. This process is then repeated until a satisfactory solution is found. An alternative ending condition for such algorithms is the exhaustion of the predefined *budget* of fitness evaluations allocated for the process. The *enumeration*, also known as the *brute-force search*[1] method, consists in selecting a finite number of points in D and evaluating all of them, keeping the best one as the solution to the problem. One way to select those points is to sample k equally spaced points in each $[a_i, b_i]$ interval, and thus construct a grid covering the whole search space. However impractical the brute-force search may be, it highlights a crucial point, namely the "step-size problem": be the grid too coarse, and one risks to miss the optimum; be it too fine, and the number of tested points increases exponentially with the problem's dimensionality, n.

The main flaw in the brute-force search is that most of the tested points are far from the optimum (and the number of such points increases exponentially with n) and testing them all is a waste of resources. The Hooke-Jeeves and the Nelder-Mead methods are two examples of derivative-free methods which are much more efficient than brute-force.

The Hooke-Jeeves algorithm (Hooke and Jeeves, 1961) is based on a single base point x_0 and a step size h. The main idea of the algorithm is to explore the neighborhood of x_0 along each of the axes of the search space, and to find whether a step of size h towards the positive or the negative direction is leading to a better fitness (if no improvement has been found after exploring both directions, the original position of x_0 on that axis is retained). Once every axis has been probed, the new point x_1, obtained by offsetting x_0 by h or $-h$ along the relevant axes, is evaluated. If the fitness of x_1 is no better than the one of x_0, a new exploration of the neighborhood of x_0 is undertaken, this time with a smaller step size. Otherwise, a new base point x_2 is chosen by taking one step further from x_1 in the direction defined by x_0 and x_1 (formally $x_2 = x_1 + (x_1 - x_0)$), optimistically assuming that the direction is leading towards a better fitness, and the algorithm is applied again on x_2.

While the Hooke-Jeeves algorithm relies on a single point and the systematic exploration of its neighborhood, the Nelder-Mead algorithm (Nelder and Mead, 1965) makes use of a set of $n + 1$ points in D, $x_0, \ldots, x_n$ forming an $n + 1$-dimensional polyhedron, or *simplex*. At each iteration of the algorithm, the indices points are sorted by their increasing fitness so that x_0 has the best fitness and x_n presents the worst fitness. The procedure then consists in constructing a candidate replacement point x_r for x_n by reflection of x_n in respect with the barycenter x_m of the other $x_0, \ldots, x_{n-1}$ points. Depending on the performance of x_r compared to x_0 and x_{n-1}, an *extension* point may be created in an optimistic attempt to explore further in the same direction, or on the contrary a *contraction* point may be computed closer to x_m. If none of the above attempts lead to a better solution,

[1] This method is comparable to the *exhaustive search* for combinatorial problems, where all the possible solutions are tested.

the simplex is contracted around its best point in order to reduce the exploration range in the next iteration of the algorithm.

It is worth noting that even though these algorithms do not require any knowledge about the fitness function and particularly its derivative, they still do make use of some crude form of gradient by sampling the search space and measuring the difference of the fitness between two points from this sample. If this "gradient" is leaning toward an improvement, the algorithms will make optimistic attempts to follow it in the hope to find yet a better point in that direction.

2 METAHEURISTICS

Meta-heuristics (from Greek μετά "beyond", meaning here "higher-level", and εὑρίσκω, "I find") are a class of general-purpose heuristic algorithms, which consist in applying a heuristic method in a controlled way to guide it into obtaining faster results compared to a totally random set of trials and errors.

The algorithms briefly described in the previous chapter, although efficient when applied to unimodal functions, suffer a serious flaw when confronted to multimodal functions, i.e., functions exhibiting more than one extremum, and therefore do not fully qualified as metaheuristics. Since a starting point is indeed required for initializing them, and because these algorithms are meant to progress towards the nearest extremum, they might not find the global extremum of the function if the starting point is located near a local, sub-optimal one. More general algorithms are thus called for.

Nature having in numerous occasions inspired scientific research, optimization is no exception and examples of optimization algorithms inspired from natural phenomena are numerous. For example, the behavior of atoms during the cooling down of a molten substance has inspired the *Simulated Annealing* algorithm, while the use of memory in order not to repeat past errors is the basis of *Tabu Search*. The random travel of foraging ants in search for a source of food have inspired the *Random Walk* algorithm, and the mechanism by which the ants mark the shortest path to a source of food is applied in the *Ant Colony* algorithm. Finally, one must mention the theory of evolution, natural selection and genetics, which are the foundation for *Genetic Algorithms*, *Genetic Programming* and *Evolution Strategies*.

Among the above examples, a distinction can be made between single-point methods and population-based methods.

2.1 Single-Individual Metaheuristics

The Random Walk algorithm (see for example Solis and Wets, 1981) is probably one of the simplest global optimization algorithm: a random initial base point is

chosen in the search space, and, on each iteration of the algorithm, a candidate point is generated by adding a random, normally distributed deviation to the base point. If this candidate point presents a better fitness than the base point, it becomes the new base point. Since the normal distribution is unbounded, a deviation of any size may occur and the algorithm has the possibility to move out of a local optimum in order to find a better optimum and eventually the global one. The drawback of this method is that the probability for this to happen is very low, and the algorithm may take an unacceptably long time to find the global optimum.

The Simulated Annealing algorithm (Kirkpatrick et al., 1983) is similar to the Random Walk, but it allows moves towards a worse solution with a given probability, which diminishes over the course of the algorithm. This design is directly inspired by the atoms of molten metal settling down into a crystallized structure when the temperature diminishes. At higher temperatures, important thermal agitation allows the atoms to take a larger number of positions with respects to each other, while at lower temperature this probability diminishes exponentially, the atoms settling into a minimum energy configuration. In Simulated Annealing, the probability of accepting a move towards a worse position is a function of the form $\Theta = e^{d\beta/T}$ where T is a temperature parameter which decreases with every iteration of the algorithm.

In Tabu Search (Glover, 1989a,b), the algorithm keeps a track, in its short-term memory, of the solutions which have already been visited in order to prevent wasting time by visiting them more than once. A long-term memory can also be added to the algorithm in order to reinforce attractive solutions. In continuous problems, the probability of reaching twice the same solution is extremely low, rendering the short-term memory useless. Tabu Search therefore needs to be adapted to continuous functions, for example by defining a neighborhood around the solution to be ignored, as in Siarry and Berthiau (1998).

2.2 Population-Based Metaheuristics

In multi-point methods, the search space is sampled in multiple points, which are considered concurrently. One can consider two classes of metaheuristics, employing multiple starting points and imitating natural processes. *Evolutionary algorithms* thus considers the multiple starting points as a population of individuals which breed with each other and adapt themselves to their environment. Multiple solutions thus support each other, in the sense that new solutions are derived from several precursor solutions. In *swarm intelligence* algorithms, the starting points are considered as members of a flock or a swarm, each individual having only a limited intelligence and following simple behavioral rules, but contributing altogether to the solution of the problem. Multiple solutions are then rather following the lead of one of them.

2.2.1 Evolutionary Algorithms

The idea of applying evolutionary principles to computer science can be traced back to Turing, who mentioned "genetical or evolutionary search" in 1948, while Bremermann experimented on "optimization through evolutionary and recombination" methods in the 1950s. The most common evolutionary algorithms in use nowadays are *Evolutionary Programming*, *Genetic Algorithms*, *Evolution Strategies* and *Genetic Programming*. All these algorithms (of which exist many variants) follow a common structure, represented as pseudocode in Algorithm 1. The main characteristics of evolutionary algorithms are the use of a population of parent solutions, the creation of offspring solutions from selected parents, and the selection of survivors individuals among both parents and offsprings. These survivors form the population of the next generation and the process repeats.

```
Generate an initial population by sampling the search space
Evaluate the fitness of each candidate solution
while termination conditions are not met do
  Select parents
  Create offsprings by recombining the parents
  Mutate the offsprings
  Evaluate the fitness of each offspring
  Select survivors for the next generation
end while
```

ALGORITHM 1 General structure of an evolutionary algorithm

Evolutionary Programming

Evolutionary Programming (Fogel et al., 1965, 1966) mimics the way new features appear in living beings through mutation of their genome. Each individual is composed of a real-valued vector representing the candidate solution, and each component x_i of this vector is associated to a self-adaptive parameter σ_i.

The algorithm does not employ a recombination scheme, but relies solely on mutation. Thus, for a candidate solution of the form

$$(\bar{x}, \bar{\sigma}) = (x_1, \ldots, x_i, \ldots, x_n, \sigma_1, \ldots, \sigma_i, \ldots, \sigma_n)$$

on each iteration of the algorithm, an offspring $(\bar{x}', \bar{\sigma}')$ is generated in the following manner:

$$\sigma_i' = \sigma_i(1 + \alpha N(0,1))$$

$$x_i' = x_i + \sigma_i' N(0,1)$$

where the prime symbol $'$ indicates the newly generated component or parameter of the offspring, $N(0,1)$ is a Gaussian random variable with a mean of 0 and a standard deviation of 1, and α is a constant value traditionally set to 0.2.

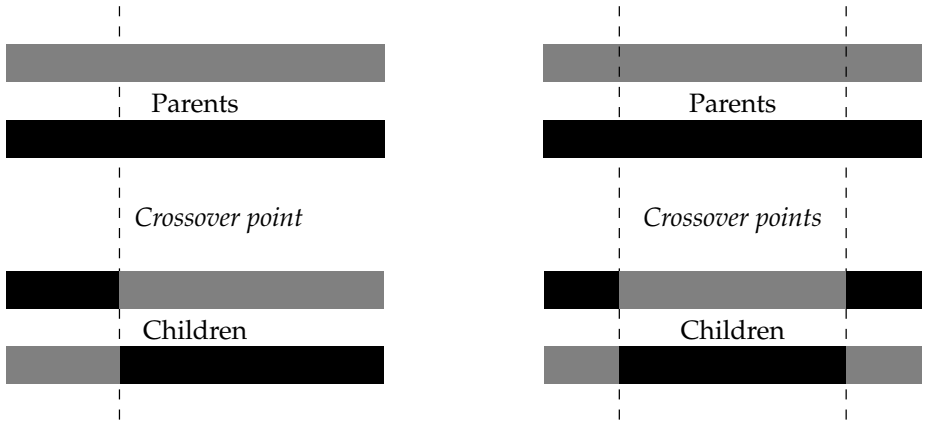
The offspring population is then evaluated, and merged with the parent population. This extended population then undergoes a survivor selection as follows: each individuals is compared against a set of other, randomly selected individuals; the individuals which won the highest number of comparisons are selected.

Genetic Algorithms

Genetic algorithms (Goldberg, 1989; Holland, 1962) attempt to reproduce some of the biological mechanisms happening in living organisms during reproduction: two parent individuals recombine their genomes in order to produce offsprings. In the original implementation of the genetic algorithm, the genome of an individual is represented by a string of bits, and each component can therefore take a value of 0 or 1. This requires for the solutions to be one way or another to be encoded as bit strings, using for example Gray code. In modern implementations, integer- and real-valued genetic algorithms have also been proposed.

Parent selection can be done in multiple ways. *Proportionate selection* assigns the parents a selection probability depending on their fitness, so that the individuals with the best fitnesses have higher chance to be selected than the one with lower fitnesses. In *k-tournament selection*, each parent is the individual presenting the best fitness out of a random sample of k individuals.

Offsprings are then generated by recombining the parents using a *crossover* function, which concatenates components from one parent with other components from the other parent. Figure 1 presents two common crossover variants, single-point and multi-point crossovers. More complex crossover schemes are also possible, such as mask crossover or partially mapped crossover, see Eiben and Smith (2003).



(a) Single-point crossover (b) Multi-point crossover

FIGURE 1 Examples of crossover

The new offspring then undergoes a mutation process, which consists in altering each component of the newly created solution with a very low probability. In the case of bit-string encoded individuals, the mutation may consist for example

in changing a bit's value from 0 to 1 or vice versa. The need for mutation arises from the fact that if one would create offsprings only by crossover, the points in the search space that can be reached by the individuals depends entirely on the initial population. Consequently, a part of the space would be totally inaccessible. Random mutations however are not subjected to that limitation, and therefore give the individuals a chance to explore the entire search space.

After the offsprings have been evaluated, the survivors are selected according to the *generational strategy*, where the parents are completely replaced by their offsprings.

Evolution Strategies

Evolution Strategies (Rechenberg, 1971) look similar in appearance to Genetic Algorithms, but since the former use real-valued solutions, they are better suited for continuous function optimization than the latter. They employ they same type of individuals as Evolutionary Programming does, where each component x_i of the candidate solution is associated with a self-adaptive parameter σ_i (see Evolutionary Programming above for more details).

In Evolution Strategies, a population of μ parents are recombined to produce a population of $\lambda \geq \mu$ offsprings. Recombination involves two randomly chosen parents and can be *discrete*, where some components are from one parent while the others come from the second parent, or *intermediate*, where the components of both parents are for example averaged.

As in Genetic Algorithms, the offsprings are then mutated, by the addition to each component of a random value, typically a zero-mean Gaussian random variable, in a way similar to the one used in Evolutionary Programming:

$$x_i' = x_i + \sigma_i' N(0,1)$$

where the prime symbol $'$ indicates the newly generated component or parameter of the offspring and $N(0,1)$ is a Gaussian random variable with a mean of 0 and a standard deviation of 1. Concerning the generation of the σ_i' parameter, several different schemes can be found in literature, such as the $1/5$ success rule (Rechenberg, 1971), uncorrelated mutation with n step sizes and correlated mutation, see Eiben and Smith (2003) for more details. As it was the case with Genetic Algorithms, mutations allow to explore regions of the search space which cannot be reached by only recombining individuals from the parent population.

Finally, two different survivor selection mechanisms can be applied. In the (μ, λ) Evolution Strategy, the μ best offsprings are selected to become the next population of parents; this variant is also known as the "comma" variant. When following the $(\mu + \lambda)$ Evolution Strategy, the parent and offspring populations are merged, and the μ best individuals are selected to become the parents for the next generation; this variant is sometimes called the "plus" variant.

Genetic Programming

Genetic Programming (Koza, 1992, 1994) differs from the above evolutionary algorithms in that the individuals do not represent a point in a geometrical space, but rather a *parse tree*, a data structure used for representing mathematical formulas, logical expressions or a computer procedure, where the internal nodes represent mathematical operators or logical programming structures, and the external leaves represent the variables and constants. Because of this difference, the algorithms behave differently from the other evolutionary algorithms.

Individuals in Genetic Programming undergo recombination and mutation, as they do in the evolutionary algorithms described above. Parent selection is usually carried out by a fitness- based proportionate procedure, as in Genetic Algorithms, although tournament-based selection is a possibility as well. In some cases, a process called over-selection is used: parents are sorted by their fitness values and separated into two groups, one containing the best individuals, and the other one containing the remaining ones. Parents in the first group are given a higher probability to be selected than those in the second group. Group sizes need to be empirically chosen following a rule of thumb (Langdon and Poli, 2001).

Recombination between parents is most commonly performed by exchanging subtrees between the two parents in order to generate two offsprings. Mutation is similarly executed by replacing a subtree with a pseudo-randomly generated one. While mutations were discouraged in the first implementations (Koza, 1992), it was shown later that a low mutation probability can be beneficial (Luke and Spector, 1997; Banzhaf et al., 1998).

Finally, survivor selection is traditionally generational, where the offsprings completely replace the parents, although in many new implementations, different schemes are employed.

2.2.2 Swarm Intelligence

While evolutionary algorithms find their inspiration in the theory of evolution and natural selection, swarm intelligence has its roots in the observation of groups of animals, where some kind of collective intelligence emerges although the animals themselves do not present signs of intelligence but rather following relatively simple rules. The observation of ants has thus led to the development of the *Ant Colony Optimization* algorithm, while the metaphor of a flock of birds is the foundation for the *Particle Swarm Optimization* algorithm.

Ant Colony Optimization

Originally developed as Ant System in Dorigo (1992), the Ant Colony Optimization algorithm is inspired by the method used by foraging ants in search for food to find the shortest path leading to it. The first application of the algorithm was to find the optimal path in a graph, and while other problems can be solved by this algorithm, they need to be expressed as a search for the best path in a weighted graph.

Each solution in the algorithm represents one ant moving on the graph. While the virtual insect travels the graph randomly at first, it lays down a pheromone trail, which can be picked up and followed by other ants who are thus going to reinforce it. The pheromone however has the property to evaporate over time, resulting in the trail disappearing on paths which are only seldom used, making them less attractive to the ants. Shorter paths being travelled more quickly than longer ones, they are used by more ants and therefore receive more pheromone than the former. Eventually, all the ants are going to follow the same, optimal path.

The virtual pheromone is a set of parameters associated with graph components (either nodes or edges) whose values are modified by the ants during the course of the algorithm. Evaporation is an essential feature in Ant Colony Optimization, preventing the algorithm to get stuck in a local minimum: without this feature, the paths chosen at random by the first ants would be more attractive than the other.

Particle Swarm Optimization

Although the Particle Swarm Optimization (Eberhart and Kennedy, 1995) is not directly inspired by a natural phenomenon, it is based on the metaphor of a group of particles using their "personal" and "social" experience in order to explore the problem's space and locate optima. A particle represents a pseudo-randomly initialized point x_i in that space, and is associated to a pseudo-randomly initialized velocity[1] v_i. The algorithm iteratively updates the particle's position until a terminating criterion is met. The particle's position x_i is thus updated by applying the formula

$$x_i' = x_i + v_i$$

where the prime symbol $'$ represents the newly generated value. Each particle additionally keeps a record of which one, of all the positions it has taken, was the most successful; this point x_i^{best} is called the *local best*. Moreover, the population of particles tracks which one of the particles' local bests has the best fitness, and this point x^{best} is named the *global best*. The particle's velocity is then updated based on these two points, using the formula

$$v_i' = v_i + \phi_1 U(0,1)(x_i^{best} - x_i) + \phi_2 U(0,1)(x^{best} - x_i)$$

where ϕ_1 and ϕ_2 are parameters and $U(0,1)$ represents a uniformly-distributed pseudo-random variable in the interval $[0,1]$.

A particle's local best can be considered as the particle's memory, and thus the personal learning experience of the particle due to successful and unsuccessful moves. The global best, being shared by all the particles represents the above-mentioned social experience. The particles therefore decide of their movements along two dimensions, the first one being their own experience, and the second

[1] Strictly speaking, the velocity should be better named a perturbation or a displacement vector, since it does not represent a speed in the physical acceptance of the term.

one being the imitation of the population's most successful individual. The utilization of random scale factors allow to maintain diversity in the population and to prevent premature convergence.

2.2.3 Memetic Algorithms

Memetic algorithms are defined in Moscato and Norman (1989) as hybrids of one metaheuristic algorithm with one or more local search algorithms. A *meme*, as defined by Dawkins (1976), is an abstract element of cultural ideas, symbols or practice, transmitted between individuals by means of speech, gestures, rituals... An individual (in the context of a metaheuristic algorithm) to which a local search algorithm is applied can be compared to a person going to school, learning skills which are not transmitted genetically but rather memetically, making this person a "better" person and making the individual a better performing one. The rationale behind memetic algorithms is the conviction that a combination of several algorithms is more efficient than one single algorithm.

Memetic algorithms form a broad class of optimization algorithms; merging one of the numerous metaheuristics with one or several of the many existing local search algorithms leads to a tremendously large number of possible combinations. Memetic algorithms can be classified in several ways. Lozano et al. (2004) discriminates between algorithms where the local search is implemented within the variation operator (e.g., mutation or crossover) in order to control the generation of offsprings, and algorithms where the local search is applied to already generated offsprings in order to improve upon their initial fitness value (this manner of using a local search algorithm is named *life-time learning*).

Another way of classifying memetic algorithms has been proposed in Ong et al. (2006), and depends on the manner the local search algorithmic components are selected, in the case of memetic algorithms employing multiple local search components. The first category groups algorithms governed by a set of predefined, immutable rules governing the behavior of the search, and are called *hyperheuristic* algorithms. In this case, the designer of the algorithm must take several decisions, such as: whether to use multiple local search components (Krasnogor, 2002; Krasnogor et al., 2002), to choose local search algorithms which employ a variety of local search logics (Krasnogor, 2004), how to balance the amount of fitness evaluations allocated to the local search component with respect to the amount devoted to the global search component (Ishibuchi et al., 2003; Ong and Keane, 2004), how to coordinate the algorithmic components by observing the population diversity (Caponio et al., 2007)... One can however alleviate the need to take some of those decisions by letting the algorithm decide by itself, thus self-adapting at runtime. Algorithms of the second category present those properties, by making use of multiple memes competing against each other, selecting the best meme for being used in the future; these algorithms are qualified as *meta-lamarckian*. The third and last category goes beyond the concept of meta-lamarckian algorithms, and the memes are undergoing their own evolution, as part of the individuals' genomes (self-adaptive memetic al-

gorithms) or as co-operating populations of memes (co-evolutionary memetic algorithms).

3 DIFFERENTIAL EVOLUTION

Differential evolution (Price et al., 2005) is a versatile optimizer which, although being originally described as an evolutionary algorithm, shares in certain circumstances some features with Swarm Intelligence algorithm. The general structure of DE is the same as the one of other evolutionary algorithms (see Algorithm 1 on page 21), with which it shares in particular the concepts of mutation and crossover, but the Differential Evolution cannot be anymore considered to be inspired by evolutionary processes found in Nature. What sets Differential Evolution apart from e.g., Genetic Algorithms or Evolution Strategies is that while in those the mutation is an operation which produces a random change in an individual, the mutation operator in Differential Evolution takes place before the crossover and produces on one hand a deterministic change, and on the other hand may not involve the individual itself at all. Another noticeable difference is that the crossover operation in the Differential Evolution involves one parent and its provisional offspring rather than two parents as is the case in the above-mentioned evolutionary algorithms. The general structure of the Differential evolution is illustrated in Algorithm 2 on the next page.

The initial sampling of the population is performed pseudo-randomly with a uniform distribution function within the decision space. At each generation, for each individual x_i of the S_{pop}, three individuals x_r, x_s and x_t are pseudo-randomly extracted from the population. According to the DE logic, a provisional offspring x'_{off} is generated by mutation as:

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

where $F \in [0, 1+]$ is a scale factor which controls the length of the exploration vector $(x_s - x_t)$ and thus determines how far from point x_t the provisional offspring should be generated. With $F \in [0, 1+]$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1.

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

```
Generate an initial population by sampling the search space
Evaluate the fitness of each candidate solution
while termination conditions are not met do
  for each individual in the population do
    Generate a provisional offspring by mutation
    Generate the offspring by crossover between the parent and the provisional
    offspring
    Evaluate the fitness of the offspring
  end for
  for each individual in the population do
    if the individual's offspring has a better fitness than its parent then
      Substitute the offspring to the parent
    end if
  end for
end while
```

ALGORITHM 2 General structure of the Differential Evolution

probability and the final offspring x_{off} is generated:

$$x_{off,j} = \begin{cases} x'_{off,j} & \text{if} \quad U(0,1) < CR \quad \text{or} \quad j = j_{rand} \\ x_{i,j} & \text{otherwise} \end{cases} \tag{2}$$

where $U(0,1)$ is a uniformly distributed random variable in $[0,1]$; j is the index of the gene under examination and j_{rand} is a randomly selected gene which is always exchanged, in order to prevent cases where no gene from the provisional offspring is exchanged.

The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) < f(x_i)$; otherwise no replacement occurs. It must be remarked that although the replacement indexes are saved one by one during the generation, the actual replacements occur all at once at the end of the generation. For the sake of clarity, the pseudo-code highlighting the working principles of the DE is shown in Algorithm 3 on the following page.

3.1 Mutation Schemes

The version of the Differential Evolution described above employs one particular mutation scheme involving three distinct, randomly selected individuals in the population and noted DE/rand/1/bin. DE refers to the Differential Evolution, "rand" means that the first base point is chosen at random, "1" indicates that one difference vector is used and "bin" refers the the binomial crossover applied after the mutation (see below for a discussion about different types of crossover methods). Several other mutation schemes are described in Qin and Suganthan (2005) (DE/rand/1/bin is repeated here for the sake of completion):

Generate S_{pop} individuals of the initial population pseudo-randomly
while budget condition **do**
  for $i = 1 : S_{pop}$ **do**
    Compute $f(x_i)$
  end for
  for $i = 1 : S_{pop}$ **do**
    {** Mutation **}
    Select three distinct individuals x_r, x_s, and x_t, distinct from x_i
    Compute $x'_{off} = x_r + F(x_s - x_t)$
    {** Crossover **}
    $x_{off} = x_{i,j}$
    Choose j_{rand} randomly in $\{1, \ldots, n\}$
    $x_{off,j_{rand}} = x'_{off,j_{rand}}$
    for $j = 1 : n$ **do**
      Generate $U(0,1)$
      if $U(0,1) < CR$ **then**
        $x_{off,j} = x'_{off,j}$
      end if
    end for
    {** Selection **}
    if $f(x_{off}) < f(x_i)$ **then**
      Save index for replacement $x_i = x_{off}$
    end if
  end for
  Perform replacements
end while

ALGORITHM 3 DE pseudocode

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

where x_{best} is the solution with the best performance among the individuals of the population, x_u and x_v are two additional pseudo-randomly selected individuals. The number of difference vectors is by no means limited to two, and thus Chakraborty (2008) presents a generalization as

$$x'_{off} = y_i + F \cdot \frac{1}{N} \sum_{k=0}^{N-1} \left(x_{r(2k+1)} - x_{r(2k+2)} \right)$$

where y_i is the base vector and $x_{r(k)}$ are $2(N+1)$ randomly selected vectors.

It is worth mentioning the rotation invariant mutation shown in Price (1999) as well:

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

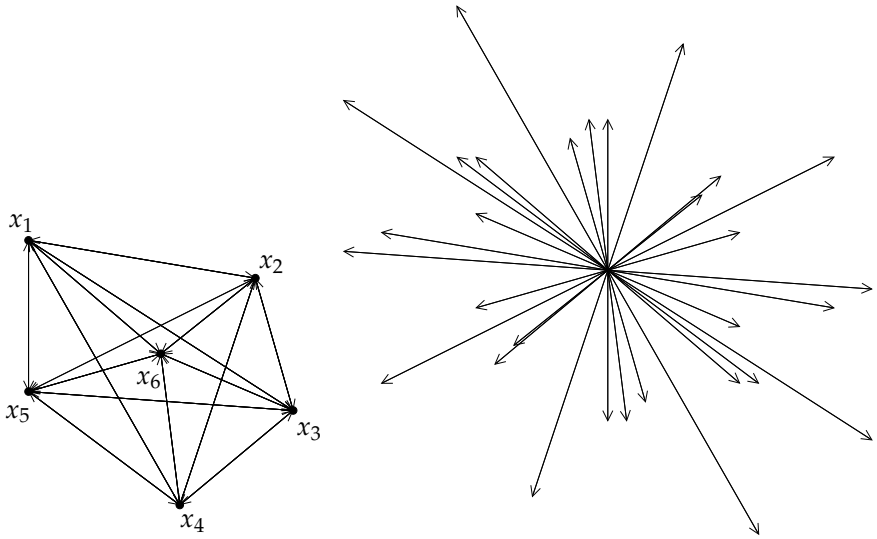
where K is is the combination coefficient, which, as suggested in Price (1999), should be chosen with a uniform random distribution from $[0, 1]$ and $F' = K \cdot F$. For this special mutation the mutated solution does not undergo a crossover operation.

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

$$x'_{off} = \begin{cases} x_r + F(x_s - x_t) & \text{if} \quad U(0,1) < p_F \\ x_r + (F+1)(\frac{x_s + x_t}{2} - x_r) & \text{otherwise} \end{cases} \tag{3}$$

As for the DE/current-to-rand/1, when this mutation scheme is applied, it is not followed by a crossover.

Figure 2 on the next page shows the distribution of difference vectors that can be constructed from a population of six points. The vectors have all been re-arranged around one point for a clearer view of the points that can potentially be reached when applying a difference vector to one point. An important fact that can be discovered by examining Figure 2b on the following page is that for every point in the population, there is only a limited number of points from the search space that can be reached by applying such a mutation scheme (see below for a more detailed discussion). One can also notice that schemes such as DE/rand/2/bin, employing more than one difference vectors, allow to increase the number of potential provisional offsprings, thus allowing to explore a wider



(a) All possible difference vectors between the six points of the population

(b) The same vectors rearranged around their origin, showing their distribution

FIGURE 2 Difference vectors and their distribution for a population of six points

area of the search space or provide a finer granularity to the exploration of the area occupied by the population.

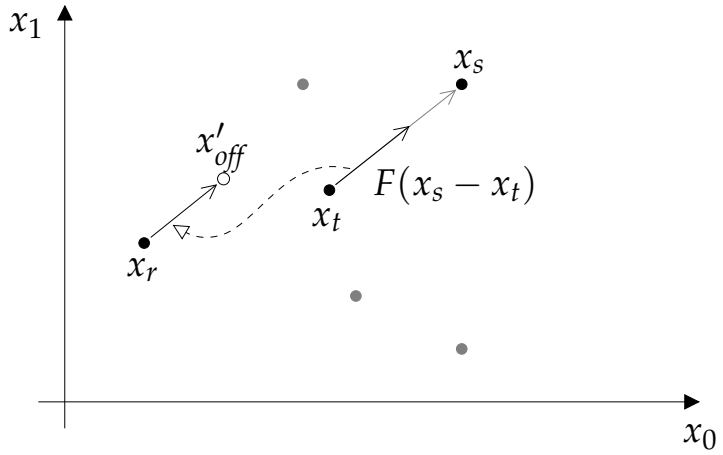
Figure 3 on the next page illustrates where provisional offsprings are generated in the search space by the various schemes described above. From the non-exhaustive list of mutation schemes presented above, one can tentatively categorize them, based on the location of the generated provisional offspring compared to the population:

- The schemes using the individual presenting the best performance (named x_{best} in the figures) as the base point, such as DE/best/1/bin and DE/best/2/bin. These schemes tend to generate the provisional offsprings around the best individuals, which is also one characteristic of Swarm Intelligence algorithms.

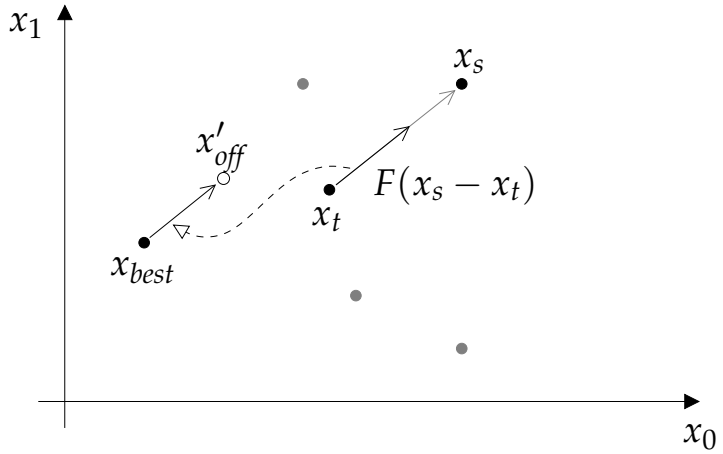
- The schemes using a random vector as the base vector, such as DE/rand/1/bin, DE/rand/2/bin or DE/rand-to-best/2, where the provisional offspring can be generated potentially anywhere in the vicinity of the population.

- The schemes using the point currently considered by the algorithm as the base point such as DE/current-to-rand/1 or DE/current-to-best/1/bin, can be considered as an intermediate between the two above categories, since the offsprings are generated in the vicinity of the current point.

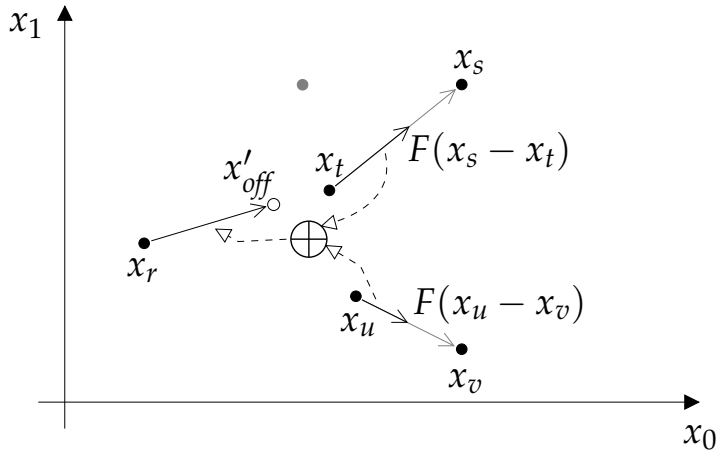
- Schemes involving the best individual without using it as the base point but rather considering the direction towards that individual can also be



(a) DE/rand/1/bin

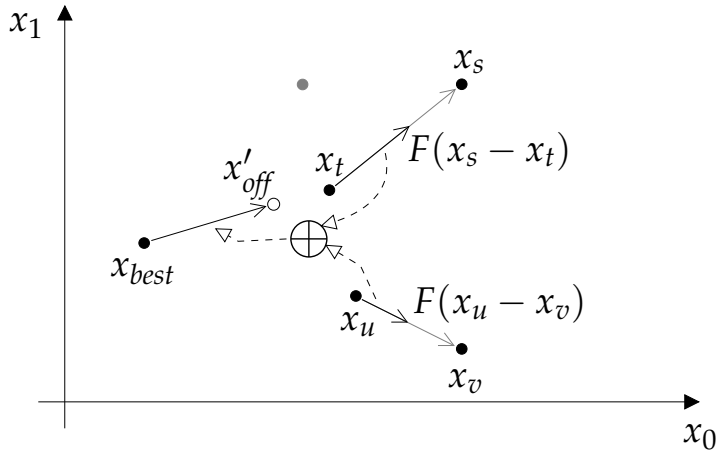


(b) DE/best/1/bin

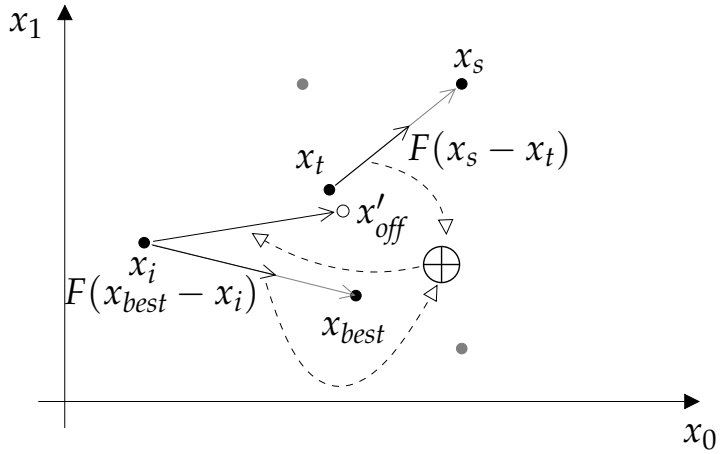


(c) DE/rand/2/bin

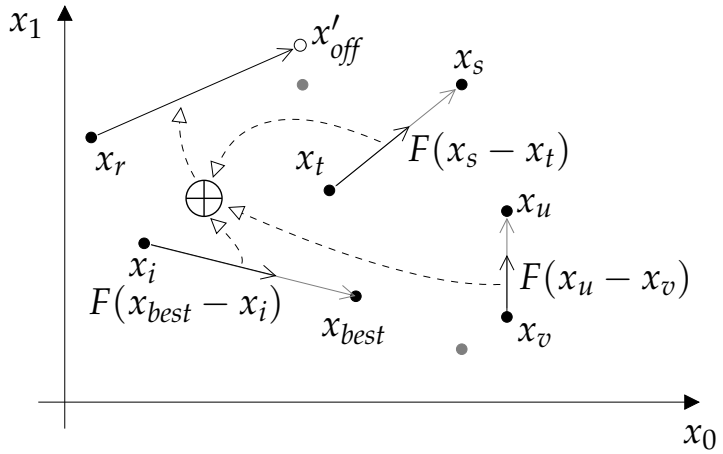
FIGURE 3 Mutation schemes



(d) DE/best/2/bin

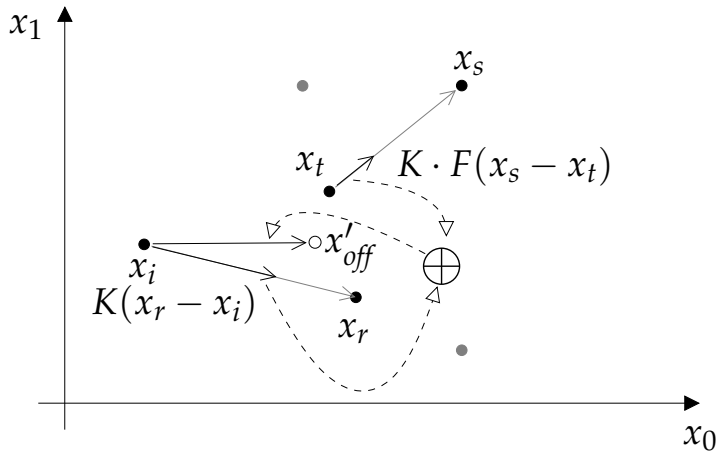


(e) DE/current-to-best/1/bin

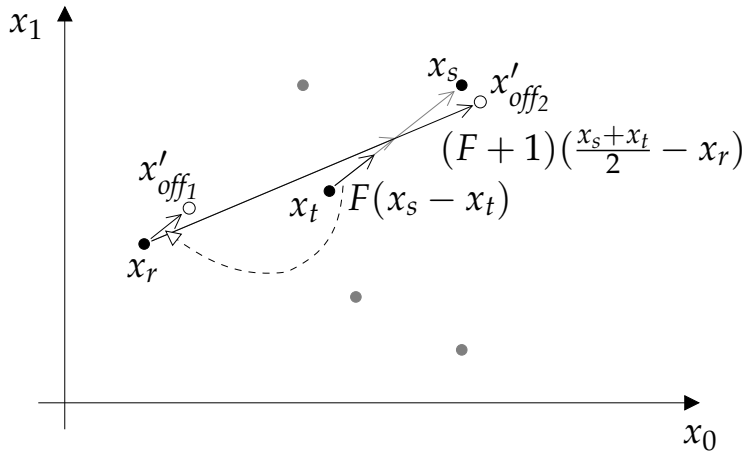


(f) DE/rand-to-best/2/bin

FIGURE 3 Mutation schemes (continued)



(g) DE/current-to-rand/1



(h) DE/rand/1/either-or

FIGURE 3 Mutation schemes (continued)

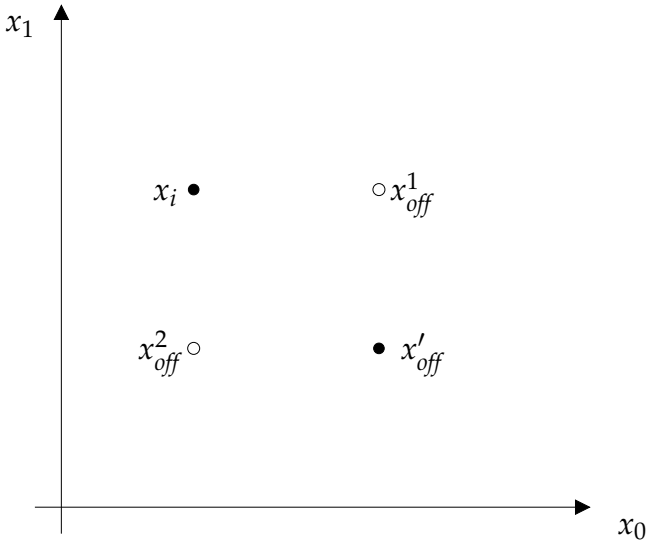


FIGURE 4 The crossover between the two-dimensional individuals x_i and x'_{off} can produce four different offsprings: x^1_{off}, x^2_{off}, and the two degenerate ones x_i and x'_{off}.

considered as an intermediate albeit in a different way than the one above. In this case the direction towards the best individual gives the mutation a bias towards that individual without restricting the explored area to its immediate vicinity.

Directed mutation schemes are a class of mutation schemes where the fitness of the individuals involved is taken into account. One such example is the trigonometric mutation presented in Section 3.6.2. A generalization of mutation schemes and a classification including the directed ones as well as the ones described above has been proposed in Feoktistov (2006): mutations schemes are categorized based on whether they are purely random, directed, using the best individual or directed and using the best individual.

On must however keep in mind that depending on the function to which the algorithm is applied, some mutation schemes perform better than others, and that none of them has proved to be the best for all problems (Chakraborty, 2008).

3.2 Crossover Schemes

Most mutation schemes require a recombination step involving the provisional offspring generated by the mutation and the parent. While the binomial crossover mentioned in Section 3.1 on page 29 is the most common recombination method used in the Differential Evolution, Price et al. (2005) mentions three other ones: one-point crossover, N-point crossover and exponential crossover.

Figure 4 illustrates the result of crossover in two dimensions (represented by axes x_0 and x_1). When recombining the parent individual x_i with the provisional

offspring x'_{off}, one can obtain four different results:

- If no gene from x_i is replaced by the corresponding gene from x'_{off}, the offspring x_{off} is equal to x_i. Since this *status quo* does not allow any improvement on x_i, one usually tries to prevent it from happening by forcing at least one gene replacement when implementing the crossover function.
- If the first gene of x_i (along axis x_0) is replaced by the corresponding gene from x'_{off}, the resulting offspring is then x^1_{off}: it shares its first gene with the provisional offspring and its second gene with its parent, and can be represented as the vector $(x'_{off,0}, x_{i,1})$.
- Conversely, if the second gene of x_i (along axis x_1) is substituted with the corresponding one from x'_{off}, the result is x^2_{off}, represented as the vector $(x_{i,0}, x'_{off,1})$.
- Finally, if both genes of x_i are replaced, the offspring is equal to x'_{off}.

Crossover can therefore be seen as another mechanism for producing exploratory moves along the axes of the search space. Similarly to the mutation schemes described above, the crossover produces a deterministic and limited number of such moves.

3.2.1 One-Point Crossover

One-point crossover (see e.g., Holland, 1992) has already briefly been presented in Section 2.2.1 on page 21 and in Figure 1a on page 22. In a more formal way, a one-point crossover between the provisional offspring x'_{off} and the current individual x_i produces an offspring x_{off} such that

$$x_{off,j} = \begin{cases} x'_{off,j} & \text{if} \quad j < j_{cr} \\ x_{i,j} & \text{otherwise} \end{cases} \tag{4}$$

where $x_{i,j}$ is the gene of x_i under consideration and j_{cr}, randomly selected in $\{1, \ldots, n\}$ with a uniform distribution, is the crossover point.

3.2.2 N-Point Crossover

N-Point crossover (see e.g., Eshelman et al., 1989), illustrated in Section 2.2.1 on page 21 and in Figure 1b on page 22, randomly subdivides x_i and x'_{off} into $N+1$ partitions; x_{off} is then generated by taking alternatively partitions from one and from the other vector:

$$x_{off,j} = \begin{cases} x'_{off,j} & \text{if} \quad \exists k \in \{1, \ldots, (N+1)/2\} | j \in p_{2k} \\ x_{i,j} & \text{otherwise} \end{cases} \tag{5}$$

where $P = \{p_1, \ldots, p_{N+1}\}$ is the set of the randomly selected partitions of the dimensions of the search space.

3.2.3 Exponential Crossover

The exponential crossover (see e.g., Caruana et al., 1989) is a two-point crossover where the boundaries of the partitions are determined by a series of random numbers. The procedure starts at a random gene in the current individual's genome, where this gene is replaced by the corresponding one of the provisional offspring. For each subsequent gene (wrapping around the end of the genome if necessary), a uniformly distributed random number is generated in $[0,1]$ and if this number is lower than the crossover rate CR, the gene is replaced by the corresponding one from the provisional offspring. If on the contrary the random number is higher than CR, the procedure stops there and no more genes are replaced. More formally, the exponential crossover can be described as follows:

$$x_{off,j} = \begin{cases} x'_{off,j} & \text{if} \quad j = j_{rand} \quad \text{or} \quad (j - j_{rand}) \bmod n \in J \\ x_{i,j} & \text{otherwise} \end{cases} \tag{6}$$

where j is the index of the gene under consideration, j_{rand} is a randomly selected gene and J is the set of gene indexes defined as $J = \{1, \ldots, k\}$, $k < n | \forall m \in J, r_m < \text{CR} \quad \text{and} \quad (k = n-1 \quad \text{or} \quad (k < n-1 \quad \textit{and} \quad r_{k+1} \geq \text{CR}))$, with $R = \{r_1, \ldots, r_{n-1}\}$ a set of $n-1$ uniformly distributed random numbers in $[0,1]$. The gene at index j_{rand} is always replaced, which ensures that the offspring always differs from its parent by at least one gene.

The name of this crossover method comes from the fact that the probability that k or fewer genes are inherited from the provisional offspring is $1 - \text{CR}^k$ (see Price et al. (2005) for the demonstration), which is an exponential function of k.

3.2.4 Uniform (Binomial) Crossover

In the uniform crossover (see e.g., Syswerda, 1989), each gene from the parent individual has an equal probability CR to be replaced by a gene from the provisional offspring. Since this method of recombination was described in Equation 2 on page 29, it will not be repeated here.

The uniform crossover is named after the fact that each gene has the same probability to be replaced. It is sometimes referred to as binomial due to the fact that the number of replaced genes follows a binomial distribution (see again Price et al. (2005) for the demonstration).

In the uniform crossover as well as in the exponential crossover, the crossover rate parameter CR determines how "far" from the parent the offspring is located. With a low crossover rate, close to 0, most of the offspring's genome is identical to its parent's and the offspring is therefore close to it. If on the contrary the crossover rate is close to 1, the offspring will be very similar to the provisional offspring which, depending on the selected mutation scheme can be located far from the parent, thus allowing a wider exploration radius of the search space.

3.3 Survivor Selection

The step following the generation of the offsprings consists in selecting which of the parents and of the offsprings will constitute the initial population for the next generation.

In simple Genetic Algorithms, the age is the sole selection criterion (see e.g., Goldberg, 1989), meaning that, provided that the offspring population is of the same size as the parent population, the latter is entirely replaced by the former. Since the fitness of the offsprings is not taken into account in this step, age-based survivor selection is efficient only if the selection of the parents is based on their fitness. Since the Differential Evolution does not select the parents based on their fitness, this method is not suited for this algorithm.

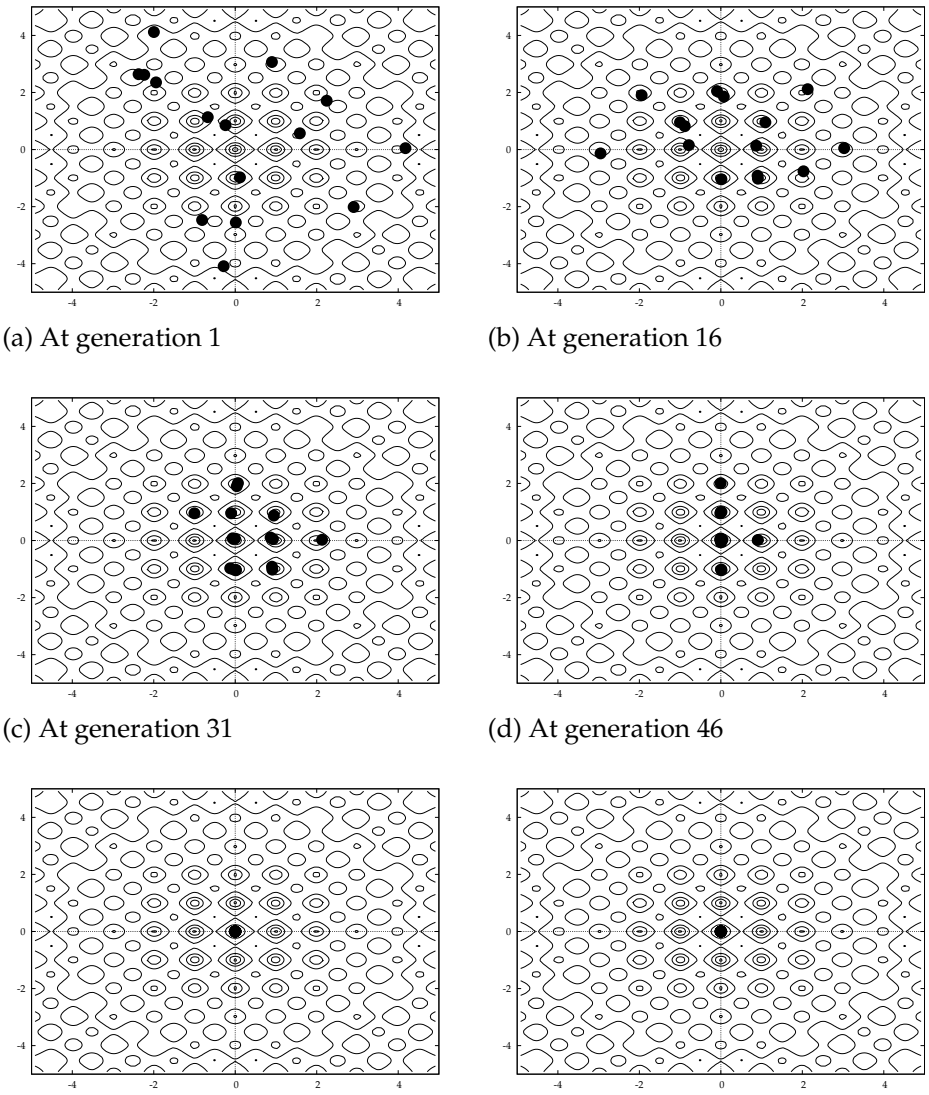
On the opposite, the surviving individuals can be selected based on their fitness only (Rudolph, 1996), a method sometimes referred to as *elitism*, and used for example in the $(\mu + \lambda)$ Evolution Strategy algorithms. With this method only the best individuals of both the parent and offspring populations are retained, regardless of whether they are parents or offsprings.

The (μ, λ) Evolution Strategy algorithms make use of an intermediary approach, where only the μ best individuals out of the λ offsprings are retained, thus considering both the age and the fitness criteria (Bäck and Schwefel, 1995).

Tournament selection, used in Genetic Algorithms and Evolutionary Programming for parent selection, can also be applied to select survivors (Fogel et al., 1966). The *one-to-one tournament* selection used in the Differential Evolution is a form of tournament involving only the parent and its offspring; the best of them is selected as the survivor. This method of selection, which is also used for the same purpose in the Particle Swarm Optimization, ensures that only offsprings that are better than their parent are retained, and that the best individuals in the population are preserved as well. It must be noted that there are two different ways of implementing the one-to-one tournament in the Differential Evolution: the parents that are outperformed by their offsprings can be either replaced immediately after the offspring has been generated (*continuous selection*), or all at once after all the offsprings have been generated (*discrete selection*). In the first case, one of more of such offsprings may be selected as x_r, x_s and x_t, whereas in the second case only parent individuals can be selected. While the original definition of the Differential Evolution makes use of a discrete selection mechanism, continuous selection has been studied e.g., in Tagawa (2009).

3.4 The Evolutionary Algorithm-Swarm Intelligence Duality

The Differential Evolution was designed originally as a form of evolutionary algorithm, with which it shares its main structure: parent selection, offspring generation by mutation and recombination, and survivor selection. Some of its



(a) At generation 1

(b) At generation 16

(c) At generation 31

(d) At generation 46

(e) At generation 61

(f) At generation 76

FIGURE 5 Example of convergence of the DE/rand/1/bin Differential Evolution with a population of 15 individuals on the two-dimensional Rastrigin function

characteristics however allow to classify it as a swarm intelligence as well.

The behavior of any evolutionary algorithm can be summarized into three main phases (Eiben and Smith, 2003): exploration of the search space for promising solutions, exploitation of those solutions to improve upon them and finally convergence towards the global optimum. This evolution is illustrated in Figure 5 on the preceding page, where the Differential Evolution is used for finding the global optimum of the Rastrigin function: at first the individuals of the population are spread over the search space, they then slowly cluster around the best solutions before finally converging towards the global minimum of the function.

In some cases however, an evolutionary algorithm may fail at finding the global optimum (Eiben and Smith, 2003). Different algorithms can thus be characterized by the manner in which this failure happens: Genetic Algorithms (Goldberg, 1989) for example tend to converge quickly toward one local minimum, while swarm intelligence algorithms on the contrary maintain a high spatial diversity, preventing the population from converging (Bonabeau et al., 1999). While many evolutionary algorithms suffer from *premature convergence* by exploiting too early their solutions, the Differential Evolution rather behaves as a swarm intelligence in this respect and exhibits symptoms of *stagnation*, where the population explores the search space indefinitely without finding any exploitable solution (Lampinen and Zelinka, 2000).

Another argument in favor of classifying the Differential Evolution as a swarm intelligence is that when the mutation scheme employed involves the best individual, the population of a Differential Evolution follows the evolution of its best individual, which is one of the characteristics of swarm intelligence algorithms.

Finally, one can argue that the mutation and crossover operators in the Differential Evolution differ quite much from those, inspired by natural mechanisms, employed in other evolutionary algorithms (Chiong et al., 2010). In the Differential Evolution, the crossover involves a parent and its offsprings instead of two parents as it does in e.g., Genetic Algorithms, and the mutation does not use a purely random modification, as can be found in e.g., Evolution Strategies.

Depending on the point of view, the Differential Evolution may thus be considered either as an evolutionary algorithm or as a swarm intelligence, with both of which it shares characteristics.

3.5 Implicit Self-Adaptation: a Double-Edged Sword

One of the main characteristics of the Differential Evolution lies in the mechanism of self-adaptation that emerges from the algorithmic structure (Feoktistov, 2006). It has been shown in the description of the algorithm that the generation of an offspring by a parent individual depends on the other individuals in the population, through the mutation and crossover mechanisms. In the beginning of the optimization process, when the individuals are widely spread over the

search space, the large distances separating the individuals allow for difference vectors with a large norm and therefore allow to explore the space with a large step size. During the course of the optimization, the individuals tend to concentrate towards portions of the space exhibiting potentially good solutions, therefore reducing the norm of most difference vectors (if the population is divided in at least two, widely separated clusters, large difference vectors can still be produced by choosing individuals from two distinct clusters, but difference vectors produced by individuals selected from within one cluster will have a comparatively small norm). The step size of the exploration is therefore progressively reduced and the search performed in the neighborhood of the solutions. The behavior of the algorithm thus progressively shifts from exploring the entire search space to exploiting the promising solutions discovered during the exploration (Mininno and Neri, 2010).

This self-adaptation mechanism is however inherently flawed (Neri and Tirronen, 2010). Since the number of distinct offsprings that can potentially be produced by one population is deterministic and limited by the number of individuals in that population, the case can arise when no offspring exhibits a better performance than its parent, causing the population to stagnate i.e., it will not be renewed by the offsprings. The search will be then repeated with the same population and the same step size, but the population is likely not going to converge, not even towards a local optimum.

3.6 Overcoming Stagnation

Some amount of effort has been put in solving the stagnation problem described above (see e.g., Neri and Mininno, 2010; Neri and Tirronen, 2010), which can be summarized as a need for the algorithm to be allowed more search moves than those defined by the basic mutation scheme and crossover. The variants of the Differential Evolution that have been published over the years as a result of that research can be categorized according to the mechanism they use:

- The control parameters, scale factor and crossover rate, can be dynamically changed during the course of the optimization.
- Supplementary moves can be generated by various means.
- The size of the population can be changed during the process, reducing the number of individuals or increasing it.
- The problem space can be separated into sub-spaces which are optimized separately.

3.6.1 Dynamic Control Parameters

If one considers that stagnation is caused by too large step sizes, the obvious response is to change the step size during the course of the optimization. The step size depends on the difference of two vectors in the population and on the value of the scale factor. While the former cannot be acted upon when e.g., the DE/rand/1/bin scheme is employed, the latter can however be modified dynamically.

In Zaharie (2002), the scale factor F is replaced by a normally distributed random variable hereafter named F_i or F_j. The practice of sampling F_i for each individual x_i is called *dithering*, while sampling F_j for each component $x_{i,j}$ of the individual x_i is called *jitter*. While dithering only changes the norm of the difference vector, jitter changes its orientation as well, which causes this process to fundamentally differ from the classic Differential Evolution using a constant scale factor. Using a normally distributed random variable $N_j(0, \sigma)$ for the scale factor as described above causes the value of F_j to be close to 0 when σ approaches 0. One variant of the dithering/jitter algorithm consists therefore in defining $F_j = F \cdot N_j(1, \sigma)$, which generates a distribution of F_j values centered around F. This variant, as well as the employment of other statistical distributions such as log-normal, uniform and power law as an alternative to the normal distribution, are described in Price et al. (2005).

Differential Evolution with Random Scale Factor (Das et al., 2005), designed for the optimization of noisy functions, is one other example of dithering based on a uniform distribution with $F_i = 0.5 \times (1 + U(0, 1))$, where $U(0, 1)$ is a random number sampled with a uniform probability in $[0, 1]$. The algorithm additionally modifies the selection mechanism by introducing a threshold margin $\tau = k \cdot \sigma_n^2$ based on the variance σ_n^2 of the noise: the fitness of the offspring must be better than its parent's by at least τ for the former to be allowed to replace the latter.

Self-Adaptive Control Parameter Distributed Evolution (Brest et al., 2006) uses an approach similar to dithering but extends it to both control parameters. Each individual x_i is assigned a scale factor F_i and a crossover rate CR_i which are used for generating that individual's offspring; those two values are inherited by the offspring as well. On every generation, F_i and CR_i each have a fixed probability to be updated to a new value based on a uniform distribution.

The Fuzzy Adaptive Differential Evolution (Liu and Lampinen, 2005) makes use of two fuzzy logic controllers to adapt the values of the control parameters to the evolution of the population between one generation and the next one. The magnitude of the evolution is measured both in the search space by computing the root mean square value of the components of all the individuals between generation g and $g + 1$, and in the fitness domain, by computing the root mean square value of the fitness of all the individuals. These values are then mapped to fuzzy variables which can take the values *small*, *medium* or *big*, and new values for the scale factor and the crossover rate are eventually computed according to a set of rules.

These dynamic changes in the values of the control parameter ensures that

the step size does not remain constant over time, thus giving the possibility to a stagnant population to overcome the problem and to continue improving upon its solutions.

3.6.2 Supplementary Search Moves

Stagnation can also be considered to be caused by a lack of "good" search moves, i.e., search moves leading to offsprings which outperform their parent. Supplementary search moves can be introduced by hybridizing the Differential Evolution with a local search algorithm that attempts to improve some of the individuals in the population (often the one with the best performance). Another possibility is the generation of additional search moves based on the existing ones (i.e., produced by the mutation scheme).

The Differential Evolution with Trigonometric Mutation (Fan and Lampinen, 2003) makes use of a mutation scheme which differs from the one described in Section 3.1 on page 29 in that it takes into account the fitness of the three randomly selected vectors it is based on. The provisional offspring is thus computed as

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

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

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

The provisional offspring thus generated is the centroid of the triangle formed by x_r, x_s and x_t, where the weight of each individual is a function of its own fitness fitness and of the average fitness of the other two individuals. The goal of this mutation scheme is to increase the explorative component of the Differential Evolution algorithm by detecting optimal directions; it can be in a sense considered as a single-step local search.

The incorporation of a crossover-based local search into the Differential Evolution in order to improve the performance of the algorithm is presented in Noman and Iba (2008). Since the length of the local search i.e., the number of steps of local search that must be performed depends on the problem, it is impossible to define it *a priori*. This algorithm therefore proposes to adaptively change the length of the local search by taking feedback from the search. This local search is applied at the beginning of every generation to the individual presenting the highest performance. The chosen local search is the simplex crossover algorithm presented in Tsutsui et al. (1999), which takes for input the target individual and additional $n_p - 1$ parents randomly selected in the population. The target individual is then recursively recombined with each of those parents to produce one offspring. If this offspring exhibits better performance than the target individual, it is selected as the new target individual and one additional step of local search is performed. Otherwise, the target individual replaces the original best individual in the population, and the Differential Evolution algorithm goes on.

The Differential Evolution with Global and Local Neighborhoods uses the concepts of population neighborhoods presented in Chakraborty et al. (2006) and

Das et al. (2009). The individuals of the population being randomly sorted, the individuals $x_{i-k}, \ldots, x_i, \ldots, x_{i+k}$ represent a neighborhood of radius k of individuals x_i. A new mutation operator is then designed as

$$x_{off} = wG_i + (1 - w)L_i$$

where w is a weight factor set between 0 and 1 balancing the impact of the local contribution L_i and the global contribution G_i. The former is defined as

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

and the latter as

$$G_i = x_i + \alpha(x_{p-\text{best}} - x_i) + \beta(x_r - x_s)$$

where $(x_{l-\text{best}}$ is the best performing individual in the neighborhood of x_i, x_p and x_q are randomly selected in the neighborhood, $x_{p-\text{best}}$ is the best performing individuals in the whole population and x_r and x_s are two randomly selected individuals from the population. While Chakraborty et al. (2006) suggests to set $\alpha = \beta$, the weight w varies during the course of the optimization process according to

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

where where w_{min} and w_{max} are the lower and upper bounds of the weight factor, respectively. The indexes g and g_{max} denote the current generation index and the maximum amount of generations, respectively. Three additional schemes for the weight factor update have been presented and compared in Das et al. (2009): an exponential increase rule, a randomly generated w and an evolutionary rule. In the latter case, each individuals x_i is assigned a weight w_i, and on every generation, a new value w_i' is computed as

$$w_i' = w_i + F(w_{best} - w_i) + F(w_r - w_s)$$

where w_{best} is the weight of the best performing individual and w_r and w_s are the weights of x_r and x_s in the formula of G_i.

The Opposition-Based Differential Evolution presented in Rahnamayan et al. (2006) and Rahnamayan et al. (2008) proposes to create supplementary search moves by taking the symmetrical of the individuals with regard to the center of a rectangular hyperparallelepiped. More formally, the individual $\tilde{x}_i$, symmetrical of x_i is defined by its genes as

$$\tilde{x}_{i,j} = a_j + b_j - x_{i,j}$$

where j is the index of the genes under consideration and a_j and b_j are the extremities of the bounding box along the j^{th} dimension of the search space. At the beginning of the algorithm, after the initial sampling of the population, a first set of symmetrical individuals is generated, which are then merged with the initial population, and the S_{pop} best ones are retained for the first generation. During the subsequent generations, this process has a given probability j_r (jump rate) to

be repeated, this time using the center of the population's bounding box i.e., of the smallest possible rectangular hyperparallelepiped containing the whole population; from this point on, the symmetrical individuals are generated according to

$$\tilde{x}_{i,j} = \min_i x_{i,j} + \max_i x_{i,j} - x_{i,j}$$

where $\min_i x_{i,j}$ and $\max_i x_{i,j}$ are the respectively minimum and maximum values taken by the j^{th} component of the individuals in the population.

3.6.3 Hybrid Approaches

The scale factor being a key element in the definition of the step size of the Differential Evolution, its value is crucial for the performance of the algorithm. The optimal value of the scale factor however depends on the problem being optimized. In Differential Evolution with Scale Factor Local Search (Tirronen et al., 2009; Neri et al., 2009; Neri and Tirronen, 2009), the authors propose to consider the choice of the scale factor as a one-dimensional problem to be optimized with a local search algorithm. During the generation of an offspring, the local search algorithm has a given probability to be activated. The fitness of a particular value of the scale factor is the fitness of an offspring which is generated with this scale factor. It must be noted that during the local search, the search domain of the scale factor is $[-1.2, 1.2]$; negative values mean that the direction of the search (in the Differential Evolution) is reversed.

In the JADE algorithm presented in Zhang and Sanderson (2007), the values of the control parameters are updated based on a memory of values leading to offsprings that outperform their parents. The algorithm employs the DE/current-to-$p$best/1/bin mutation scheme

$$x'_{off} = x_i + F_i(x^p_{best} - x_i) + F_i(x_r - x_s)$$

where x^p_{best} is randomly chosen among the $100p\%$ best individuals in the population with $p \in (0, 1]$, F_i is the scale factor associated with individuals x_i and x_r and x_s are two individuals randomly selected in the population. The candidate offspring x'_{off} is then recombined with individual x_i using the uniform crossover defined in Equation 2 on page 29 where the crossover rate CR is replaced by the value CR_i associated to individuals x_i. The values of F_i and CR_i are randomly generated at the beginning of every generation based on two parameters μ_F and μ_{CR}. While the CR_i values are based on a normally distributed random variable centered on μ_{CR}, the F_i values of one third of the individuals are generated by a uniformly distributed random variable, while the others are generated by a normally distributed random variable centered on μ_F. When an offspring x_{off} outperforms its parent, the scale factor F_i and the crossover rate CR_i of its parent are saved in two sets respectively named S_F and S_{CR}. At the end of every generation of the algorithm, the values of μ_F and μ_{CR} are updated by calculating a weighted average between the current value of the parameter and, for μ_{CR} the mean value

of S_{CR}, and for μ_F the Lehmer mean of S_F; the latter is defined by

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

In the Self-Adaptive Differential Evolution (Qin and Suganthan, 2005), multiple mutation schemes are employed and each individual x_i is assigned a set of probability values p_i^k, one for each mutation scheme. In the more efficient implementation of the algorithm presented in Qin et al. (2009), the mutations schemes in use are DE/rand/1/bin, DE/rand-to-best/2/bin, DE/rand/2/bin and DE/current-to-rand/1. The probabilities $p_i^k, k = 1 \ldots, 4$ for each of those schemes are initialized to 0.25. During the subsequent learning period of LP generations, the number of offsprings $n_s^{i,k}$ that better their parent is accounted, for each individual x_i and for each mutation scheme k; the number of offsprings $n_f^{i,k}$ that are worse than their parent is accounted as well. At the end of the learning period, on each generation G the probabilities p_i^k are updated according to the formula

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

where

$$S_i^k = \frac{\sum_{G-LP}^{G-1} n_s^{i,k}}{\sum_{G-LP}^{G-1} n_s^{i,k} + \sum_{G-LPi}^{G-1} n_f^{i,k}} + \epsilon$$

and ϵ is a small value set to 0.01, designed to ensure a numerical stability of the algorithm in the case $S_i^k = 0$ for all the mutation schemes. The mutation scheme used when producing one particular provisional offspring is then selected by Stochastic Universal Sampling (Baker, 1987) based on the probabilities assigned to the four possible mutation schemes by the parent individual. Each individual x_i is additionally assigned one scale factor F_i and one crossover rate CR_i^k for each of the four mutation schemes. The value of F_i is initialized by means of a random value sampled from a normal distribution $N(\mu, \sigma)$ where $\mu = 0.5$ and $\sigma = 0.3$. The K factor involved in the DE/current-to-rand/1 is sampled from $[0, 1]$ with a uniform probability. The crossover rate values are initialized to 0.5 for each strategy, and then updated on every generation of the learning period by a random value sampled from a normal distribution $N(\mu, \sigma)$ where $\mu = CR_i^k$ and $\sigma = 0.1$. During the same period, the values of CR_i^k which lead to offsprings bettering their parents are saved for each individual and each mutation scheme. At the end of the learning period, CR_i^k is replaced by the the median of those saved CR_i^k values; CR_i^k continues to be updated as before during the remainder of the optimization process.

3.6.4 Variable Population Sizes

Since the variability of the search moves depends on the individuals present in the population, acting upon the size of the population can affect the performance

of the Differential Evolution. Stagnation may be prevented by focusing the search in progressively smaller search spaces. The search is highly explorative during the early stages of the process, making use of a large population, and becomes progressively exploitative by reducing the size of the population and thus narrowing the searched region.

In the Differential Evolution with Population Size Reduction (Brest and Maučec, 2008), the algorithm starts with an initial population size S^1_{pop}, a total budget of fitness evaluations T_b and a number of stages N_s. The total budget is divided into N_s periods, each of which is characterized by a population size of S^k_{pop}, with $k = 1, \ldots, N_s$. A number of generations N^k_g is computed for each period by

$$N^k_g = \left| \frac{T_b}{N_s S^k_{pop}} \right| + r_k$$

where $r_k \geq 0$ takes a non-null value when T_b is not divisible by N_s. At the end of every period except the last one, the population size is halved. Therefore, $S^{k+1}_{pop} = S^k_{pop}/2$ for $k = 1, \ldots, N_s - 1$. The selection of the individuals surviving the reduction of the population is done by separating the population into two unsorted lists of equal sizes and comparing the fitnesses of the two individuals located at the same index in both lists; the best individual of the two is kept in the new, smaller population.

In the algorithm presented in Teo (2006), the population size, as well as the other control parameters, are self-adaptive: these parameters are appended to the genome of the individual and updated using a difference vector and a crossover, as are the individual's regular genes. It must be noted however that the algorithm described in the paper does not follow the structure of the original Differential Evolution, since it employs a scale factor of 1 and perturbs the offspring prior to selection by adding to each component a random value sampled in a normal distribution.

4 DIFFERENTIAL EVOLUTION WITH STRUCTURED POPULATION

In order to increase the execution speed of an algorithm, one can consider using multiple processors to execute parts of the algorithm in parallel. Alba and Tomassini (2002) presents three manners in which parallelism can be achieved: master-slave, coarse-grained and fine-grained structures. In the master-slave approach, the master controls the logic of the algorithm and distributes only the evaluation of the fitness function to the slaves; the algorithm does not require any structural modification and, provided that the time required to evaluate the fitness of one individual is greater than the time required to communicate with the slave, the execution time is reduced. Coarse and fine-grained structures are very different since they imply the modification of the the algorithm in order to assign one *subpopulation* to each processor of the parallel computer; each processor also runs its own instance of the algorithm, and processors are allowed to communicate with each other. The difference between coarse and fine-grained resides in the size, number and *topology* of these subpopulations: coarse-grained employ few subpopulations of quite large size, while fine-grained are composed of many subpopulations of reduced size, possibly limited to one single individual. In both cases, the different processors are organized into a given topology and are allowed to exchange information. Hybrids between master-slave, coarse- and fine-grained structures are also possible, for example by using a master-slave implementation of the instance running at each node of a coarse-grained algorithm.

Algorithms employing a structured population are not however required to be executed on parallel computers: the instructions which would be executed in parallel can be serialized and executed on a single processor. This implementation requires more time to execute than the parallel one, but it highlights the structured population not anymore as a manner to reduce the execution time but rather as a new variation of the original algorithm. The serialization of the algorithm also removes the synchronization problems which occurs on parallel computers when the different processors need to communicate with each other and where the need may arise for the initiator of a communication to wait for the acceptor to complete its current task before being able to respond.

4.1 Differential Evolution Algorithms with Structured Populations

4.1.1 Parallel Differential Evolution

The problem of parallelization for Differential Evolution schemes has been studied in Tasoulis et al. (2004) through an experimental analysis, and an algorithm, namely Parallel Differential Evolution has been proposed.

The original Parallel Differential Evolution implementation uses the Parallel Virtual Machine, allowing multiple computers (called *nodes*) to be organized as a cluster and exchange arbitrary messages. The algorithm is structured around one master node and N subpopulations running each on one node, and organized as a unidirectional ring, as illustrated in Figure 6. It must be noted that although the logical topology is a ring which does not contain the master node, the actual topology is a star, where all communications (i.e., the migrations of individuals from one subpopulation to another) are passing through the master.

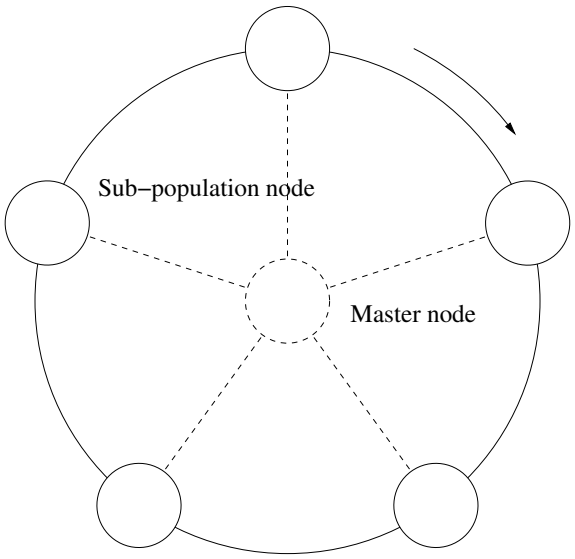


FIGURE 6 Unidirectional ring topology in the Parallel Differential Evolution algorithm

Each subpopulation runs a regular Differential Evolution algorithm while the master node coordinates the migration of individuals between the subpopulations. On each generation, the subpopulation has a given probability to send a copy of its best individual to its next neighbor in the ring. Algorithm 4 on the next page describes the behaviors of both the master node and the subpopulations in more details.

The Differential Evolution variant run by each subpopulation is the same across all the subpopulations. In Tasoulis et al. (2004), six mutation strategies have been compared, namely DE/best/1, DE/rand/1, DE/cur-to-best/1, DE/best/2, DE/rand/2 described in Section 3.1 on page 29, as well as the trigonometric operator described in Section 3.6.2 on page 44. Each strategy is used with different values of the migration constant ϕ and compared over seven test functions whose

At the master node:

spawn N subpopulations, each one on a different processor
for each generation **do**
  receive an individual from each subpopulation
  for each received individual **do**
    if rand$(0, 1) < \phi$ **then**
      send the individual to the next subpopulation in the ring
    end if
  end for
  if the stop criterion for the objective function is met **then**
    send a termination signal to all the subpopulations
  end if
end for

At each subpopulation:

for each generation **do**
  perform a Differential Evolution step
  send a copy of the best individual to the master node
  if a migrated individual has been received **then**
    replace a random individual, different from the best, by this migrated individual
  end if
  if a termination signal has been received **then**
    terminate the execution
  end if
end for

ALGORITHM 4 Pseudo-code of the Parallel Differential Evolution algorithm for both the master node and a subpopulation

dimensions vary between 2 and 30. The results show that DE/best/1 is the most efficient mutation strategy and quite stable across different values of ϕ, whereas the results of DE/rand/1 are average and quite unstable when ϕ varies.

4.1.2 Island-Based Distributed Differential Evolution

A distributed Differential Evolution, namely Island Based Distributed Differential Evolution has been proposed in Apolloni et al. (2008). The algorithm is a modified version of the Parallel Differential Evolution described in Section 4.1.1 on page 50. The algorithm is described in a generic way, presenting a population $\mathcal{P}$ structured in m subpopulations P_p of n_p individuals. The size of $\mathcal{P}$ is noted $N = \sum_{i=1}^{n} n_i$. The migration policy is then defined as a five-tuple $\mathcal{M} = (\gamma, \rho, \phi_s, \phi_r, \tau)$. $\gamma \in \mathbb{N}$ is the number of generations between two migrations, $\rho \in \mathbb{N}$ is the number of individuals which are migrated from a subpopulation during each migration, ϕ_s is the selection function which, applied to a subpopulation, returns the migrating individuals, ϕ_r is the replacement function that selects the individuals to be replaced by the immigrants in the receiving subpopulation, and $\tau : \mathcal{P} \rightarrow 2^{\mathcal{P}}$ is the topological model, which selects what subpopulation can send to (or receive from) what other subpopulation. Algorithm 5 describes the algorithm as pseudo-code.

```
initialize(P_p)
while the stopping condition is not met do
  perform a Differential Evolution step
  if the last migration was γ generations ago then
    for each of the ρ individuals to send do
      v'^i_g ← φ_s(P_p)
      send v'^i_g to P_j chosen by τ
    end for
  end if
  {** Asynchronous communication **}
  while individuals are arriving do
    receive v'^i_g from P_j
    replace an individual chosen from φ_s(P_p) by v'^i_g
  end while
end while
```

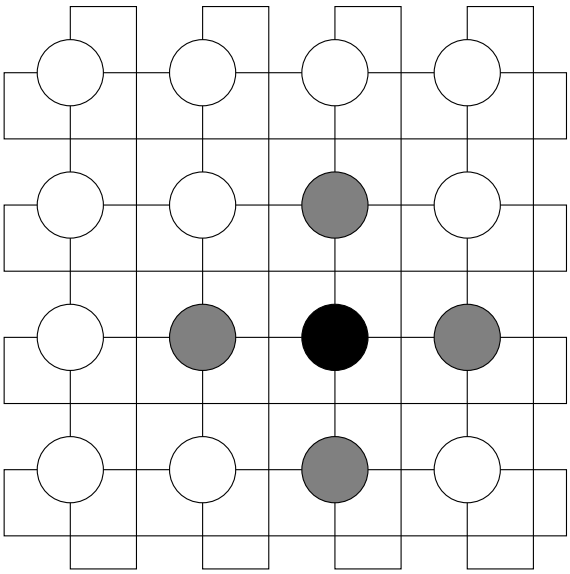
ALGORITHM 5 Pseudo-code of the Island-Based Distributed Differential Evolution algorithm for subpopulation P_p

For the actual experiments, the population size N is set to 20, and the population is divided into two subpopulations of 10 individuals in one experiment, and into four subpopulations of 5 individuals in a second experiment. The migration parameters are set to $\gamma = 100$, $\rho = 1$, the functions ϕ_s and ϕ_r are defined to randomly select an individual, and the topology τ is a unidirectional ring, very similar to the logical topology used by Parallel Differential Evolution (see Section 4.1.1 on page 50). The mutation strategy for the Differential Evolution is DE/rand/1, and

the algorithm is tested on 25 different test functions in 30 and 50 dimensions, for a total of 50 test functions.

4.1.3 Distributed Differential Evolution

In order to solve an image registration problem, a distributed Differential Evolution has been proposed in Falco et al. (2007a,b,c). This algorithm differs from the Parallel Differential Evolution and the Island-Based Distributed Differential Evolution by the topology it uses. Instead of a unidirectional ring, the Distributed Differential Evolution uses a locally connected topology, where each node is connected to μ other nodes. Figure 7 represents such a topology where the nodes are arranged in a mesh folded into a torus.



The grey discs represent the neighbors of the black disc.

FIGURE 7 Torus topology in the Distributed Differential Evolution algorithm

In this configuration $\mu = 4$, i.e., each node (such as the black disc in the figure) has exactly four nearest neighbors (represented by the four grey discs). In the Distributed Differential Evolution, each node represents one processor running a Differential Evolution algorithm with a DE/rand/1 mutation strategy on a subpopulation. Every M_I generations (the migration interval), each subpopulation is allowed to exchange S_I (the migration rate) individuals with its nearest neighbors. In the experimental setup, each node sends a copy of its best individual to its neighbors. Algorithm 6 describes the algorithm as pseudo-code.

The Distributed Differential Evolution also makes use of a master node, the role of which is to collect the best solutions found in each subpopulation and to present the results to the user.

```
initialize the subpopulation
while the stopping condition is not met do
  perform a Differential Evolution step
  if the last migration was M_I generations ago then
    send a copy of the best individual to each neighbor
  end if
  if there are incoming individuals then
    replace the worst S_I × μ individuals by the S_I × μ incoming ones
  end if
end while
```

ALGORITHM 6 Pseudo-code of the Distributed Differential Evolution algorithm at a subpopulation

4.1.4 Randomly Connected Topologies

The ring and torus topologies employed by the algorithms described above do not represent all the possible topologies used in designing distributed algorithms. Although Zaharie and Petcu (2004) has not been studied with as much attention as the three algorithms above, it is worth mentioning its use of a random topology, where the subpopulations are not considered to be connected to each other but rather share a common communication medium. After a given number of generations, each subpopulation is selecting a random other subpopulation and each individual in the first subpopulation has a given probability to be exchanged with an individual in the other subpopulation. This process is then repeated until the end of the algorithm.

A similar topology is used in **PI**, but only the individuals presenting the best performance in the subpopulation under consideration are selected for migration to a randomly chosen subpopulation. The incoming individuals are merged with the subpopulation's own individuals and only the best ones are retained, in a fashion similar to the survivor selection process of the $(\mu + \lambda)$ Evolution Strategy algorithm described in Section 2.2.1 on page 23. It must be noted that **PI** additionally differs from the other articles presented in this work by many aspects. The problem under study is the optimization of an artificial player based on a neural network, for a game involving an important part of randomness. The problem was therefore noisy and computationally intensive, leading to the the development of a truly distributed algorithm that was run in parallel on several computers. This problem however sparked the author's interest for parallel optimization algorithms and lead to the work described in the next chapter.

5 BUILDING UPON STRUCTURED POPULATION ALGORITHMS

Considering a given topology for a distributed Differential Evolution, one can attempt at enhancing its performance by modifying some parts of the basic algorithm, in a fashion similar to the numerous variants of the original Differential Evolution presented in Section 3.6 on page 42. Each of the articles presented in this work, with the exception of **PI**, presents such variations, which can be organized into two categories: the variations that act on the subpopulations of a distributed Differential Evolution, changing its size or the individuals it contains, and the variations that modify the operators which are at the core of the Differential Evolution's algorithm. The algorithm presented in article **PVII** having been specifically designed for large scale problems, it is described in a separate section. It must also be noted that the budget of fitness evaluations allotted to the algorithms in Articles **PII** to **PVI** are considerably shorter than the ones used in **PVII**.

5.1 Test Framework

Articles **PII** to **PVI** form a common set of algorithmic modules that can be added to Differential Evolution algorithms with structured population. These modules have been implemented within at least the Parallel Differential Evolution, and tested on a common set of continuous functions, using similar parameter settings. Table 1 summarizes these functions. In shorter articles, only a subset of these functions has been considered. Many of these test functions being additionally separable, they can be optimized one component at a time, making them potentially easier for algorithms searching along the axes of the problem space. To make the problems more difficult, rotated versions of some of these functions have been used as well. The rotation is performed by multiplying the input vector by a given, random orthonormal matrix. The test function is then applied to the rotated input vector.

The optimization algorithms presented below being stochastic processes, one cannot run them only once on a test function to measure their performance: the

TABLE 1 Test Problems

Test Problem	**Function**	**Decision Space**
Ackley	$-20 + e + 20\exp\left(-\frac{0.2}{n}\sqrt{\sum_{i=1}^{n} x_i^2}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i)x_i\right)$	$[-1,1]^n$
Alpine	$\sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1x_i\rvert$	$[-10,10]^n$
Axis-parallel hyper-ellipsoid	$\sum_{i=1}^{n} ix_i^2$	$[-5.12,5.12]^n$
DeJong	$\lVert x\rVert^2$	$[-5.12,5.12]^n$
DropWave	$-\frac{1+\cos\left(12\sqrt{\lVert x\rVert^2}\right)}{\frac{1}{2}\lVert x\rVert^2+2}$	$[-5.12,5.12]^n$
Griewangk	$\frac{\lVert x\rVert^2}{4000} - \prod_{i=0}^{n}\cos\frac{x_i}{\sqrt{i}} + 1$	$[-600,600]^n$
Michalewicz	$-\sum_{i=1}^{n}\sin x_i\left(\sin\left(\frac{i\cdot x_i^2}{\pi}\right)\right)^{20}$	$[0,\pi]^n$
Pathological	$\sum_{i=1}^{n-1}\left(0.5 + \frac{\sin^2\left(\sqrt{100x_i^2+x_{i+1}^2}-0.5\right)}{1+0.001*\left(x_i^2-2x_ix_{i+1}+x_{i+1}^2\right)^2}\right)$	$[-100,100]^n$
Rastrigin	$10n + \sum_{i=0}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12,5.12]^n$
Rosenbrock valley	$\sum_{i=1}^{n-1}\left(100\left(x_{i+1}-x_i^2\right)^2 + (1-x_i)^2\right)$	$[-2.048,2.048]^n$
Schwefel	$\sum_{i=1}^{n} -x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	$[-500,500]^n$
Sum of powers	$\sum_{i=1}^{n}\lvert x_i\rvert^{i+1}$	$[-1,1]^n$
Tirronen	$3\exp\left(-\frac{\lVert x\rVert^2}{10n}\right) - 10\exp\left(-8\lVert x\rVert^2\right)$ $+\frac{2.5}{n}\sum_{i=1}^{n}\cos\left(5\left(x_i + (1+i \bmod 2)cos\left(\lVert x\rVert^2\right)\right)\right)$	$[-10,5]^n$

result of a single run could be exceptionally "good" or conversely exceptionally "bad" and not reflect the actual performance of the algorithm. The current practice is thus to run each algorithm repeatedly on a test function, which produces a set of fitness values; the comparison of the performance of two algorithms when applied to that test function is therefore a non-trivial task. To that effect, multiple complementary approaches have been employed in the articles presented below. The first approach is to compare the average values returned by the algorithms over their multiple runs; those are therefore presented in a table, which contains additionally the corresponding standard deviations. Averages alone are however not significant enough, especially if the ranges of the samples are overlapping to some extent. Statistical tests such as Student's t-test or Wilcoxon's rank-sum test are therefore applied to the results; these tests attempt at determining, with a given confidence level, if the two statistical distributions, from which the sets of results produced by the two algorithms have been sampled, are identical. In the case the test rejects the null hypothesis e.g., their equality, one can conclude that one algorithm is significantly outperforming the other. As the tests above are applied to results obtained from one function only, a post-hoc test such as Wilcoxon's signed-rank test or Holm's procedure can be applied to compare the performance of several algorithms over the whole benchmark they have been applied to. Descriptions of those tests and detailed discussions about the reasons they are needed can be found e.g., in García et al. (2008a,b). Finally, the convergence speed of the algorithms is analyzed (except in **PVII** where it was not required) through the so-called Q-test, which measures the proportion of runs where the algorithm reaches a given threshold. Algorithms which never manage to reach the threshold on some functions are said to be unstable, since their efficiency is not guaranteed, and considered to be performing less well than the ones consistently reaching the threshold.

5.2 Population Modification

The population of an evolutionary algorithm can be modified in at least two ways: by changing its size, or by replacing one individuals with another one. The latter differs from the normal way Differential Evolution works in that the individuals which is introduced is not the product of the algorithm itself, but is alien to the population. These two mechanisms are applied in the examples below.

In **PII**, the distributed algorithms described in Sections 4.1.1, 4.1.2 and 4.1.3 are compared to a new algorithm named Distributed Differential Evolution with Explorative-Exploitative Population Families. In this algorithm, two families of populations are evolving independently at first, one family running the Parallel Differential Evolution described in Section 4.1.1, while the other runs independent Differential Evolution with Population Size Reduction algorithms (see Section 3.6.4 on page 48). After a period of observation, the best individuals from the second family of subpopulations are injected into the subpopulations of the first family,

in the prospect that this individual would allow the subpopulation new search moves that would lead it closer to the global optimum.

In the Parallel Random Injection Differential Evolution **PIV**, random, new individuals are injected at random intervals into one randomly chosen subpopulation of a Parallel Differential Evolution, replacing one randomly selected individual (an exception is made of the individual presenting the best performance, which is never replaced). Although this new individual is not expected to be improving on the ones in the subpopulation, it modifies the spatial distribution of the subpopulation and therefore allows the mutation operator to produce new search moves and prevents the algorithm from stagnating.

Results presented in both articles **PII** and **PIV** show that indeed a slight perturbation of the algorithm's population improves its performance compared to the reference algorithms where no such perturbation is performed.

5.3 Operator Modification

The operators which are here referred to are the the mutation and the crossover. Section 3.6 on page 42 presents several examples of modified operators in the Differential Evolution which were a source of inspiration for the following algorithms.

In the F-Adaptive Control Parallel Differential Evolution **PIII**, a Parallel Differential Evolution (see 4.1.1 on page 50) is setup with, in each subpopulation, random values of the scale factor. When a subpopulation's best individual is migrated to another subpopulation, a slightly perturbed value of the scale factor is migrated along with that individual. This way, a "good" scale factor can propagate and enhance the efficiency of the optimization process. The value which is migrated is modified by adding a normally distributed random value with a small standard deviation, allowing the modified value to be different from the original value but without varying to much. This mechanism, comparable to the mutation operation in e.g., Evolutionary Programming (see 2.2.1 on page 21), allows to optimize the value of the scale factor in an attempt to adapt this parameter to the function being optimized.

The Parallel Differential Evolution with Endemic Random Control Parameters **PV** is based on the Parallel Differential Evolution algorithm. In each subpopulation, both control parameters are initialized to random values. During the course of the optimization, the values of the control parameters are updated to a new random value based on a given probability. One unexpected finding was that the repeated updates of the crossover rate are detrimental to the performance or the algorithm, but repeated updates of the scale factor are beneficial.

The Parallel Component Decomposition Differential Evolution described in **PVI** is aimed at large scale optimization (see also Section 5.4 on the next page for a brief introduction to the challenges posed by large scale problems). When confronted to problems with a large number of variables, most classical algorithms

fail. Potter and De Jong (1994) introduces the idea of restricting the optimization to only some of the variables of the problem instead of attempting to optimize them all at once. This idea is applied to a Parallel Differential Evolution by assigning the Differential Evolution algorithms running on each subpopulation a subset of the problem's variables. This is implemented by modifying the crossover function to limit its action to the subset assigned to the subpopulation. Individuals migrating from population to population are therefore optimized on different subsets, eventually being optimized over the whole search space. One would expect such a mechanism to perform well on separable functions i.e., functions which can be optimized component by component, but less well on non-separable functions. The results presented in this article however show that even on rotated versions of separable functions (the rotation removing the function's separability property), the algorithm remains competitive.

The experimental results presented in Articles **PIII** and **PV** show that variable scale factors generally improve the performance of a Parallel Differential Evolution. Article **PV** moreover indicates that the crossover rate is more sensitive to repeated variations and although different crossover rates assigned to different subpopulations of a Parallel Differential Evolution are beneficial to the optimization process compared to a single, common value, its update in the course of the optimization process is detrimental. However, modifications of the crossover mechanism itself, presented in Article **PVI**, without modifying the value of the corresponding control parameter, are shown to be beneficial compared to the reference Parallel Differential Evolution.

5.4 Hybrid Approach for Large-Scale Optimization

Optimization algorithms suffer from the so-called "curse of dimensionality", i.e., that the problem to optimize becomes exponentially more difficult when its dimensionality increases. This is easily illustrated with a simple function of the type $f(x) = \sum_1^n x_i^2$ with $x \in [-0.5, 0.5]^n$: if one considers, when $n = 1$, that the area located "near" the global optimum is $[-0.05, 0.05]$ (one tenth of the whole interval), one can deduce that in n dimensions, that area represents 10^{-n} of the whole search space. When, e.g., $n = 1000$ one can easily understand that the "interesting" area represents only a minute fraction of the whole space that must be explored by the optimization algorithm. Large-scale optimization therefore aims at developing algorithms able to optimize problems with a large number of dimensions.

In **PVII**, the Shuffle Or Update Parallel Differential Evolution algorithm is applied to a benchmark of nineteen large-scale test functions, which was provided by the guest editor of the special issue of the journal it was published in, and is compared to three other, well understood optimization algorithms (also imposed by the guest editor): the Differential Evolution using the DE/rand/1/exp variant, the Real-coded CHC algorithm (Eshelman, 1991; Eshelman and Schaffer, 1993)

and the G-CMA-ES algorithm (Auger and Hansen, 2005). The Shuffle Or Update Parallel Differential Evolution algorithm is based on a structured population and belongs to both categories described in 5 on page 55. Each subpopulation runs a Differential Evolution where the scale factor is updated on each generation, with a given probability, to a new, random value. Additionally, on each generation, with a given probability, all the individuals of the subpopulations are "shuffled", i.e., gathered and randomly redistributed to the subpopulations. This algorithm uses, in a structured population Differential Evolution (which, lacking the migration mechanism, is not a Parallel Differential Evolution), a combination of a mechanism modifying the population and the modified mutation operator already presented in **PV**.

Both Wilcoxon's signed-rank test and Holm's procedure were applied to the experimental results, and although the former concludes that the Shuffle Or Update Parallel Differential Evolution outperforms the reference algorithms in all dimensionalities, the latter statistical test only concludes that its results, although slightly better, are not significantly different from the second best algorithm's.

6 CONCLUSION

The Differential Evolution is a high performance population-based optimization algorithm suitable for a wide range of continuous problems. One of the core principles of the algorithm is to search for improved solutions by producing mutant individuals based solely on individuals found in the algorithm's population: one randomly chosen individual is perturbed by the scaled differences between two other, randomly chosen individuals. The Differential Evolution however contains an inherent flaw which can cause it to fail at improving upon existing solutions in two ways: stagnation and premature convergence.

Stagnation is a condition occurring when the algorithm is unable to produce a mutant that outperforms its parent. Since the mutants are produced solely from material found within the population, the number of different mutants that can be produced from a given population is limited by the population itself. When no combination of individuals produces a mutant able to outperform its parent, the population is unable to evolve and thus fails at closing towards the global optimum.

Premature convergence on the contrary is a condition happening when a population converges towards a local optimum and is unable to escape from it. Since mutants are produced from the scaled difference between two individuals in the population, the largest step that can occur is limited on one hand by the difference between the two individuals which are the farthest from each other and on the other hand by the value of the scale factor. When the population converges, its individuals move nearer to each other; the size of the largest step thus diminishes and, when too small, prevents the population from escaping the local minimum it may have converged to since any move away from it produces a solution worse than the current ones.

Various research directions have been explored in order to enhance the Differential Evolution and prevent stagnation and premature convergence. Among those, one can mention the use of structured populations i.e., a set of interacting subpopulations, instead of monolithic ones, and various attempts at modifying the search logic (i.e., the mutation or the crossover operators) or at modifying the population itself. With the exception of the first article presented in this thesis,

which, although it employs a distributed Evolution Strategy and not a Differential Evolution, started the author's interest in distributed optimization algorithms, the other six articles describe algorithmic improvements applied to one or more algorithms with a structured-population based on the Differential Evolution. These improvements can be classified into two categories: those modifying the population, and those modifying the Differential Evolution's search logic. Modifications of the population is performed by injecting into a population a partially optimized or a completely random individual, or by randomly redistributing the individuals between subpopulations. Modifications of the mutation operator consist in changing the value of the scale factor during the course of the optimization process, either randomly or by inheriting its value from another subpopulation. Finally, one article describes a modified crossover operator that limits its action to a subset of the problem's variables; different subpopulations then optimize different subsets and a migration mechanism allows individuals to see their whole set of variables to be optimized.

Although different at first glance, both categories of improvements aim towards a single goal: preventing the algorithm from stagnating or from converging prematurely by acting either on the pool of individuals used for producing mutants, or by adding a stochastic component to the mutation operator. These two modifications allow the Differential Evolution to produce solutions which would otherwise have been inaccessible, and thus increase the algorithm's probability of improving upon its current solutions.

A remarkable learning that can be drawn from this work is that the Differential Evolution can be improved and adapted to large-scale problems without resorting to complex memory mechanisms or awkward-to-determine learning periods: conceptually simple mechanisms that add, in a controlled way, a stochastic component to the algorithm allow to significantly improve its performance.

YHTEENVETO (FINNISH SUMMARY)

Tämän työn nimi on "Rinnakkainen globaalioptimointi. Populaation rakenteen määrittäminen differentiaalievoluutiossa". Differentiaalievoluutio on erittäin suorituskykyinen globaalioptimointimenetelmä, joka soveltuu moniin jatkuviin tehtäviin. Yksi algoritmin pääperiaatteista on populaation yksilöiden käyttäminen seuraavan populaation tuottamiseen: yhtä satunnaisesti valittua yksilöä muokataan kahden muun satunnaisesti valitun yksilön välisillä skaalatuilla eroilla. Rakenteensa takia differentiaalievoluutio kärsii usein ennenaikaisesta konvergenssista, sekä ns. stagnaatiosta, joka estää globaalin minimin löytymisen.

Stagnaatiolla tarkoitetaan sellaista menetelmän tilaa, jossa kaikki mahdolliset jälkeläiset ovat populaation jäseniä huonompia ja menetelmä ei voi edetä. Tähän tilaan saatetaan päätyä, sillä uusi sukupolvi luodaan täysin edellisen perusteella. Näin ollen populaation koko asettaa ehdottoman ylärajan sille, mitä pisteitä menetelmä voi kokeilla.

Ennenaikainen konvergenssi on tila, jossa populaatio on saavuttanut kokonaisuudessaan lokaalin optimin, eikä pysty pakenemaan siitä. Koska yritepisteet muodostetaan edellisen populaation muodon perusteella, suurin mahdollinen siirtymä määräytyy populaation hajonnan ja menetelmän skaalaparametrin yhteisvaikutuksesta. Suorituksen aikana populaation hajonta pienenee ja alue, joka on menetelmän saavutettavissa, kutistuu. Jos menetelmä konvergoituu lokaalin minimin vaikutuksesta liian pitkälle ja väärälle alueelle, se ei pysty tuottamaan yritepisteitä, jotka eivät johtaisi samaan minimiin.

Tässä työssä differentiaalievoluutiota on pyritty tehostamaan tarkastelemalla menetelmää stagnaation ja ennenaikaisen konvergenssin näkökulmasta. Työssä on tutkittu useita eri parannuksia menetelmään, kuten rakenteisia populaatioita sekä suoria hakulogiikan muunnoksia.

Väitöskirjan ensimmäinen artikkeli käsittelee differentiaalievoluution sijasta hajautettua evoluutiostrategiaa, ja se on otettu mukaan, koska se sai kirjoittajan alun perin kiinnostumaan hajautetuista optimointialgoritmeista. Muut kuusi artikkelia kuvaavat algoritmisia parannuksia rakenteiseen populaatioon perustuviin differentiaalievoluutiomenetelmän muunnelmiin. Esitetyt parannukset voidaan jakaa kahteen luokkaan: populaation kokoa ja tilaa suoraan muokkaavat, ja hakulogiikkaa muokkaavat parannukset.

Hakulogiikan muutokset koostuvat skaalauskertoimen arvon muuttamisesta optimointiprosessin aikana joko satunnaisesti tai johtamalla se toisesta osasta rakenteista populaatiota. Viimeinen artikkeli kuvaa menetelmän muunnoksen, joka käyttää rakenteista populaatiota ongelman jakamiseksi osiin. Jokainen alipopulaatio pyrkii ratkaisemaan tehtävästä oman komponenttinsa ja kommunikoimaan ratkaisunsa eteenpäin.

Tässä työssä esitettyjen parannusten ansiosta differentiaalievoluutio voi tuottaa ratkaisuja, jotka olisivat muuten saavuttamattomissa, ja näin tuottaa nykyistä parempia ratkaisuja.

Differentiaalievoluutiota voidaan siis parantaa ja soveltaa suuren mittakaa-

van ongelmiin tarvitsematta turvautua monimutkaisiin muistimekanismeihin tai vaikeasti määritettäviin oppimisjaksoihin. Algoritmin tehokkuutta voidaan merkittävästi parantaa käsitteellisesti yksinkertaisten mekanismien avulla.

REFERENCES

Alba, E. & Tomassini, M. 2002. Parallelism and evolutionary algorithms. IEEE Transactions on Evolutionary Computation 6 (5), 443–462.

Apolloni, J., Leguizamón, G., García-Nieto, J. & Alba, E. 2008. Island based distributed differential evolution: An experimental study on hybrid testbeds. In Proceedings of the IEEE International Conference on Hybrid Intelligent Systems, 696–701.

Auger, A. & Hansen, N. 2005. A restart CMA evolution strategy with increasing population size. In 2005 IEEE Congress on Evolutionary Computation, 1769–1776.

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

Banzhaf, W., Nordin, P., Keller, R. E. & Francone, F. D. 1998. Genetic programming – An introduction on the automatic evolution of computer programs and its application. Morgan Kaufmann.

Bonabeau, E., Rodrigo, M. & Theraulaz, G. 1999. Swarm intelligence: from natural to artificial systems. Oxford University Press.

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

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

Bäck, T. & Schwefel, H.-P. 1995. Evolution strategies I: variants and their computational implementation. In J. Periaux & G. Winter (Eds.) Genetic algorithms in engineering and computer science. Wiley.

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

Caruana, R. A., Eshelman, L. J. & Schaffer, J. D. 1989. 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, Vol. 1. Morgan Kaufmann, 750–755.

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

Chakraborty, U. K. (Ed.) 2008. Advances in Differential Evolution, Vol. 143. Springer. Studies in Computational Intelligence.

Chiong, R., Neri, F. & McKay, R. I. 2010. Nature that breeds solutions. In R. Chiong (Ed.) Nature-Inspired Informatics for Intelligent Applications and Knowledge Discovery: Implications in Business, Science, and Engineering. IGI Global, 1–24.

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

Das, S., Abraham, A., Chakraborty, U. K. & Konar, A. 2009. Differential evolution with a neighborhood-based mutation operator. IEEE Transactions on Evolutionary Computation 13 (3), 526–553.

Dawkins, R. 1976. The selfish game. Oxford University Press.

Dorigo, M. 1992. Optimization, learning and natural algorithms. Politecnico de Milano. Ph. D. Thesis.

Eberhart, R. C. & Kennedy, J. 1995. A new optimizer using particle swarm theory. In Proceedings of the Sixth International Symposium on Micromachine and Human Science, 39–43.

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

Eshelman, L. 1991. The CHC adaptive search algorithm: how to have safe search when engaging in nontraditional genetic recombination. In G. Rawlin (Ed.) Foundations of Genetic Algorithms 1. San Mateo, CA: Morgan Kaufmann, 265–283.

Eshelman, L. J. & Schaffer, J. D. 1993. Real-coded genetic algorithm and interval schemata. In Foundation of Genetic Algorithms, 187–202.

Eshelman, L. J., Caruana, R. A. & Schaffer, J. D. 1989. Biases in the crossover landscape. In Proceedings of the third international conference on Genetic algorithms. Morgan Kaufmann, 10–19.

Falco, I. D., Cioppa, A. D., Maisto, D., Scafuri, U. & Tarantino, E. 2007a. Satellite image registration by distributed differential evolution. In Applications of Evolutionary Computing, Vol. 4448. Springer. Lectures Notes in Computer Science, 251–260.

Falco, I. D., Maisto, D., Scafuri, U., Tarantino, E. & Cioppa, A. D. 2007b. Distributed differential evolution for the registration of remotely sensed images. In Proceedings of the IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 358–362.

Falco, I. D., Scafuri, U., Tarantino, E. & Cioppa, A. D. 2007c. A distributed differential evolution approach for mapping in a grid environment. In Proceedings of the IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing, 442–449.

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

Feoktistov, V. 2006. Differential Evolution in Search of Solutions. Springer, 83–86.

Fogel, L. J., Owens, A. J. & Walsh, M. J. 1965. Artificial intelligence through a simulation of the evolution. In A. M. Maxfield & L. J. Fogel (Eds.) Biophysics and cybernetics systems. Spartan Book Co: Washington, DC, 131–156.

Fogel, L. J., Owens, A. J. & Walsh, M. J. 1966. Artificial intelligence through simulated evolution. John Wiley & Sons, Inc.

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

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

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

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

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

Holland, J. 1962. Outline for a logical theory of adaptive systems. Journal of the Association for Computing Machinery 3, 297–314.

Holland, J. H. 1992. Adaptation in Natural and Artificial Systems. MIT Press.

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

Ishibuchi, H., Yoshida, T. & Murata, T. 2003. Balance between genetic search and local search in memetic algorithms for multiobjective permutation flow shop scheduling. IEEE Transactions on Evolutionary Computation 7 (2), 204-223.

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

Koza, J. R. 1992. Genetic programming: On the programming of computers by means of natural selection. The MIT Press.

Koza, J. R. 1994. Genetic programming II. The MIT Press.

Krasnogor, N., Blackburne, B., Burke, E. & Hirst, J. 2002. Multimeme algorithms for proteine structure prediction. In Proceeding of Parallel Problem Solving in Nature VII. Lecture Notes in Computer Science, Springer-Verlag.

Krasnogor, N. 2002. Studies in the Theory and Design Space of Memetic Algorithms. University of West England. Ph. D. Thesis.

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

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

Langdon, W. B. & Poli, R. 2001. Foundations of genetic programming. Springer Verlag.

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

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

Luke, S. & Spector, L. 1997. A comparison of crossover and mutation in genetic programming. In J. K. et al. (Ed.) Genetic programming 1997: Proceedings of the 2nd Annual Conference. San Francisco: Morgan Kaufmann, 240–248.

Mininno, E. & Neri, F. 2010. Estimation distribution differential evolution. In Applications of Evolutionary Computation, Vol. 6024/2010. Springer Berlin / Heidelberg. Lecture Notes in Computer Science, 522-531.

Moscato, P. & Norman, M. 1989. A Competitive and Cooperative Approach to Complex Combinatorial Search.

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

Neri, F., Tirronen, V. & Kärkkäinen, T. 2009. Enhancing differential evolution frameworks by scale factor local search – part II. In Proceedings of the IEEE Congress on Evolutionary Computation, 118–125.

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

Neri, F. & Mininno, E. 2010. Memetic compact differential evolution for cartesian robot control. IEEE Computational Intelligence Magazine 5 (2), 54–65.

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

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

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

Ong, Y. S., Lim, M. H., Zhu, N. & Wong, K. W. 2006. Classification of adaptive memetic algorithms: A comparative study. IEEE Transactions On Systems, Man and Cybernetics - Part B 36 (1), 141–152.

Potter, M. A. & De Jong, K. A. 1994. A cooperative coevolutionary approach to function optimization. In Proceedings of the Third Conference on Parallel Problem Solving from Nature. Springer-Verlag, 249–257.

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

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

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

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

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

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

Rall, L. B. 1981. Automatic Differentiation: Techniques and Applications, Vol. 120/1981. Springer Berlin / Heidelberg. Lecture Notes in Computer Science.

Rechenberg, I. 1971. Evolutionsstrategie - Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Technische Universität Berlin. Ph. D. Thesis.

Rudolph, G. 1996. Convergence of evolutionary algorithms in general search spaces. In Proceedings of the third IEEE conference on evolutionary computation. IEEE Press, 50–54.

Siarry, P. & Berthiau, G. 1998. Fitting of tabu search to optimize functions of continuous variables. International Journal for Numerical Methods in Engineering 40 (3), 2449–2457.

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

Syswerda, G. 1989. Uniform crossover in genetic algorithms. In J. Shaffer (Ed.) Proceedings of the third international conference on genetic algorithms. San Francisco: Morgan Kaufmann, 2–9.

Tagawa, K. 2009. A statistical study of the differential evolution based on continuous generation model. In Evolutionary Computation, 2009. CEC '09. IEEE Congress on, 2614–2621.

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

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

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

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

Zaharie, D. & Petcu, D. 2004. Adaptive pareto differential evolution and its parallelization. In Parallel Processing and Applied Mathematics, Vol. 3019. Lecture Notes in Computer Science, 261–268.

Zaharie, D. 2002. Critical values for control parameters of differential evolution algorithm. In R. Matuŝek & P. Oŝmera (Eds.) Proceedings of 8th International Mendel Conference on Soft Computing, 62–67.

Zhang, J. & Sanderson, A. 2007. JADE: Self-adaptive differential evolution with fast and reliable convergence performance. In Evolutionary Computation, 2007. CEC 2007. IEEE Congress on, 2251–2258.

ORIGINAL PAPERS

PI

FITNESS DIVERSITY PARALLEL EVOLUTION ALGORITHMS IN THE TURTLE RACE GAME

by

Matthieu Weber, Ville Tirronen and Ferrante Neri 2009

In Applications of Evolutionary Computing, volume 5484/2009 of *Lecture Notes in Computer Science*, pages 303–312

Fitness Diversity Parallel Evolution Algorithms in the Turtle Race Game

Matthieu Weber, Ville Tirronen, and Ferrante Neri

Department of Mathematical Information Technology,
University of Jyväskylä, P.O. Box 35 (Agora), FI-40014, Finland
{matthieu.weber, ville.tirronen, neferran}@jyu.fi

Abstract. This paper proposes an artificial player for the *Turtle Race* game, with the goal of creating an opponent that will provide some amount of challenge to a human player. *Turtle Race* is a game of imperfect information, where the players know which one of the game pieces is theirs, but do not know which ones belong to the other players and which ones are neutral. Moreover, movement of the pieces is determined by cards randomly drawn from a deck. The artificial player is based on a non-linear neural network whose training is performed by means of a novel parallel evolutionary algorithm with fitness diversity adaptation. The algorithm handles, in parallel, several populations which cooperate with each other by exchanging individuals when a population registers a diversity loss. Four popular evolutionary algorithms have been tested for the proposed parallel framework. Numerical results show that an evolution strategy can be very efficient for the problem under examination and that the proposed adaptation tends to improve upon the algorithmic performance without any addition in computational overhead. The resulting artificial player displayed a high performance against other artificial players and a challenging behavior for expert human players.

1 Introduction

In recent years, artificial game players have extensively been studied, see [1]. These games and thus the architecture of the artificial players can be classified into two groups: *complete information* games, when all players have the same information about the state of the game; *incomplete information* games, when each players has only an incomplete view of the game state.

Complete information games are also deterministic, thus a good player puts most of his efforts into foreseeing possible scenarios and movements the opponent may make in order to detect the most profitable move. Games of this kind have been widely studied by means of various artificial intelligence techniques. Some representative examples include from the simple tic-tac-toe [2] to the eminently complex go [3], [4], from classical chess and checkers [5] to the more recent sudoku [6].

In incomplete information games (see analogies and differences with the concept of imperfect information [7]), the player does not have a complete set of

information related to game state since, for example, he cannot know moves of the opponent, as in the classical case of the prisoner dilemma, or cannot predict moves the other players will make, as often happens in card games (this problem has been clear since the dawn of Artificial Intelligence, see [8]). Some examples of incomplete information games can be poker [9], bridge, backgammon, the ant wars [10]. In card games, each time a deck of cards is shuffled prior to the start of the game, a new game is settled. The overall process can, in this case, be seen as a system characterized by a state (i.e., what the players see on the table) plus some parameters (i.e., a deck of cards) which are affected by uncertainties.

This paper studies a complex game, namely *Turtle Race*, which is characterized by several uncertainties, including the unknown identity of the other players and proposes an artificial player for it. The core of this artificial player is a multi-perceptron neural network trained by means of a novel adaptive parallel evolutionary algorithm. This algorithm employs a multi-population and a refreshment of the individuals by means of migration controlled by a fitness diversity adaptation.

2 The Turtle Race Game

Schildkrötenrennen (*Turtle Race* in English) is a board game for two to five players by Reiner Knizia, initially published in 2004 by Winning Moves, Inc. Fig. 1 shows a picture of the game components. The game is composed of five wooden turtles in five different colors (red, green, blue, yellow and purple), five turtle tokens of the same colors, fifty-two cards and a board representing a linear track of ten squares. Each player is secretely assigned one color by randomly drawing one of the tokens, and all five turtles are placed on the first square of the board, called the *start square*. The goal of this game is to move one's turtle from the start square to the tenth square, called the *goal*. The cards are used for deciding how the turtles will be moved.

Fig. 1. Game components

The players are dealt five cards each, and the remaining cards are placed face down, forming the draw deck. Players take turns until at least one turtle

reaches the goal. On his turn, the player selects one card from his hand, moves one turtle based on the action symbolized on the card, and places the card face up on the discard pile. He then draws a new card from the draw deck. If the draw stack is empty, the discard stack is shuffled and replaces the draw deck.

Each card represents a turtle bearing a symbol. The color of the turtle determines which turtle will be moved, while the symbol defines in which direction and by how many squares it will move. Some cards represent a rainbow-colored turtle, meaning that the player must decide which turtle will be moved. The symbols are *plus*, *minus*, *double plus*, *arrow* and *double arrow*. *Plus*, *minus* and *double plus* respectively mean one square forward, one square backwards, and two square forward. The *arrow* and *double arrow* respectively mean one square forward and two squares forward, but can be applied only to the turtles which are the nearest to the start square, In the configuration depicted in Fig. 2 for example, *arrow* and *double arrow* cards can be applied only to turtle Y. The turtles bearing *arrow* or *double arrow* are always rainbow-colored. The cards are distributed as follows: 5 *plus*, 1 *double plus* and 2 *minus* for each color, 5 rainbow *plus*, 2 rainbow *minus*, 3 rainbow *arrow* and 2 rainbow *double arrow*.

Turtles can be stacked: as illustrated in Fig. 2, when moving one turtle, one also moves at the same time all turtles that are stacked on its back. When a turtle reaches a square already occupied by one or more stacked turtles, it is placed on top of that stack. The only exception is the start square, where the turtles are never stacked but placed side-by-side. The stacking mechanism allows a player, in some cases, to move his turtle even if he does not currently have a card of the right color in his hand, by moving another turtle situated below his own in the stack. Moreover, when playing an *arrow* or a *double arrow* card, if there is more than one turtle on the square nearest to the start, the player decides which one of these turtles he will move.

There are always five turtles in the game, even if there are less than five players. In this case, one or more turtles are *neutral*, i.e., not belonging to any player. When a turtle reaches the goal square, the game immediately ends and the players reveal their colors. It is possible that the turtle reaching the goal square is a neutral one. The rule for determining the winner is therefore as follows: the game winner is the player whose turtle is nearest to the goal square (possibly located exactly on this square) and is lowest in its stack. In other words, one considers the stacks of turtles from the goal rearward, and in each stack the turtles from bottom to top. The first non-neutral turtle is then the winner (see Fig. 3).

3 The Artificial Player

The core of the artificial player is a neural network, which associates a weight to each card held by the player. Rainbow-colored cards are decomposed into virtual single-colored cards, and a weight is associated to each one of them. The artificial player then sorts the cards according to their weights and the card with the lowest weight is selected to be played.

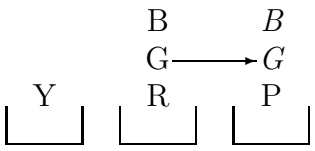


Fig. 2. When moving turtle G by one square forward, turtle B located on top of it moves at the same time.

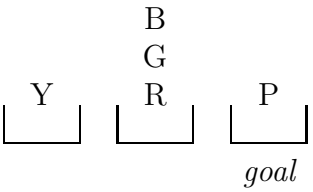


Fig. 3. To determine the winner, one examines P, R, G, and B in that order.

In order for the neural network to weigh a given card, the game logic considers the current game state, creates a copy of it and changes this copy according to the action depicted on the card. The neural network is then presented with these *before* and *after* game states and calculates the weight of that card based on this information.

The neural network chosen to be the core of the artificial player is a three-layer perceptron, using the hyperbolic tangent as the activation function [11]. This configuration has been chosen since it has shown a high performance with similar games, see [12] and [13]. The neural network is composed of nineteen neurons organized in three layers, has nineteen inputs and one output, and counts 325, real-number weights. Inputs of the neural network are the artificial player's own color, a synthetic representation of the game state before playing a given card and the representation of the game state after playing that card. Both representations have the same parameters:

- The position on the board of the artificial player's turtle, as an offset from the start square (integer number between zero and nine) and its position in the stack, as an offset from the bottom of the stack (integer number between zero and four). These two parameters give the neural network a global view of its position in the game.
- The number of turtles situated above and below the artificial player's within the stack (as integer numbers between zero and four).
- The number of turtles situated on the square preceding the artificial player's, i.e., nearer to the start square, the number of turtles situated on the square following that of the artificial player's, and the number of turtles situated on the second square after that of the artificial player's (integer numbers between zero and four).
- The number of turtles situated on the start square (integer number between zero and five).
- The number of squares between the artificial player's turtle and the goal square (integer number between one and nine).

The artificial player design consists of detecting a set of 325 weights of the neural network, ensuring a high performance in playing the turtle game.

It can be seen that except for its own color, the neural network is fed with no further information about the color of the other turtles. The reason is that

a player ignores the color of the various opponents as well as the color of the neutral turtles. This fact leads to the consequence that a player (the neural network in this case) makes a decision on the basis of those cards he can see (his own cards) and possible configurations which are visible on the board. Thus, it is clear that, due to the nature of the game, given a configuration of turtles on the track, success of the decision made by the artificial player is heavily influenced by parameters that cannot be controlled, i.e., cards held by the other players and the way the deck has been shuffled. consequently, evaluation of a playing strategy in a single match is not reliable since "good luck" and "bad luck" play an important role in this game. This phenomenon can be seen as noise unavoidably introduced into the system. Thus, in order to perform proper ranking of two players a certain amount of games are required and the quality of a player can be extracted by means of a statistical analysis on the success of the players under consideration.

Thus, the quality of playing strategy has been performed in this paper by means of the following method. Let a and b be two players and N the number of games played between a and b. If a is the playing strategy proposed by the aforementioned neural network, the objective function to be minimized is then defined as:

$$F(a,b) = 1 - \frac{V_a}{V_a + D_a} \tag{1}$$

where V_a is the number of victories of a against b, and D_a the number of defeats of a against b. The value of N has been fixed to 100, which has been found to be a good trade-off between statistical significance and speed of execution. It must be noted that $V_a + D_a$ is not necessarily equal to N, since there are cases where a neutral turtle reaches the goal square while both players' turtles are still on the start square. There is no rule in the game to determine the winner in such a situation.

The opponent player for the neural network is a random player, i.e., a player which randomly performs its choices during the game time. This choice is based on the assumption that the neural network will challenge a large amount of random players and eventually develop a strategy that allows a robust behavior against other players which have an actual playing strategy (e.g., human players). This assumption is limited to such games as Turtle Race which have an inherently large amount of randomness in their structure. This choice can be further justified by the theoretical results given in [14]. It is shown that a complex and expert player can be replaced by a simple random player which executes multiple games. This fact based on the property of the random player's average behavior, given a large enough number of trials, tends to asymptotically converge toward the behavior of an expert player. This property has already been used when developing so-called *Monte Carlo* players for various games, see [15]. Although not identical, the use of a random player as an opponent when training a neural network-based player is strongly correlated to its use in Monte Carlo players, which has been confirmed experimentally.

The problem studied in this paper is then the minimization of the fitness function F in eq. (1) dependent on the 325 weights which characterize neural

network a. These weights vary in a decision space $D =]-\infty, \infty[^{325}$. As shown above, the quality of a candidate neural network is identified by its success rate against 100 random players.

4 Fitness Diversity Parallel Evolutionary Algorithms

In order to perform the neural network training, four evolutionary algorithms have been tested. More specifically the following algorithms have been considered: 1) the evolutionary algorithm proposed in [5] for a neural network training, indicated here with Checkers Algorithm (CA), 2) a standard Differential Evolution (DE) [16], [17] with the so called DE/rand/1 strategy, 3) a Particle Swarm Optimization (PSO) according to the implementation proposed in [18] for a similar neural network training, and 4) an Evolutionary Strategy, see [19] and [20], employing the so called "plus strategy" (ES+), i.e., the offspring solutions are merged with the parent solution and the survivor selection scheme selects a predetermined amount of candidate solutions having the best performance and uncorrelated mutation with n step sizes, i.e., a step size is sampled for each design variable.

In order to evaluate the performance of each candidate solution, 100 games against 100 random players are performed and the fitness value is computed as shown in eq. (1). For those evolutionary algorithms which allow an individual to survive over several generations (e.g., DE or ES+), the fitness value is updated, taking into account run off of the previous matches.

At the end of each generation, the range of variability of the fitness values is calculated for each population:

$$\gamma = F_{worst} - F_{best} \tag{2}$$

where F_{worst} and F_{best} are the worst and best values of the population, respectively. The index γ can be seen as a fitness diversity index, see e.g. [21], [22], [23], and [24], which is designed for this specific co-domain (the fitness values vary between 0 and 1). The index γ varies between 0 and 1; if $\gamma \approx 1$ the population is likely to be highly diverse since performance of the population is spread out over the whole range of variability of the co-domain; on the contrary, if $\gamma \approx 0$ performance for the candidate neural networks in the population is very similar and the diversity is therefore low. In order to enhance exploration of the algorithmic system and prevent undesired conditions of stagnation and premature convergence, a migration mechanism has been introduced by employing information of the fitness diversity. More specifically, if in a population $\gamma < 0.05$, 10 individuals are pseudo-randomly swapped between this population and the population with the lowest diversity value. This system allows a harmonic development of the parallel algorithm and, most importantly, performs a refreshment of genotypes in the various populations. The groups of swapped individuals are likely to have similar performance, but by means of a different strategy (genotypically different) since they evolved within different populations. The presence

of new genotypes in the population will then be beneficial, since it will allow a recombination between individuals which are strong and (probably) genotypically distant, see for analogy [25] and [26].

5 Numerical Results

The CA is based on [5], with $\sigma = 0.05$. However, some changes have been made in the parent selection scheme in order to adjust the CA to the problem at hand. A linear ranking [27] with a selective pressure $sp = 1.7$ has been included. The DE, as described in [16], has been implemented with $C_r = 0.3$ and $F = 0.7$. The Particle Swarm Optimization (PSO), as described in [18], has been implemented with parameters $\phi_1 = 2$ and $\phi_2 = 2$. ES+ was implemented, as described in [20], with $\lambda = \mu$ and initial values of mutation step size $\sigma_i = 0.1$. Each algorithm has been run with a population size of 30 individuals; each was run for 35000 fitness evaluations. Each algorithm has been run twice with 10 parallel populations each (i.e., a statistic set of 20 evolving populations). The same algorithms have also been run without the adaptive migration mechanism: each algorithm has been run for 20 independent runs. In order to perform a fair comparison, the same initial populations used for the fitness diversity parallel evolutionary algorithms have also been employed for this experiment on isolated populations.

Fig. 4 shows the performance trend (averaged over the 20 populations) of the proposed fitness diversity parallel evolutionary algorithms. In order to enhance clarity in the result representation, the results have been averaged within intervals of 600 fitness evaluations. Fig. 5 shows the average performance trend (over 20 independent runs) of the four algorithms under study in the isolated implementation. Table 1 lists the average final fitness values with the corresponding standard deviation values.

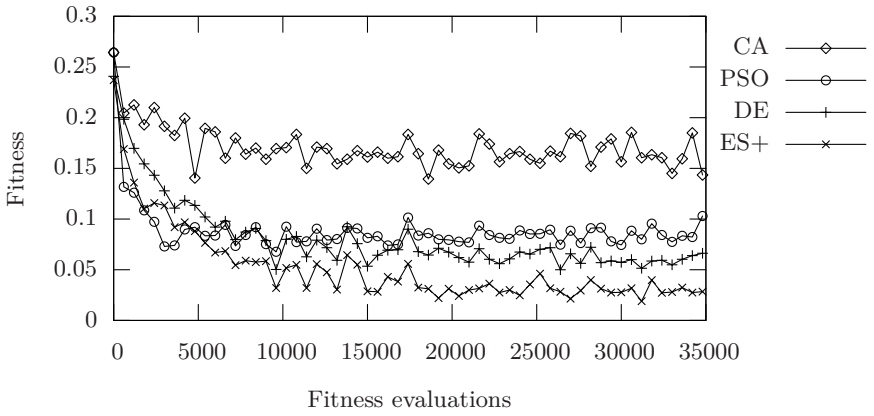


Fig. 4. Performance trend for the four algorithms with migration

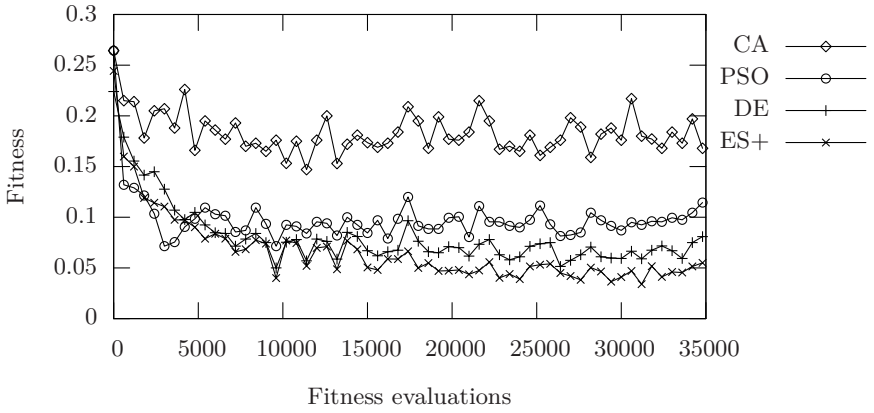


Fig. 5. Performance trend for the four algorithms without migration

As displayed in Table 1, ES+ obtained the best performance solution algorithm, while CA succeeded in performing only a marginal improvement. DE seems to be the second best and PSO comes third. The migration process seems to be beneficial for all algorithms in a similar way. As a matter of fact, it can be observed that ranking among the algorithms remains the same whether migrations are allowed or not.

Table 2 shows the performance of the best players trained according to the four algorithms with and without migrations. Each of the eight players competes in 10,000 games against all of the other seven players as well as against itself, and the score of the first player is given as a percentage of victories. One can see that the results of this cross-comparison is consistent with the results observed in Table 1, when the neural network -based players were confronted with random players.

Finally, the best player trained with ES+ and migration has been playing ten games against five expert, human players. The results are shown in Table 3. The artificial player won on average 4.4 games, with a standard deviation of 1.02, proving to be a fairly challenging player.

6 Conclusion

An artificial, neural network-based player for the *Turtle Race* game has been presented. The neural network has been trained by means of a novel parallel evolutionary algorithm which employs a fitness diversity measure to activate a migration of the individuals between populations in order to perform a refreshment of the genotypes and prevent premature convergence and stagnation. Four popular evolutionary algorithms have been tested within the proposed adaptive parallel framework. Numerical results show that an Evolution Strategy employing a "plus" strategy (despite the presence of noise) in the survivor selection

Table 1. Average fitness and standard deviation for all four algorithms, with and without migrations

Algorithm	*Fitness*
Migration	
CA	0.1435 ± 0.0414
PSO	0.1030 ± 0.0282
DE	0.0663 ± 0.0123
ES+	0.0280 ± 0.0066
Isolated	
CA	0.1680 ± 0.0407
PSO	0.1145 ± 0.0213
DE	0.0809 ± 0.0163
ES+	0.0550 ± 0.0084

Table 2. Cross-comparison of the best individuals produced by each algorithm. The values are expressed in percentage of victories of the first player over 10,000 games.

		Second Player							
		Migration				Isolated			
		CA	PSO	DE	ES+	CA	PSO	DE	ES+
First Player Migration	CA	50.2	45.8	42.7	32.1	53.3	53.3	44.1	33.7
	PSO	51.4	45.1	46.7	33.4	55.6	55.7	46.2	35.6
	DE	59.3	54.8	55.5	41.3	64.2	62.6	53.8	42.5
	ES+	67.8	64.3	66.1	51.8	72.1	75.0	66.1	51.0
Isolated	CA	46.7	41.1	37.7	28.7	49.9	54.4	40.7	31.7
	PSO	45.1	37.3	40.4	25.0	44.8	47.3	37.5	28.1
	DE	57.4	50.5	51.7	37.4	60.1	61.0	52.0	38.8
	ES+	66.4	61.4	62.0	49.1	68.8	70.7	63.8	49.8

Table 3. Number of victories of the artificial player trained with ES+ and migrations against five human players over ten games

Opponent	Player 1	Player 2	Player 3	Player 4	Player 5
Victories	3	4	6	5	4

scheme is fairly promising for this class of problems. In addition, the proposed fitness diversity migration seems to be beneficial to the tested algorithms regardless of the evolutionary structure which characterizes them. A test carried out against expert human players shows that the designed artificial player is rather challenging since it succeeds in defeating human players in almost half of the games.

References

1. Lucas, S., Kendall, G.: Evolutionary computation and games. IEEE Computational Intelligence Magazine **1** (2006) 10–18
2. Fogel, D.: Using evolutionary programming to create networks that are capable of playing tic-tac-toe. In: Proceedings of IEEE International Conference on Neural Networks. (1993) 875–880
3. Richards, N., Moriarty, D., Miikkulainen, R.: Evolving neural networks to play go. Applied Intelligence **8** (1998) 85–96
4. Runarsson, T.P., Lucas, S.M.: Coevolution versus self-play temporal difference learning for acquiring position evaluation in small-board go. IEEE Transactions on Evolutionary Computation **9** (2005) 628–640
5. Chellapilla, K., Fogel, D.: Evolving an expert checkers playing program without using human expertise. IEEE Transactions on Evolutionary Computation **5** (2001) 422–428
6. Moraglio, A., Togelius, J.: Geometric particle swarm optimization for the sudoku puzzle. In: GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation. (2007) 118–125

7. Fudenberg, D., Tirole, J.: 6. In: Game Theory. MIT Press (1993)
8. Barricelli, N.A.: Esempi numerici di processi di evoluzione. Methodos (1954) 45–68
9. Barone, L., While, L.: Evolving adaptive play for simplified poker. In: Proceedings of IEEE Intl. Conf. on Computational Intelligence (ICEC-98). (1998) 108–113
10. Jaśkowski, W., Krawiec, K., Wieloch, B.: Evolving strategy for a probabilistic game of imperfect information using genetic programming. Journal Genetic Programming and Evolvable Machines **9** (2008) 281–294
11. Engelbrecht, A.P.: Computational Intelligence: An Introduction. John Wiley & Sons Ltd. (2002)
12. Bourg, D.M., Seemann, G.: AI for Game Developers. O'Reilly (2004)
13. Chellapilla, K., Fogel, D.: Evolution, neural networks, games, and intelligence. Proceedings of the IEEE **87** (1999) 1471–1496
14. Abramson, B.: Expected-outcome: a general model of static evaluation. IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1990) 182–193
15. Brügmann, B.: Monte carlo go. IEEE Transactions on Pattern Analysis and Machine Intelligence **12** (1993) 182–193
16. Storn, R., Price, K.: Differential evolution—a simple and efficient heuristic for global optimization over continuous spaces. Journal on Global Optimization **11** (1997) 341–359
17. Price, K.V., Storn, R.M., Lampinen, J.A.: Differential Evolution–A Practical Approach to Global Optimization. Springer (2005)
18. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: IEEE International Conference on Neural Networks. Volume 4. (1995) 1942–1948
19. Rechemberg, I.: Evolutionstrategie: Optimierung Technisher Systeme nach prinzipien des Biologishen Evolution. Fromman-Hozlboog Verlag (1973)
20. Eiben, A.E., Smith, J.E. In: Introduction to Evolutionary Computation. Springerverlag, Berlin (2003) 71–87
21. Caponio, A., Cascella, G.L., Neri, F., Salvatore, N., Sumner, M.: A fast adaptive memetic algorithm for on-line and off-line control design of pmsm drives. IEEE Transactions on System Man and Cybernetics-part B **37** (2007) 28–41
22. Neri, F., Toivanen, J., Cascella, G.L., Ong, Y.S.: An adaptive multimeme algorithm for designing HIV multidrug therapies. IEEE/ACM Transactions on Computational Biology and Bioinformatics **4** (2007) 264–278
23. Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., Rossi, T.: An enhanced memetic differential evolution in filter design for defect detection in paper production. Evolutionary Computation **16** (2008) to appear.
24. Caponio, A., Neri, F., Tirronen, V.: Super-fit control adaptation in memetic differential evolution frameworks. Soft Computing-A Fusion of Foundations, Methodologies and Applications (2008) to appear.
25. Smith, J.E.: Coevolving memetic algorithms: A review and progress report. IEEE Transactions on Systems, Man, and Cybernetics, Part B **37** (2007) 6–17
26. Zamuda, A., Brest, J., Bošković, B., Žumer, V.: 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) 3719–3726
27. Back, T.: Selective pressure in evolutionary algorithms: a characterization of selection mechanisms. In: Proceedings of the IEEE World Congress on Computational Intelligence. Volume 1. (1994) 57–62

PII

DISTRIBUTED DIFFERENTIAL EVOLUTION WITH EXPLORATIVE-EXPLOITATIVE POPULATION FAMILIES

by

Matthieu Weber, Ferrante Neri and Ville Tirronen 2009

In Genetic Programming and Evolvable Machines, volume 10, issue 4, pages 343–371

Genetic Programming and Evolvable Machines manuscript No.
(will be inserted by the editor)

Distributed Differential Evolution with Explorative-Exploitative Population Families

Matthieu Weber · Ferrante Neri · Ville Tirronen

Received: date / Accepted: date

Abstract This paper proposes a novel distributed differential evolution, namely Distributed Differential Evolution with Explorative-Exploitative Population Families (DDE-EEPF). In DDE-EEPF the sub-populations are grouped into two families. Sub-populations belonging to the first family have constant population size, are arranged according to a ring topology and employ a migration mechanism acting on the individuals with the best performance. This first family of sub-populations has the role of exploring the decision space and constituting an external evolutionary framework. The second family is composed of sub-populations with a dynamic population size: the size is progressively reduced. The sub-populations belonging to the second family are highly exploitative and are supposed to quickly detect solutions with a high performance. The solutions generated by the second family then migrate to the first family. In order to verify its viability and effectiveness, the DDE-EEPF has been run on a set of various test problems and compared to four distributed differential evolution algorithms. Numerical results show that the proposed algorithm is efficient for most of the analyzed problems, and outperforms, on average, all the other algorithms considered in this study.

This research is supported by the Academy of Finland, Akatemiatutkija 00853, Algorithmic Design Issues in Memetic Computing.

Matthieu Weber
Tel.: +358-14-2603056
E-mail: matthieu.weber@jyu.fi

Ferrante Neri
Tel.: +358-14-2602764
E-mail: ferrante.neri@jyu.fi

Ville Tirronen
Tel.: +358-14-2604987
E-mail: ville.tirronen@jyu.fi

University of Jyväskylä
Department of Mathematical Information Technology
P.O. Box 35 (Agora), 40014, Jyväskylä
Finland
Fax: +358-14-2604981

Keywords Differential Evolution · Distributed Systems · Population Size Reduction · Multi-family Distributed Algorithms

1 Introduction

Panmictic Evolutionary Algorithms [3], i.e., standard evolutionary algorithms characterized by a unique population and global recombination among all the possible individuals, are commonly used tools in optimization which have shown a high performance on various problems in applied science and engineering. On the other hand, these algorithms suffer from the well-known problems of stagnation and premature convergence caused by an improper balance of the population diversity, see e.g. [16]. In other words, the main drawback of a standard Evolutionary Algorithm (EA) is the fact that it may fail to generate new promising solutions and return a poorly performing suboptimal solutions.

In order to overcome this drawback, computer scientists and practitioners, since the earliest EA implementations, have attempted to enhance the EA performance by modifying the original ideas in various manners, e.g. proposing alternative search structures [52], developing adaptation models [45], hybridizing EAs with local search algorithms [33], or designing structured versions of EAs. The latter category, which is the focus of this paper, consists of a decentralization of the population into a set of sub-populations which have diverse roles and can somehow interact. Two most famous examples of structured EAs are Cellular Evolutionary Algorithms (CEAs), see [58], [5], [25], [24], [1] and [26], and Distributed Evolutionary Algorithms (DEAs), see [56], [7], [2], [22], and [62] and classification in [4] and [3]. In CEAs, the sub-populations are constructed on the basis of a neighborhood criterion and thus each sub-population has the role of exploring (and exploiting) a different region of the decision space. In DEAs, all the sub-populations explore the entire decision space and develop a parallel evolution for the solution of the same problem, usually cooperating in the search of the optimum by means of some information exchange.

Due to their nature, EAs (as well as other population based metaheuristics) are easy to be run in parallel over multiple machines or multiple core machines. The oldest, simplest and most straightforward EA parallelization is the so called Single-population Master-Slave Parallel Model (see [27] and [43]). In this kind of parallelization, the EA runs with a single population and the fitness evaluations are distributed over several cores. Clearly, this parallelization does not influence the EA structure (since it is algorithmically equivalent to a standard EA) nor its performance.

Since structured EAs modify the normal EA by distributing the whole population into a set of many sub-populations, a parallelization can be naturally performed by assigning the management of a sub-population to each core. In this case, the distribution over several computational cores and the algorithmic modifications/enhancements are strictly connected issues. In literature several classifications of parallel EAs have been proposed, see e.g., [10], [11], and [13].

In terms of parallelization, a popular classification, see [38], distinguishes between coarse-grained and fine-grained algorithms. Coarse-grained algorithms run on a limited amount of populations with a relatively high number of individuals while fine-grained algorithms run on many populations composed of only a few individuals. This classification often coincides with the one mentioned above (DEA and

CEA) which sees the same issue from an alternative (according to the algorithmic structure) viewpoint.

Although the sub-populations evolve separately, they interact somehow and cooperate by exchanging information in order to pursue their common global optimization goal. In other words, the sub-populations copy, transfer, and exchange individuals according to various migration schemes, see e.g., [34], [8], [11], [12], and [14].

Over the recent years, among the other EAs present in literature, Differential Evolution (DE see [51], [49], [42], and [15]) stimulated the interest of computer scientists and practitioners. As many applications in engineering have proven, DE is a reliable and versatile function optimizer which is especially efficient in continuous problems. Thanks to, on one hand, its simplicity and ease of implementation, and on the other hand, its reliability and high performance, DE became a very popular solution for solving various real-world problems almost immediately after its original definition.

Although DE has a great potential, it has been clear to the scientific community that some modifications to the original structure were necessary in order to significantly improve its performance. A popular way, especially during the latest years, to enhance the DE is the integration of structured populations evolving in parallel. For example, in [30] a distributed DE scheme employing a ring topology (the cores are interconnected in a circle and the migrations occur following the ring) has been proposed for the training of a neural network. In [48], an example of DE parallelization is given for a medical imaging application. A few famous examples of distributed DE are presented in [59], [60], and [63]; in these papers the migration mechanism as well as the algorithmic parameters are adaptively coordinated on a criterion based on the genotypical diversity. In paper [61], a distributed DE for preserving the diversity in the niches is proposed in order to solve multi-modal optimization problems. In [53], a distributed DE characterized by a ring topology and the migration of the individuals with the best performance, to replace random individuals of the neighbor sub-population, has been proposed. An application of the algorithm in [53] for the training of a neural network has been presented in [39]. Following a similar logic, paper [29] proposes a distributed DE where the computational cores are arranged according to a ring topology and, during the migration, the best individual of a sub-population replaces the oldest member of the neighbor population. In [17], [18], and [19] a distributed DE has been designed for the image registration problem. In these papers, a computational core acts as a master by collecting the best individuals detected by the various sub-populations running in slave cores. The slave cores are connected in a grid and a migration is arranged among neighbor sub-populations. In [6], a distributed DE which modifies the scheme proposed in [53] has been presented. In [6], the migration is based on a probabilistic criterion depending on five parameters. It is worthwhile to mention that also some parallel implementations of panmictic DE are available in literature, see [36]. An investigation about DE parallelization is given in [31].

This paper deals with distributed versions of DE and proposes a novel implementation of distributed DE, namely Distributed Differential Evolution with Explorative-Exploitative Population Families (DDE-EEPF). The DDE-EEPF is a distributed algorithm composed of two families of sub-populations. In the first family the sub-populations have a fixed population size and employ a migration scheme. In the second family, the sub-populations have a different behavior depending on the generation number. During the beginning of the evolution, the sub-populations evolve

independently by applying a population size reduction scheme. During the late stages of the evolution, the sub-populations belonging to the second family allow a migration of the individuals with the best performance to the sub-populations belonging to the first family. The distributed mechanism counts then on a family of sub-populations for exploring the decision space and performing the global search and on a family of sub-populations for exploiting the available search directions and thus detecting high quality solutions. The second family is then supposed to assist the first one by "injecting" high performance solutions within its explorative frameworks in the middle of their optimization process. This operation is supposed to enhance the overall algorithmic performance.

The remaining part of the paper is organized in the following way. Section 2 describes the working principles of DE and explains the reasons behind the parallelization. Section 3 gives a short description of recently presented distributed versions of DE and introduces the algorithms employed for comparison in the experimental section. Section 4 describes the proposed algorithm. Section 5 shows the experimental setup and numerical results of the present study. Section 6 gives the conclusions of this paper.

2 Standard Differential Evolution

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

According to its original definition given in [51], the DE consists of the following steps. An initial sampling of S_{pop} individuals is performed pseudo-randomly with a uniform distribution function within the decision space D. At each generation, for each individual x_i of the S_{pop}, three individuals x_r, x_s and x_t are pseudo-randomly extracted from the population. According to the DE logic, a provisional offspring x'_{off} is generated by mutation as:

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

where $F \in [0, 1+[$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated. With $F \in [0, 1+[$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1, see [42]. While there is no theoretical upper limit for F, effective values are rarely greater than 1.0. The mutation scheme shown in Equation (1) is also known as DE/rand/1. Other variants of the mutation rule have been subsequently proposed in literature, see [44]:

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

where x_{best} is the solution with the best performance among the individuals of the population, x_u and x_v are two additional pseudo-randomly selected individuals. It is worthwhile to mention the rotation invariant mutation shown in [41]:

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

where K is is the combination coefficient, which, as suggested in [41], should be chosen with a uniform random distribution from $[0, 1]$ and $F' = K \cdot F$. For this special mutation the mutated solution does not undergo the crossover operation described below.

Recently, in [42], a new mutation strategy has been defined. This strategy, namely DE/rand/1/either-or, consists of the following:

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

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

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

$$x_{off,j} = \begin{cases} x_{i,j} & \text{if} \quad rand\,(0,1) < CR \\ x'_{off,j} & otherwise \end{cases} \tag{3}$$

where $rand\,(0, 1)$ is a random number between 0 and 1; j is the index of the gene under examination.

The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) \leq f(x_i)$; otherwise no replacement occurs. It must be remarked that although the replacement indexes are saved, one by one, during the generation, the actual replacements occur all at once at the end of the generation. For the sake of clarity, the pseudo-code highlighting the working principles of the DE is shown in Figure 1.

2.1 Why to Distribute Differential Evolution?

As shown in Subsection 2, DE is based on a very simple idea, i.e., the search by means of adding vectors and a one-to-one spawning for the survivor selection. Thus, DE is very simple to implement/code and contains a limited number of parameters to tune (only S_{pop}, F, and CR). In addition, the fact that DE is rather robust and versatile has encouraged engineers and practitioners to employ it in various applications. For example, in [28], a DE application to the multisensor fusion problem is given. In [50], a filter design is carried out by DE. In [54] and [55], a DE based algorithm is implemented to design a digital filter for paper industry applications. A review of DE applications is presented in [40].

From an algorithmic viewpoint, the reasons of the success of DE have been highlighted in [21]: the success of the DE is due to an implicit self-adaptation contained within the algorithmic structure. More specifically, since, for each candidate solution, the search rule depends on other solutions belonging to the population (e.g. x_t, x_r, and x_s), the capability of detecting new promising offspring solutions depends on the current distribution of the solutions within the decision space. During the early stages of the optimization process, the solutions tend to be spread out within the decision space. For a given scale factor value, this implies that the mutation appears to generate new solutions by exploring the space by means of a large step size (if x_r and

```
generate S_pop individuals of the initial population pseudo-randomly
while budget condition do
  for i = 1 : S_pop do
    compute f(x_i)
  end for
  for i = 1 : S_pop do
    {** Mutation **}
    select three individuals x_r, x_s, and x_t
    compute x'_off = x_t + F(x_r − x_s)
    {** Crossover **}
    x_off = x'_off
    for j = 1 : n do
      generate rand(0, 1)
      if rand(0, 1) < CR then
        x_off,j = x_i,j
      end if
    end for
    {** Selection **}
    if f(x_off) ≤ f(x_i) then
      save index for replacement x_i = x_off
    end if
  end for
  perform replacements
end while
```

Fig. 1 DE pseudocode

x_s are distant solutions, $F\left(x_r - x_s\right)$ is a vector characterized by a large modulus). During the optimization process, the solutions of the population tend to concentrate on specific parts of the decision space. Therefore, the step size in the mutation is progressively reduced and the search is performed in the neighborhood of the solutions. In other words, due to its structure, a DE scheme is highly explorative at the beginning of the evolution and subsequently becomes more exploitative during the optimization.

Although this mechanism seems at first glance very efficient, it hides a limitation. If for some reasons, the algorithm does not succeed in generating offspring solutions which outperform the corresponding parent, the search is repeated again with similar step size values and likely fails by falling into the undesired stagnation condition (see [32]). Stagnation is the undesired effect which occurs when a population-based algorithm does not converge to a solution (even suboptimal) and the population diversity is still high. In the case of the DE, stagnation occurs when the algorithm does not manage to improve upon any solution of its population for a prolonged number of generations. In other words, the main drawback of the DE is that the scheme has, for each stage of the optimization process, a limited amount of exploratory moves and if these moves are not enough for generating new promising solutions, the search can be heavily compromised.

Thus, in order to enhance the DE performance, alternative search moves should support the original scheme and promote a successful continuation of the optimization process. The use of multiple populations in distributed DE algorithms allows an observation of the decision space from various perspectives and, most importantly, decreases the risk of stagnation since each sub-population imposes a high exploitation pressure. In addition, the migration mechanism ensures that solutions with a high

performance are included within the sub-populations during their evolution. This fact is equivalent to modifying the set of search moves. If the migration privileges the best solutions, the new search moves promote the detection of new promising search directions and thus allow the DE search structure to be periodically "refurbished". Thus, the migration is supposed to mitigate the risk of stagnation of the DE (sub-)populations and to enhance the global algorithmic performance.

In addition, within a DE framework there is, with respect to the other EAs, a different relationship between algorithmic functioning and the population diversity. As it is well known, the concept of population diversity is very important in many EAs and, in order to obtain a proper algorithmic behavior, it is crucial to design a system to maintain the diversity high throughout the evolution. It might likely happen that a diversity loss in an EA can cause the premature convergence towards solutions characterized by a poor performance. In DE, as for swarm intelligence optimizers, there is not a quick diversity loss and the algorithm can perform the entire optimization process and still keep the diversity high. This fact can cause an excessively exploratory behavior and then the stagnation phenomenon mentioned above. Thus a successful DE, as explained above, is supposed to explore the entire decision space only during the early stage of the evolution and subsequently narrows its search in a small (and interesting) portion of the domain. In summary, DE is an atypical EA: operations which aim at maintaining the diversity high are beneficial to most of the EAs while being detrimental to DE.

On the other hand, it must be remarked that many population-based metaheuristics and not only the DE can greatly benefit from a proper parallelization. Generally speaking, a distributed population structure can offer an alternative search view to the algorithmic structures and generate compensation to the weak points of the chosen algorithm. In DE stagnation problems and the limited set of available moves have been highlighted. A classical example of the advantages of distribution can be the Distributed Evolution Strategies, see [46]. Evolution Strategies, especially those employing the "plus" strategy, suffer from premature convergence and diversity loss. The use of a distributed population with a proper migration mechanism can be an efficient countermeasure against diversity loss and thus an enhancement in the algorithmic performance.

In other words, the distribution of the population can be beneficial to DE (as well as to swarm intelligence algorithms) since it can generate extra moves in the search logic, thus mitigating the stagnation effect, and to other EAs since it can produce genotypes which increase the diversity, thus mitigating the premature convergence inconveniences.

The implementation of the parallelization e.g., occurrence of the migrations as well as the choice of migrating individuals, replacement rules, and choice of population involved in the migration events have been intensively discussed in literature and, in many cases, are still points of investigation for computer scientists. As a matter of fact, although the employment of structured populations can be beneficial to many meta-heuristics, each case must be, in our view, separately analyzed and the design of the distributed structure must be performed taking into account the nature and the limitation of each core algorithm.

The next section illustrates three successful, recently developed distributed DEs and highlights their differences and similarities.

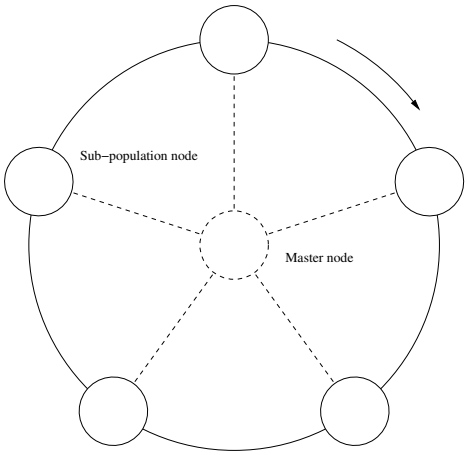


Fig. 2 Unidirectional ring topology in the Parallel Differential Evolution algorithm

3 Distributed Differential Evolution: Recently Developed Algorithms

This section describes three distributed algorithm based on a DE structure recently proposed in literature. The algorithms described in this section are, according to our judgement, representative of the state of the art of structured DE algorithms and have been included in the benchmark in order to compare the performance of the proposed approach. Although the notation can generate some confusion, i.e., all the algorithms are distributed and can easily be parallelized, we decided to indicate them according to the original terminology defined by their respective authors.

3.1 Parallel Differential Evolution

In [53], the problem of parallelization for DE schemes has been studied through an experimental analysis and an algorithm, namely Parallel Differential Evolution (here indicated with PDE) has been proposed.

The original PDE implementation uses the Parallel Virtual Machine (PVM), allowing multiple computers (called *nodes*) to be organized as a cluster and exchange arbitrary messages. PDE is organized around one master node and N sub-populations running each on one node, and organized as a unidirectional ring, as illustrated in Figure 2. It must be noted that although the logical topology is a ring which does not contain the master node, the actual topology is a star, where all communications (i.e., the migrations of individuals) are passing through the master.

Each sub-population runs a regular DE algorithm while the master node coordinates the migration of individuals between the sub-populations. On each generation, the sub-population has a given probability to send a copy of its best individual to its next neighbor in the ring. Figure 3 describes the behaviors of both the master node and the sub-populations in more details.

The DE variant run by each sub-population is the same across all the sub-populations. In [53], six mutation strategies have been compared, namely DE/best/1, DE/rand/1, DE/cur-to-best/1, DE/best/2, DE/rand/2 described in Section 2, as

```
spawn N sub-populations, each one on a different processor
for each generation do
   receive an individual from each sub-population
   for each received individual do
      if rand(0, 1) < φ then
         send the individual to the next subpopulation in the ring
      end if
   end for
   if the stop criterion for the objective function is met then
      send a termination signal to all the sub-populations
   end if
end for
```

(a) At the master node

```
for each generation do
   perform a DE step
   send a copy of the best individual to the master node
   if a migrated individual has been received then
      replace a random individual, different from the best, by this migrated individual
   end if
   if a termination signal has been received then
      terminate the execution
   end if
end for
```

(b) At each sub-population

Fig. 3 Pseudo-code of the PDE algorithm

well as the trigonometric operator described in [20]. Each strategy is used with different values of the migration constant ϕ and compared over seven test functions whose dimensions vary between 2 and 30. The results show that DE/best/1 is the most efficient mutation strategy and quite stable across different values of ϕ, whereas the results of DE/rand/1 are average and quite unstable when ϕ varies.

3.2 Island Based Distributed Differential Evolution

In [6] a distributed DE, namely Island Based Distributed Differential Evolution (IBDDE) has been proposed. The IBDDE is a modified version of the PDE described in Subsection 3.1. The algorithm is described in a generic way, presenting a population $\mathcal{P}$ structured in m sub-populations P_p of n_p individuals. The size of $\mathcal{P}$ is noted $N = \sum_{i=1}^{n} n_i$. The migration policy is then defined as a five-tuple $\mathcal{M} = (\gamma, \rho, \phi_s, \phi_r, \tau)$. $\gamma \in \mathbb{N}$ is the number of generations between two migrations, $\rho \in \mathbb{N}$ is the number of individuals which are migrated from a sub-population during each migration, ϕ_s is the selection function which, applied to a sub-population, returns the migrating individuals, ϕ_r is the replacement function that selects the individuals to be replaced by the immigrants in the receiving sub-population, and $\tau : \mathcal{P} \rightarrow 2^{\mathcal{P}}$ is the topological model, which selects what sub-population can send to (or receive from) what other sub-population. Figure 4 describes the algorithm as pseudo-code.

For the actual experiments, the population size N is set to 20, and the population is divided into two sub-populations of 10 individuals in one experiment, and

```
initialize(P_p)
while the stopping condition is not met do
  perform a DE step
  if the last migration was γ generations ago then
    for each of the ρ individuals to send do
      v'^i_g ← φ_s(P_p)
      send v'^i_g to P_j chosen by τ
    end for
  end if{** Asynchronous communication **}
  while individuals are arriving do
    receive v'^i_g from P_j
    replace an individual chosen from φ_s(P_p) by v'^i_g
  end while
end while
```

Fig. 4 Pseudo-code of the IBDDE algorithm for sub-population P_p

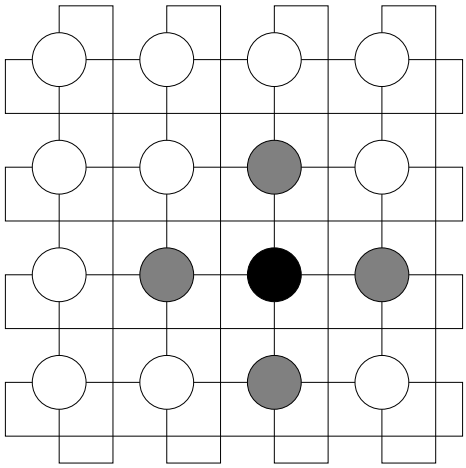


Fig. 5 Torus topology in the Distributed Differential Evolution algorithm

into four sub-populations of 5 individuals in a second experiment. The migration parameters are set to $\gamma = 100$, $\rho = 1$, the functions ϕ_s and ϕ_r are defined to randomly select an individual, and the topology τ is a unidirectional ring, very similar to the logical topology used by PDE (see Subsection 3.1). The mutation strategy for DE is DE/rand/1, and the algorithm is tested on 25 different test functions in 30 and 50 dimensions, for a total of 50 test functions.

3.3 Distributed Differential Evolution

In [17], [18], and [19], in order to solve and image registration problem a distributed DE (here indicated with DDE) has been proposed. This algorithm differs from PDE and IBDDE by the topology it uses. Instead of a unidirectional ring, DDE uses a locally connected topology, where each node is connected to μ other nodes. Figure 5 represents such a topology where the nodes are arranged in a mesh folded into a torus.

```
initialize the sub-population
while the stopping condition is not met do
  perform a DE step
  if the last migration was M_I generations ago then
    send a copy of the best individual to each neighbor
  end if
  if there are incoming individuals then
    replace the worst S_I × μ individuals by the S_I × μ incoming ones
  end if
end while
```

Fig. 6 Pseudo-code of the DDE algorithm at a sub-population

In this configuration $\mu = 4$, i.e., each node (such as the black disc in the figure) has exactly four nearest neighbors (represented by the four grey discs). In DDE, each node represents one processor running a DE algorithm with a DE/rand/1 mutation strategy on a sub-population. Every M_I generations (the migration interval), each sub-population is allowed to exchange S_I (the migration rate) individuals with its nearest neighbors. In the experimental setup, each node sends a copy of its best individual to its neighbors. Figure 6 describes the algorithm as pseudo-code.

DDE also makes use of a master node, whose role is to collect the best solutions found in each sub-population and to present the results to the user.

4 Distributed Differential Evolution with Explorative-Exploitative Population Families

The proposed algorithm, namely Distributed Differential Evolution with Explorative-Exploitative Population Families (DDE-EEPF), consists of the following steps.

An initial population of S_{pop} individuals is pseudo-randomly sampled within the decision space D. These S_{pop} individuals are distributed over m sub-populations; each sub-population has a size equal to $S_{s\text{-}pop} = \frac{S_{pop}}{m}$. The m sub-populations are then divided into two families: α are assigned to the first family (hereafter named P_p with $p = 1, 2, \ldots, \alpha$) and β to the second (named Q_q with $q = 1, 2, \ldots, \beta$); $\alpha + \beta = m$.

The α sub-populations belonging to the first family are arranged according to a ring topology following the suggestions given in [53]. In the first family, each sub-population P_p evolves like a standard DE and employs the DE/rand/1 mutation strategy illustrated in Equation (1) and the crossover described in Equation (3). The replacements of the individuals are performed according to the one-to-one spawning shown in Section 2. These α sub-populations interact by means of a migration scheme. For each population P_p, $p = 1, \ldots, \alpha$, with a probability p_{mig}, a copy of the best individual $x_{best}^{P_p}$ of sub-population P_p is sent to the next sub-population in the ring. At that sub-population, the incoming $x_{best}^{P_p}$ replaces a pseudo-randomly selected (uniform distribution) solution, which is then discarded.

The behavior of the β sub-populations Q_q composing the second family is divided into two ages. During the first age, the sub-populations evolve independently without the support of a migration scheme. Each sub-population employs the population size reduction strategy introduced in [9] (see also [23] and [57]). This strategy requires that initial population size $S_{s\text{-}pop}^1$ (in our case $S_{s\text{-}pop}^1 = S_{s\text{-}pop}$), duration of the first age in terms of fitness evaluations (time budget T_b), and number of stages N_s (the

number of population sizes employed during the algorithm's run), are prearranged. Then, the first age (T_b) is divided into N_s periods, each period being characterized by a population size value $S^k_{s\text{-}pop}$ (for $k = 1$ we obtain the initial population size). Each period is composed of N^k_g generations which are calculated in the following way:

$$N^k_g = \left\lfloor \frac{T_b}{N_s S^k_{s\text{-}pop}} \right\rfloor + r_k \tag{4}$$

where r_k is a constant non-negative value which takes a positive value when T_b is not divisible by N_s. In this case r_k extra generations are performed. The population reduction is simply carried out by halving the population size at the beginning of the new stage, see [9]. In other words, for $k = 1, 2, \ldots, N_s$, the population size is halved $S^{k+1}_{s\text{-}pop} = \frac{S^k_{s\text{-}pop}}{2}$. The selection of the survivors occurs by dividing into groups the sub-population according to their indexes and performing the one-to-one spawning to each corresponding pair of individuals. Finally, it should be remarked that in order to guarantee a proper functioning of the population reduction mechanism, populations should never undergo sorting of any kind.

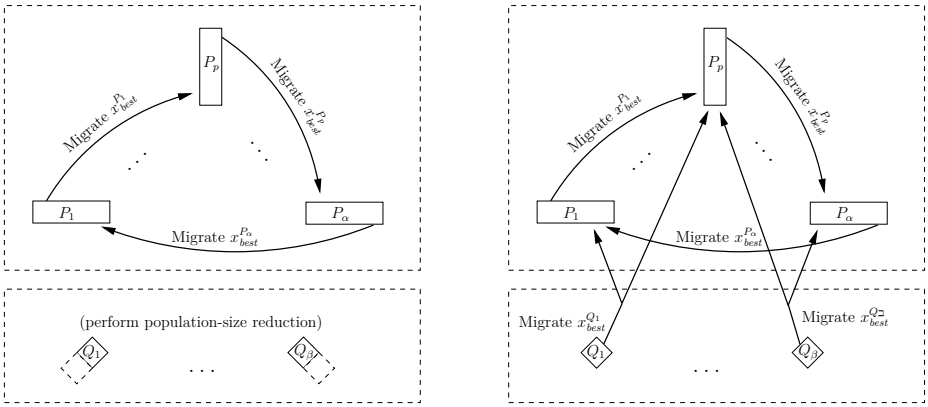
In this way, the population size is progressively reduced during the optimization process until the budget of the first age (T_b) is reached. The concept behind this strategy is that of focusing the search in progressively smaller search spaces in order to inhibit the DE stagnation. During the early stages of the optimization process, the search requires a highly explorative search rule, i.e., a large population size, in order to explore a large portion of the decision space. During the optimization, the search space is progressively narrowed by decreasing the population size and thus exploiting the promising search directions previously detected. Although this strategy does not guarantee the detection of the global optimum, it allows quick improvements in the solution performance.

During the second age, the sub-populations composing the second family keep their size constant at their minimum and a migration scheme is employed. At each generation, for each sub-population belonging to the second family, its individual with the best performance migrates to a sub-population of the first family. As for the migration among sub-populations of the first family, individuals are copied and replace a pseudo-randomly selected individual of the target population. The choice of the target population is performed at pseudo-random with a uniform probability.

It is important to remark that a synchronization of the computational overhead has been implemented among the sub-populations belonging to the first and second family. In other words, the amount of fitness evaluations during a generation is kept constant for all the sub-populations. For example, if $k = 2$, the size of a sub-population belonging to the first family is twice bigger than the size of a sub-population of the second family. Under these conditions, for each generation in the first family two consecutive generations are performed in the second family.

Figure 7 gives a graphical representation of the DDE-EEPF during both its ages of the evolution.

According to the idea proposed in this paper, the whole DE population is divided into sub-populations. These sub-populations are grouped into two families. Each family plays a different role on the optimization process. The sub-populations of the first family are supposed to explore the decision space and to attempt detecting the global optimum. The risk of stagnation is mitigated by the migration mechanism among the sub-populations of the first family. On the contrary, the sub-populations



(a) Operation during the first age (b) Operation during the second age

Fig. 7 Operation scheme of the DDE-EEPF

belonging to the second family have a completely different role and behavior. The sub-populations of the second family do not aim at exploring the entire decision space, but their role is focused on the greedy achievement of solutions with a high performance, despite the fact that these solutions can be suboptimal. There is no migration mechanism within the second family in order to allow a full exploitation of the available search directions. Migrations would slow down the search since they result in an increase of the exploration properties. When the sub-populations of the second family detected high quality solutions, the inter-family migration occurs. The introduction of high quality solutions into the exploratory search structure of the first family is supposed to further decrease the risk of stagnation and, most importantly, to assist the global search by proposing the exploration of promising search directions. Hence, during the second age, the sub-populations of the first family have the role to improve upon the immigrants coming from the second family and continue the search towards the optimum. In this sense, the first family is meant to be explorative while the second is to be exploitative.

5 Experimental Results

The test problems listed in Table 1 have been considered in this study.

The rotated version of some of the test problems listed in Table 1 have been included into the benchmark set. These rotated problems have been generated by means of the multiplication of the vector of variables by a randomly generated orthogonal rotation matrix. In total, twenty-four test problems have been considered in this study with both $n = 500$ and $n = 1,000$. Each algorithm has been run for $500,000$ fitness evaluations in the case of $n = 500$ and for $1,000,000$ fitness evaluations when $n = 1,000$. Fifty independent runs have been performed for each algorithm involved in this paper.

The DDE-EEPF has been tested and compared with the PDE, IBDDE, DDE and one more algorithm designed by us for comparison (see below). Preliminary experiments related to the sequential DE have shown that DE is not at all competitive

Table 1 Test Problems

Test Problem	**Function**	**Decision Space**
Ackley	$-20 + e + 20\exp\left(-\frac{0.2}{n}\sqrt{\sum_{i=1}^{n} x_i^2}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i)x_i\right)$	$[-1,1]^n$
Alpine	$\sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1x_i \rvert$	$[-10,10]^n$
Axis-parallel hyper-ellipsoid	$\sum_{i=1}^{n} i x_i^2$	$[-5.12,5.12]^n$
DeJong	$\lVert x \rVert^2$	$[-5.12,5.12]^n$
DropWave	$-\frac{1+\cos\left(12\sqrt{\lVert x \rVert^2}\right)}{\frac{1}{2}\lVert x \rVert^2+2}$	$[-5.12,5.12]^n$
Griewangk	$\frac{\lVert x \rVert^2}{4000} - \prod_{i=0}^{n}\cos\frac{x_i}{\sqrt{i}} + 1$	$[-600,600]^n$
Michalewicz	$-\sum_{i=1}^{n}\sin x_i \left(\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)^{20}$	$[0,\pi]^n$
Pathological	$\sum_{i=1}^{n-1}\left(0.5 + \frac{\sin^2\left(\sqrt{100x_i^2+x_{i+1}^2}-0.5\right)}{1+0.001*\left(x_i^2-2x_ix_{i+1}+x_{i+1}^2\right)^2}\right)$	$[-100,100]^n$
Rastrigin	$10n + \sum_{i=0}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12,5.12]^n$
Rosenbrock valley	$\sum_{i=1}^{n-1}\left(100\left(x_{i+1}-x_i^2\right)^2 + (1-x_i)^2\right)$	$[-2.048,2.048]^n$
Schwefel	$\sum_{i=1}^{n} -x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500,500]^n$
Sum of powers	$\sum_{i=1}^{n} \lvert x_i \rvert^{i+1}$	$[-1,1]^n$
Tirronen	$3\exp\left(-\frac{\lVert x \rVert^2}{10n}\right) - 10\exp\left(-8\lVert x \rVert^2\right)$ $+\frac{2.5}{n}\sum_{i=1}^{n}\cos\left(5\left(x_i + (1+i \bmod 2)cos\left(\lVert x \rVert^2\right)\right)\right)$	$[-10,5]^n$

with the above distributed ones, and has therefore been left out from these result presentation.

The algorithms considered in this study have been run with the following parameter setting.

- The DE used within the sub-populations of each of the distributed algorithms has been run with $F = 0.7$ and $CR = 0.3$, in accordance with the suggestions given in [65] and [64].
- The PDE has been run with populations of 200 or 400 individuals divided into 5 sub-populations of 40 or 80 individuals each, for the 500 and 1,000-dimensional problems, respectively. Despite [53] showing better performance for the DE/best/1 mutation strategy in 30 and 50 dimensions, it has proven ex-

cessively exploitative and has lead to premature convergence of the solutions when used on higher dimension problems. In order to perform a fair comparison, an analysis on mutations strategies has been made, leading to the choice of DE/rand/1 and setting the migration constant to $\phi = 0.2$. These are the settings which have been chosen for the experiments described below.

- Similarly to PDE, the IBDDE has been run with populations of 200 or 400 individuals divided into 5 sub-populations of 40 or 80 individuals each, depending on the dimensionality of the test problems. The other parameters have been chosen according to the values in [6]: the sub-populations exchange one individual ($\rho = 1$) every 100 generations ($\gamma = 100$). ϕ_s and ϕ_r have been defined so as to choose a uniformly random individual, and τ has been set to a unidirectional ring.
- For the 500-dimensional problems, the DDE has been run with a population of 200 individuals divided into 16 sub-populations of alternatively 12 or 13 individuals. In the case of the 1,000-dimensional problems, the population has been set to 400 individuals divided into 16 sub-populations of 25 individuals. Following the suggestions in [18] the sub-populations have been organized into a 4×4 grid folded into a torus ($\mu = 4$). Each sub-population migrated only its best individual ($S_I = 1$) every $M_I = 5$ generations.
- The DDE-EEPF has been run with $S_{pop} = 200$ or $S_{pop} = 400$ (representing 5 populations of 40 or 80 individuals, for the 500 or 1,000-dimensional problems, respectively). The first family has been composed of $\alpha = 3$ sub-populations while the second family has been made of $\beta = 2$ sub-populations. Although we do not have a theoretical explanation for the choice of α and β, it is worthwhile commenting the performed setting. Since in this algorithm α sub-populations are supposed to explore the decision space while β are supposed to exploit it, the choice of α and β should be made in order to efficiently balance the exploration and exploitation features of the algorithm. Our preliminary experiments have clearly shown that α should be greater than β. On the other hand, the role of the second family is very important and requires some computational effort. As a general guideline, on the basis of empirical observations, we suggest to set β about $20 - 40\%$ of m. With both 500 and 1,000-dimensional problems, the migration constant has been set to $p_{mig} = 0.5$, and the T_b parameter for the population reduction algorithm has been set to 60% of the total budget, i.e., $T_b = 300,000$ and $T_b = 600,000$ for 500 and 1,000 dimensions respectively. This proportion of the budget seems to guarantee a robust behavior of the algorithm: according to our interpretation, a too low value of T_b implies a too short duration of the first age, which makes the algorithm too exploitative during the process, promoting quick improvements in the fitness values, but also causing premature convergence. Conversely, if T_b is too high (e.g., the first age lasts for 80% of the duration of the optimization process), the algorithm is too explorative and the second age does not have the opportunity to exploit the promising search directions generated by the population size reduction algorithm. The other parameters of the population reduction algorithms have been set to $r_k = 0$ for all k values, and to $N_s = 4$ steps of reduction; see [9] for parameter setting.
- In order to evaluate the impact, in DDE-EEPF, of the injection of individuals from the second family of sub-populations into the first family, a variant of DDE-EEPF named Parallel Differential Evolution With Random Injections (PDE-WRI) has been run, under the exact same conditions as DDE-EEPF. The only

difference between DDE-EEPF and PDE-WRI is that, at the point when DDE-EEPF would send an individual from the second family to the first one, PDE-WRI introduces instead a new, randomly generated (uniform distribution) individual into the first family.

It is worthwhile commenting the choice of the population sizes $S_{pop} = 200$ and $S_{pop} = 400$. Although in [52] it is suggested to set the DE population size equal to about ten times the dimensionality of the problem, this indication is not confirmed by a recent study in [35] where it is shown that a population size lower than the dimensionality of the problem can be optimal in many cases.

Table 2 shows the average of the final results detected by each algorithm $\pm$ the standard deviations, for the 500 dimension case, for the DE with additional components. Table 3 shows the results for the $1,000$ dimension case. The best results are highlighted in bold face.

Results in Tables 2 and 3 show that the proposed DDE-EEPF seems promising in terms of final result, since it detected (on average) the best performing solutions in fourteen cases out of the twenty-four considered in this study in 500 dimensions. PDE obtained the best results only in two cases, DDE is the best in three cases and PDE-WRI in five cases. In 1,000 dimensions, DDE-EEPF detected on average the best performing solutions in nine cases out of twenty-four, DDE is the best in nine cases, and PDE and PDE-WRI win in three cases each.

In order to prove the statistical significance of the results, the two-tail unequal variance t-test has been applied according to the description given in [47] (see also [37]) for a confidence level of 0.95. Tables 4 and 5 show the results of the test. A "+" indicates the case when the DDE-EEPF statistically outperforms, for the corresponding test problem, the algorithm mentioned in the column; a "=" indicates that the pairwise comparison leads to the success of the t-test, i.e., the two algorithms have the same performance; a "-" indicates that DDE-EEPF is outperformed.

In the case of 500-dimensional problems, the t-test results show that the DDE-EEPF outperforms PDE in 54.1% of the cases, IBDDE in 95.8%, DDE in 75.0% and PDE-WRI in 58.3% of the cases. On the opposite, DDE-EEPF is outperformed by PDE in 33.3% of the cases, by IBDDE in 4.17%, by DDE in 8.33% and by PDE-WRI in 25.0% of the cases. In the 1,000-dimensional problems, the t-test results show that DDE-EEPF wins against PDE, IBDDE, DDE and PDE-WRI in 79.1%, 95.8%, 50.0% and 41.7% of the cases, respectively. It loses in 4.17%, 0.00%, 25.0% and 12.5% of the cases against PDE, IBDDE, DDE and PDE-WRI, respectively.

In order to carry out a numerical comparison of the convergence speed performance, for each test problem, the average final fitness value returned by the best performing algorithm G has been considered. Subsequently, the average fitness value at the beginning of the optimization process J has also been computed. The threshold value $THR = J - 0.95(G - J)$ has then been calculated. The value THR represents 95% of the decay in the fitness value of the algorithm with the best performance. If an algorithm succeeds during a certain run to reach the value THR, the run is said to be successful. For each test problem, the average amount of fitness evaluations $\bar{ne}$ required, for each algorithm, to reach THR has been computed. Subsequently, the Q-test (Q stands for Quality) described in [21] has been applied. For each test problem and each algorithm, the Q measure is computed as:

$$Q = \frac{\bar{ne}}{R} \tag{5}$$

Table 2 Average final fitness values ± standard deviations for 500 dimensions problems

	PDE	IBDDE	DDE
Ackley	$1.62e-01 \pm 1.67e-02$	$3.55e+00 \pm 2.96e-02$	$1.51e-01 \pm 6.96e-02$
Alpine	$8.88e+01 \pm 1.26e+01$	$1.21e+03 \pm 3.43e+01$	$1.50e+02 \pm 4.35e+01$
Ax.-par. hyp.-ell.	$3.68e+03 \pm 6.84e+02$	$7.27e+05 \pm 3.70e+04$	$4.09e+03 \pm 4.73e+03$
DeJong	$1.92e+01 \pm 3.57e+00$	$3.19e+03 \pm 2.67e+02$	$1.61e+01 \pm 1.15e+01$
DropWave	$-4.11e-03 \pm 3.96e-04$	$-1.13e-03 \pm 8.03e-05$	$-2.56e-03 \pm 2.69e-04$
Griewangk	$5.62e+02 \pm 1.09e+01$	$1.13e+04 \pm 7.65e+02$	$5.68e+02 \pm 4.86e+01$
Michalewicz	$-3.06e+02 \pm 5.68e+00$	$-9.13e+01 \pm 2.09e+00$	$-2.60e+02 \pm 8.56e+00$
Pathological	$\mathbf{-3.34e+02 \pm 5.98e+00}$	$-3.34e+02 \pm 7.40e+00$	$-3.06e+02 \pm 7.28e+00$
Rastrigin	$1.91e+03 \pm 9.94e+01$	$7.78e+03 \pm 1.55e+02$	$2.73e+03 \pm 2.28e+02$
Rosenbrock	$2.11e+03 \pm 1.77e+02$	$1.35e+05 \pm 1.61e+04$	$\mathbf{1.82e+03 \pm 6.14e+02}$
Schwefel	$-1.30e+05 \pm 3.17e+03$	$-4.48e+04 \pm 8.15e+02$	$-1.06e+05 \pm 4.19e+03$
Sum of powers	$1.06e-05 \pm 5.19e-05$	$3.09e-01 \pm 1.67e-01$	$1.02e-03 \pm 2.51e-03$
Tirronen	$-1.57e+00 \pm 3.44e-02$	$-1.01e+00 \pm 1.90e-02$	$-1.38e+00 \pm 6.76e-02$
Rt. Ackley	$2.15e-01 \pm 2.50e-02$	$3.53e+00 \pm 3.82e-02$	$1.94e-01 \pm 7.21e-02$
Rt. Alpine	$1.03e+02 \pm 9.74e+00$	$1.23e+03 \pm 4.13e+01$	$1.39e+02 \pm 2.64e+01$
Rt. Ax.-par. hyp.-ell.	$4.90e+03 \pm 7.92e+02$	$7.12e+05 \pm 3.97e+04$	$\mathbf{3.87e+03 \pm 2.20e+03}$
Rt. Griewangk	$5.66e+02 \pm 1.03e+01$	$1.10e+04 \pm 6.18e+02$	$5.56e+02 \pm 3.75e+01$
Rt. Michalewicz	$-1.76e+02 \pm 7.76e+00$	$-4.99e+01 \pm 1.62e+00$	$-1.37e+02 \pm 7.98e+00$
Rt. Pathological	$-1.21e+02 \pm 7.30e+00$	$-2.11e+01 \pm 1.05e+00$	$-1.01e+02 \pm 1.20e+01$
Rt. Rastrigin	$1.95e+03 \pm 1.51e+02$	$7.85e+03 \pm 2.27e+02$	$2.70e+03 \pm 2.60e+02$
Rt. Rosenbrock	$1.66e+03 \pm 1.53e+02$	$1.47e+05 \pm 1.56e+04$	$\mathbf{1.45e+03 \pm 4.68e+02}$
Rt. Schwefel	$-1.65e+05 \pm 4.74e+03$	$-5.31e+04 \pm 1.87e+03$	$-1.27e+05 \pm 7.80e+03$
Rt. Sum of powers	$1.06e-05 \pm 5.20e-05$	$2.13e+15 \pm 7.15e+15$	$2.27e-03 \pm 6.51e-03$
Rt. Tirronen	$\mathbf{-1.24e+00 \pm 7.73e-02}$	$-5.08e-01 \pm 2.45e-02$	$-9.31e-01 \pm 9.41e-02$

	PDE-WRI	DDE-EEPF
Ackley	$1.45e-01 \pm 1.76e-02$	$\mathbf{1.06e-01 \pm 1.05e-02}$
Alpine	$6.29e+01 \pm 9.95e+00$	$\mathbf{4.34e+01 \pm 6.87e+00}$
Ax.-par. hyp.-ell.	$2.77e+03 \pm 5.37e+02$	$\mathbf{1.63e+03 \pm 2.39e+02}$
DeJong	$1.57e+01 \pm 2.26e+00$	$\mathbf{8.22e+00 \pm 1.01e+00}$
DropWave	$-3.45e-03 \pm 2.08e-04$	$\mathbf{-5.96e-03 \pm 4.16e-04}$
Griewangk	$5.52e+02 \pm 1.09e+01$	$\mathbf{5.28e+02 \pm 4.22e+00}$
Michalewicz	$-3.39e+02 \pm 4.77e+00$	$\mathbf{-3.49e+02 \pm 4.98e+00}$
Pathological	$-3.31e+02 \pm 6.02e+00$	$-3.27e+02 \pm 5.70e+00$
Rastrigin	$1.62e+03 \pm 8.41e+01$	$\mathbf{1.24e+03 \pm 5.74e+01}$
Rosenbrock	$2.37e+03 \pm 1.98e+02$	$2.24e+03 \pm 2.10e+02$
Schwefel	$-1.41e+05 \pm 2.78e+03$	$\mathbf{-1.53e+05 \pm 2.20e+03}$
Sum of powers	$\mathbf{3.35e-06 \pm 1.05e-05}$	$5.84e-05 \pm 2.02e-04$
Tirronen	$-1.49e+00 \pm 4.05e-02$	$\mathbf{-1.58e+00 \pm 4.79e-02}$
Rt. Ackley	$2.19e-01 \pm 2.93e-02$	$\mathbf{1.80e-01 \pm 2.15e-02}$
Rt. Alpine	$\mathbf{9.85e+01 \pm 9.57e+00}$	$1.33e+02 \pm 1.31e+01$
Rt. Ax.-par. hyp.-ell.	$4.81e+03 \pm 7.48e+02$	$3.99e+03 \pm 6.32e+02$
Rt. Griewangk	$5.66e+02 \pm 9.94e+00$	$\mathbf{5.49e+02 \pm 6.42e+00}$
Rt. Michalewicz	$-1.95e+02 \pm 6.02e+00$	$\mathbf{-1.96e+02 \pm 5.58e+00}$
Rt. Pathological	$\mathbf{-1.50e+02 \pm 7.08e+00}$	$-1.11e+02 \pm 1.22e+01$
Rt. Rastrigin	$\mathbf{1.85e+03 \pm 1.14e+02}$	$2.13e+03 \pm 1.59e+02$
Rt. Rosenbrock	$1.84e+03 \pm 2.05e+02$	$1.89e+03 \pm 2.13e+02$
Rt. Schwefel	$\mathbf{-1.66e+05 \pm 3.83e+03}$	$-1.57e+05 \pm 5.02e+03$
Rt. Sum of powers	$2.15e-09 \pm 1.24e-08$	$\mathbf{1.87e-09 \pm 1.22e-08}$
Rt. Tirronen	$-1.13e+00 \pm 7.77e-02$	$-9.95e-01 \pm 9.07e-02$

Table 3 Average final fitness values ± standard deviations for 1,000 dimensions problems

	PDE	IBDDE	DDE
Ackley	$9.75e-01 \pm 4.77e-02$	$3.72e+00 \pm 1.61e-02$	$7.25e-01 \pm 8.17e-02$
Alpine	$6.24e+02 \pm 3.28e+01$	$2.66e+03 \pm 6.16e+01$	$3.85e+02 \pm 4.99e+01$
Ax.-par. hyp.-ell.	$1.81e+05 \pm 1.23e+04$	$3.20e+06 \pm 1.62e+05$	$\mathbf{1.06e+05 \pm 1.86e+04}$
DeJong	$4.66e+02 \pm 2.81e+01$	$6.29e+03 \pm 3.74e+02$	$\mathbf{2.83e+02 \pm 5.12e+01}$
DropWave	$-1.62e-03 \pm 9.29e-05$	$-5.20e-04 \pm 2.21e-05$	$-1.17e-03 \pm 9.73e-05$
Griewangk	$2.58e+03 \pm 1.03e+02$	$2.28e+04 \pm 1.55e+03$	$\mathbf{1.96e+03 \pm 1.80e+02}$
Michalewicz	$-4.18e+02 \pm 9.73e+00$	$-1.49e+02 \pm 2.24e+00$	$-4.43e+02 \pm 1.22e+01$
Pathological	$\mathbf{-6.39e+02 \pm 8.43e+00}$	$-1.87e+02 \pm 3.77e+01$	$-5.70e+02 \pm 1.30e+01$
Rastrigin	$6.65e+03 \pm 2.16e+02$	$1.69e+04 \pm 3.40e+02$	$6.18e+03 \pm 3.96e+02$
Rosenbrock	$1.34e+04 \pm 9.17e+02$	$2.90e+05 \pm 3.35e+04$	$\mathbf{8.96e+03 \pm 1.82e+03}$
Schwefel	$-1.91e+05 \pm 4.04e+03$	$-6.45e+04 \pm 1.08e+03$	$-1.86e+05 \pm 8.61e+03$
Sum of powers	$\mathbf{3.07e-06 \pm 1.01e-05}$	$1.07e+00 \pm 3.69e-01$	$4.68e-04 \pm 1.64e-03$
Tirronen	$\mathbf{-1.45e+00 \pm 2.04e-02}$	$-8.63e-01 \pm 1.40e-02$	$-1.35e+00 \pm 5.11e-02$
Rt. Ackley	$9.35e-01 \pm 5.37e-02$	$3.69e+00 \pm 4.53e-02$	$\mathbf{6.90e-01 \pm 7.83e-02}$
Rt. Alpine	$6.13e+02 \pm 2.58e+01$	$2.70e+03 \pm 8.77e+01$	$\mathbf{4.01e+02 \pm 4.19e+01}$
Rt. Ax.-par. hyp.-ell.	$1.59e+05 \pm 1.25e+04$	$3.31e+06 \pm 2.20e+05$	$\mathbf{8.52e+04 \pm 1.60e+04}$
Rt. Griewangk	$2.24e+03 \pm 9.40e+01$	$2.33e+04 \pm 1.52e+03$	$\mathbf{1.75e+03 \pm 1.60e+02}$
Rt. Michalewicz	$-2.45e+02 \pm 7.54e+00$	$-7.24e+01 \pm 2.19e+00$	$-2.29e+02 \pm 1.23e+01$
Rt. Pathological	$-1.45e+02 \pm 6.58e+00$	$-2.99e+01 \pm 1.56e+00$	$-1.80e+02 \pm 1.89e+01$
Rt. Rastrigin	$6.71e+03 \pm 2.04e+02$	$1.66e+04 \pm 5.33e+02$	$5.90e+03 \pm 4.07e+02$
Rt. Rosenbrock	$1.08e+04 \pm 7.14e+02$	$3.18e+05 \pm 3.03e+04$	$\mathbf{6.95e+03 \pm 9.68e+02}$
Rt. Schwefel	$-2.48e+05 \pm 7.65e+03$	$-7.18e+04 \pm 1.83e+03$	$-2.48e+05 \pm 1.06e+04$
Rt. Sum of powers	$1.00e-07 \pm 4.88e-07$	$8.67e+50 \pm 4.47e+51$	$2.02e-02 \pm 8.35e-02$
Rt. Tirronen	$-9.73e-01 \pm 4.86e-02$	$-3.74e-01 \pm 1.89e-02$	$-8.12e-01 \pm 6.75e-02$

	PDE-WRI	DDE-EEPF
Ackley	$7.33e-01 \pm 5.62e-02$	$\mathbf{6.06e-01 \pm 3.39e-02}$
Alpine	$3.76e+02 \pm 2.04e+01$	$\mathbf{3.66e+02 \pm 1.90e+01}$
Ax.-par. hyp.-ell.	$1.31e+05 \pm 9.03e+03$	$1.08e+05 \pm 8.53e+03$
DeJong	$3.42e+02 \pm 2.52e+01$	$3.26e+02 \pm 3.19e+01$
DropWave	$-1.45e-03 \pm 5.33e-05$	$\mathbf{-1.82e-03 \pm 1.08e-04}$
Griewangk	$2.17e+03 \pm 8.73e+01$	$2.13e+03 \pm 1.12e+02$
Michalewicz	$-5.04e+02 \pm 9.17e+00$	$\mathbf{-5.07e+02 \pm 9.84e+00}$
Pathological	$-6.37e+02 \pm 8.00e+00$	$-6.34e+02 \pm 9.41e+00$
Rastrigin	$5.13e+03 \pm 1.98e+02$	$\mathbf{5.01e+03 \pm 1.91e+02}$
Rosenbrock	$1.13e+04 \pm 7.02e+02$	$1.12e+04 \pm 6.85e+02$
Schwefel	$-2.32e+05 \pm 4.51e+03$	$\mathbf{-2.33e+05 \pm 4.31e+03}$
Sum of powers	$6.88e-06 \pm 2.94e-05$	$1.40e-04 \pm 5.34e-04$
Tirronen	$-1.44e+00 \pm 2.66e-02$	$-1.45e+00 \pm 1.94e-02$
Rt. Ackley	$7.81e-01 \pm 4.94e-02$	$7.01e-01 \pm 4.27e-02$
Rt. Alpine	$4.07e+02 \pm 2.58e+01$	$4.02e+02 \pm 2.48e+01$
Rt. Ax.-par. hyp.-ell.	$1.10e+05 \pm 9.44e+03$	$9.68e+04 \pm 7.37e+03$
Rt. Griewangk	$1.90e+03 \pm 6.19e+01$	$1.78e+03 \pm 5.05e+01$
Rt. Michalewicz	$-3.03e+02 \pm 7.15e+00$	$\mathbf{-3.05e+02 \pm 6.82e+00}$
Rt. Pathological	$\mathbf{-1.94e+02 \pm 7.64e+00}$	$-1.71e+02 \pm 1.01e+01$
Rt. Rastrigin	$5.31e+03 \pm 1.98e+02$	$\mathbf{5.27e+03 \pm 2.04e+02}$
Rt. Rosenbrock	$8.89e+03 \pm 5.79e+02$	$8.47e+03 \pm 5.78e+02$
Rt. Schwefel	$\mathbf{-2.81e+05 \pm 7.87e+03}$	$-2.80e+05 \pm 9.56e+03$
Rt. Sum of powers	$2.86e-08 \pm 1.53e-07$	$\mathbf{2.80e-08 \pm 1.53e-07}$
Rt. Tirronen	$\mathbf{-9.97e-01 \pm 6.33e-02}$	$-9.44e-01 \pm 9.02e-02$

Table 4 Results of the unequal variance t-test for 500 dimensions problems

	PDE	IBDDE	DDE	PDE-WRI
Ackley	+	+	+	+
Alpine	+	+	+	+
Ax.-par. hyp.-ell.	+	+	+	+
DeJong	+	+	+	+
DropWave	+	+	+	+
Griewangk	+	+	+	+
Michalewicz	+	+	+	+
Pathological	-	-	+	-
Rastrigin	+	+	+	+
Rosenbrock	-	+	-	+
Schwefel	+	+	+	+
Sum of powers	=	+	+	=
Tirronen	=	+	+	+
Rt. Ackley	+	+	=	+
Rt. Alpine	-	+	=	-
Rt. Ax.-par. hyp.-ell.	+	+	=	+
Rt. Griewangk	+	+	=	+
Rt. Michalewicz	+	+	+	=
Rt. Pathological	-	+	+	-
Rt. Rastrigin	-	+	+	-
Rt. Rosenbrock	-	+	-	=
Rt. Schwefel	-	+	+	-
Rt. Sum of powers	=	+	+	=
Rt. Tirronen	-	+	+	-

where the robustness R is the percentage of successful runs. It is clear that, for each test problem, the smallest value equals the best performance in terms of convergence speed. The value "∞" means that $R = 0$, i.e., the algorithm never reached the *THR*. It is important to remark that the Q-measure implicitly includes a piece of information on the computational time needed to reach a reasonably good performance. More explicitly, in order to determine the time that each algorithm requires in order to reach the threshold value it is enough to compute $Q \times$ *number of runs (50)* $\times$ *time of a single fitness evaluation*. The time of each fitness evaluation clearly depends on the test problem and on the hardware involved. The ∞ value means that the corresponding algorithm required an infinite time to reach the threshold.

Tables 6 and 7 show the Q values for 500-dimensional problems and 1,000-dimensional problems respectively. The best results are highlighted in bold face.

Regarding the Q-measures in Table 6, in 500 dimensions, the DDE-EEPF obtained the best results in 7 cases, while PDE, DDE and PDE-WRI obtained the best results in 4, 10 and 3 cases, respectively. In 1,000 dimensions (Table 7), DDE-EEPF obtained the best results in 8 cases, while PDE, DDE and PDE-WRI obtained the best results in 0, 12 and 4 cases, respectively. It is worthwhile commenting the DDE behavior: due to its grid structure, the DDE seems very fast in the early stages of the evolution but it tends to focus its search on not so promising areas of the decision space since it is often outperformed by PDE and DDE-EEPF in terms of quality of final solutions. Nevertheless, considering that the DDE was designed for a domain-specific application, it can be considered a rather robust algorithm.

Table 5 Results of the unequal variance t-test for 1,000 dimensions problems

	PDE	IBDDE	DDE	PDE-WRI
Ackley	+	+	+	+
Alpine	+	+	+	+
Ax.-par. hyp.-ell.	+	+	=	+
DeJong	+	+	-	+
DropWave	+	+	+	+
Griewangk	+	+	-	=
Michalewicz	+	+	+	=
Pathological	-	+	+	-
Rastrigin	+	+	+	+
Rosenbrock	+	+	-	=
Schwefel	+	+	+	=
Sum of powers	=	+	=	=
Tirronen	=	+	+	=
Rt. Ackley	+	+	=	+
Rt. Alpine	+	+	=	=
Rt. Ax.-par. hyp.-ell.	+	+	-	+
Rt. Griewangk	+	+	=	+
Rt. Michalewicz	+	+	+	=
Rt. Pathological	+	+	-	-
Rt. Rastrigin	+	+	+	=
Rt. Rosenbrock	+	+	-	+
Rt. Schwefel	+	+	+	=
Rt. Sum of powers	=	=	=	=
Rt. Tirronen	=	+	+	-

Regarding the IBDDE, this study highlights that although the algorithm was very promising in thirty dimensions as showed in [6], it seems to suffer from the curse of dimensionality and to lose its high-quality performance.

It is important to remark that the proposed DDE-EEPF displays in Tables 6 and 7 the smallest number of ∞ values, which means that there is only one case in which it is not competitive with the best algorithm in 500 dimensions, and zero case in 1,000 dimensions. For the remaining test problems, the DDE-EEPF is either the best or a competitive algorithm. Thus, the DDE-EEPF demonstrated the best performance in terms of robustness.

Figure 8 shows average performance trends of the five considered algorithms over a selection of the test problems listed in Table 1 in 500 dimensions.

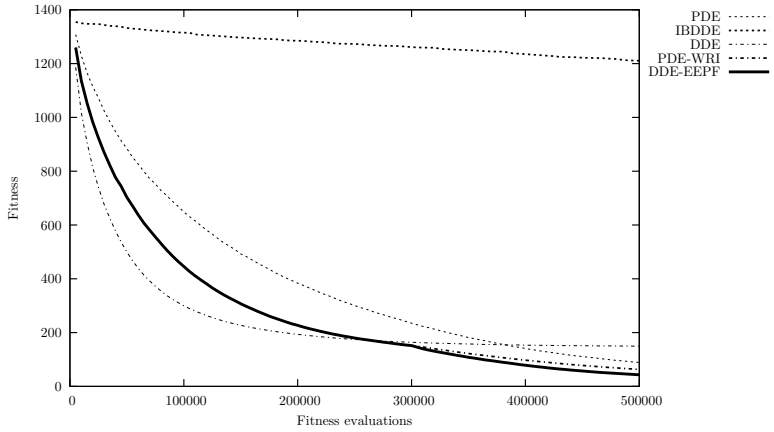
Figures 8(a), 8(c), 8(d) and 8(f) show that IBDDE improves only marginally. DDE, on the contrary, shows in those figures a very steep curve in the beginning of the optimization, outperforming at this point all the other algorithms, but quickly ceases to improve on its solutions. PDE's start is less steep than DDE's, but it continuously improves on its solutions and outperforms the algorithms above by a large margin. Finally, PDE-WRI and DDE-EEPF start with a steeper slope than PDE (although less steep than DDE's), but show hints that their improvement rates will deteriorate more quickly than PDE's. This can be explained by the fact that DDE-EEPF uses the same algorithm as PDE in the beginning, albeit with only three populations (instead of five for PDE) and with a higher migration constant which makes it more greedy and more prone to lose its ability to improve on its solutions.

Table 6 Results of the Q-test for 500 dimensions problems

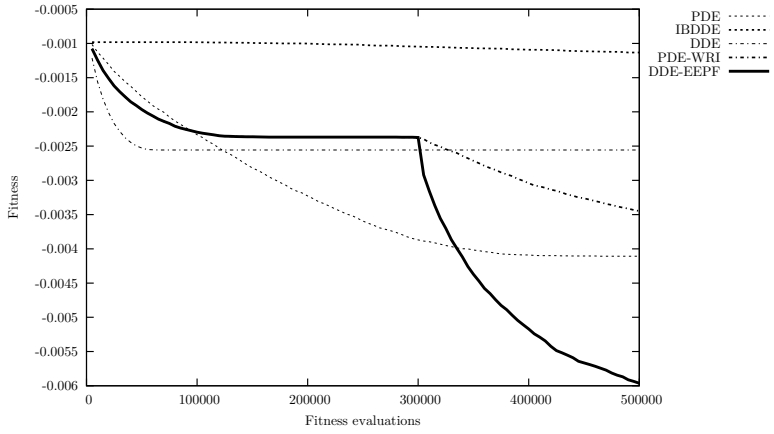
	PDE	IBDDE	DDE	PDE-WRI	DDE-EEPF
Ackley	4.01e+03	∞	3.46e+03	3.70e+03	**3.43e+03**
Alpine	4.77e+03	∞	1.25e+04	3.72e+03	**3.46e+03**
Ax.-par. hyp.-ell.	2.11e+03	∞	**1.18e+03**	1.67e+03	1.67e+03
DeJong	2.32e+03	∞	**1.32e+03**	1.94e+03	1.94e+03
DropWave	∞	∞	∞	∞	**5.80e+03**
Griewangk	2.30e+03	∞	**1.37e+03**	1.92e+03	1.92e+03
Michalewicz	∞	∞	∞	5.67e+03	**4.53e+03**
Pathological	**1.69e+03**	3.98e+03	3.92e+04	1.78e+03	1.80e+03
Rastrigin	∞	∞	∞	1.08e+04	**4.05e+03**
Rosenbrock	1.56e+03	∞	**7.12e+02**	1.22e+03	1.22e+03
Schwefel	∞	∞	∞	6.04e+04	**4.39e+03**
Sum of powers	2.32e+02	3.52e+04	**7.97e+01**	1.39e+02	1.39e+02
Tirronen	**2.70e+03**	∞	∞	8.79e+03	3.89e+03
Rt. Ackley	3.96e+03	∞	**3.23e+03**	3.85e+03	3.58e+03
Rt. Alpine	3.78e+03	∞	**3.08e+03**	3.14e+03	3.32e+03
Rt. Ax.-par. hyp.-ell.	1.98e+03	∞	**9.97e+02**	1.53e+03	1.53e+03
Rt. Griewangk	2.11e+03	∞	**1.14e+03**	1.67e+03	1.67e+03
Rt. Michalewicz	5.91e+04	∞	∞	5.11e+03	**4.75e+03**
Rt. Pathological	∞	∞	∞	**5.33e+03**	∞
Rt. Rastrigin	4.32e+03	∞	9.63e+04	**3.87e+03**	5.79e+03
Rt. Rosenbrock	1.38e+03	∞	**5.97e+02**	1.01e+03	1.01e+03
Rt. Schwefel	4.32e+03	∞	∞	**4.00e+03**	1.01e+04
Rt. Sum of powers	**4.12e+00**	1.24e+01	4.28e+00	4.40e+00	4.40e+00
Rt. Tirronen	**3.98e+03**	∞	∞	1.40e+04	9.06e+04

Table 7 Results of the Q-test for 1,000 dimensions problems

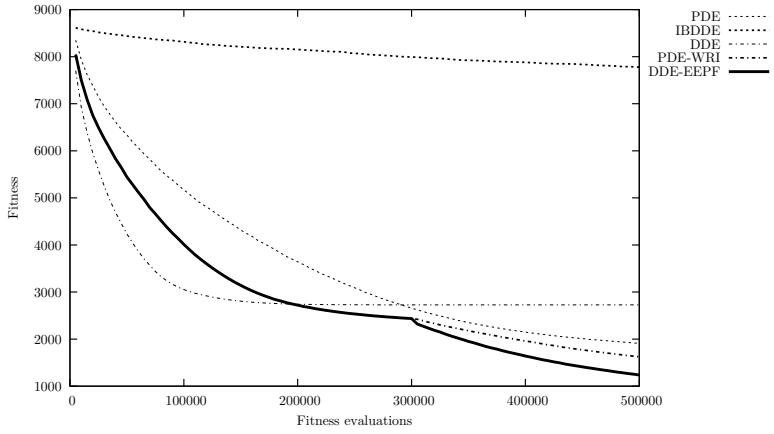
	PDE	IBDDE	DDE	PDE-WRI	DDE-EEPF
Ackley	∞	∞	1.24e+04	1.28e+04	**8.88e+03**
Alpine	∞	∞	**6.68e+03**	7.97e+03	7.90e+03
Ax.-par. hyp.-ell.	7.53e+03	∞	**4.31e+03**	5.83e+03	5.83e+03
DeJong	8.04e+03	∞	**4.77e+03**	6.45e+03	6.43e+03
DropWave	7.74e+04	∞	∞	∞	**1.26e+04**
Griewangk	8.03e+03	∞	**4.84e+03**	6.45e+03	6.43e+03
Michalewicz	∞	∞	∞	9.69e+03	**9.59e+03**
Pathological	6.79e+03	∞	∞	**4.99e+03**	4.99e+03
Rastrigin	∞	∞	8.16e+04	8.45e+03	**8.30e+03**
Rosenbrock	5.58e+03	∞	**2.60e+03**	4.23e+03	4.23e+03
Schwefel	∞	∞	∞	8.82e+03	**8.72e+03**
Sum of powers	6.84e+02	∞	**1.90e+02**	4.06e+02	4.06e+02
Tirronen	7.15e+03	∞	7.74e+03	5.84e+03	**5.70e+03**
Rt. Ackley	4.95e+05	∞	**8.04e+03**	1.04e+04	8.77e+03
Rt. Alpine	∞	∞	**5.73e+03**	7.62e+03	7.61e+03
Rt. Ax.-par. hyp.-ell.	7.26e+03	∞	**3.84e+03**	5.37e+03	5.37e+03
Rt. Griewangk	7.45e+03	∞	**4.21e+03**	5.69e+03	5.69e+03
Rt. Michalewicz	∞	∞	∞	9.63e+03	**9.17e+03**
Rt. Pathological	∞	∞	1.54e+04	**1.01e+04**	4.95e+05
Rt. Rastrigin	∞	∞	1.40e+04	8.16e+03	**8.08e+03**
Rt. Rosenbrock	4.96e+03	∞	**2.07e+03**	3.58e+03	3.58e+03
Rt. Schwefel	∞	∞	4.92e+05	**8.94e+03**	9.19e+03
Rt. Sum of powers	1.29e+01	3.62e+01	**1.12e+01**	1.38e+01	1.38e+01
Rt. Tirronen	1.22e+04	∞	∞	**8.68e+03**	1.37e+04



(a) Alpine

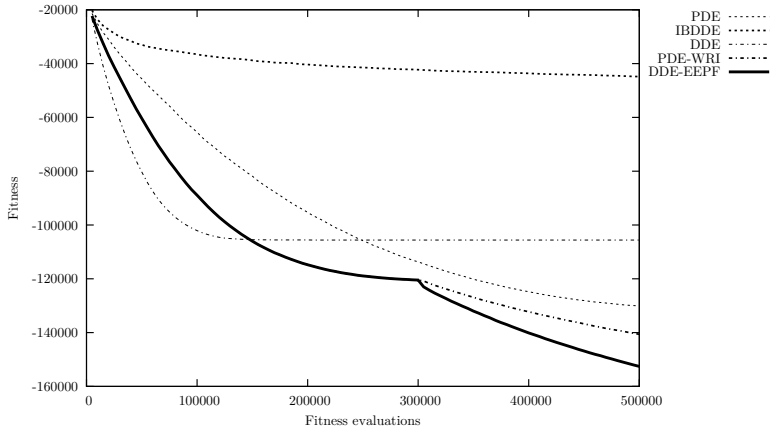


(b) DropWave

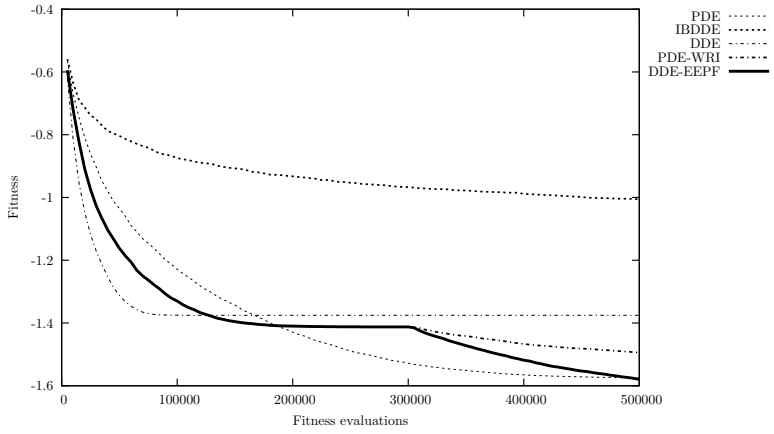


(c) Rastrigin

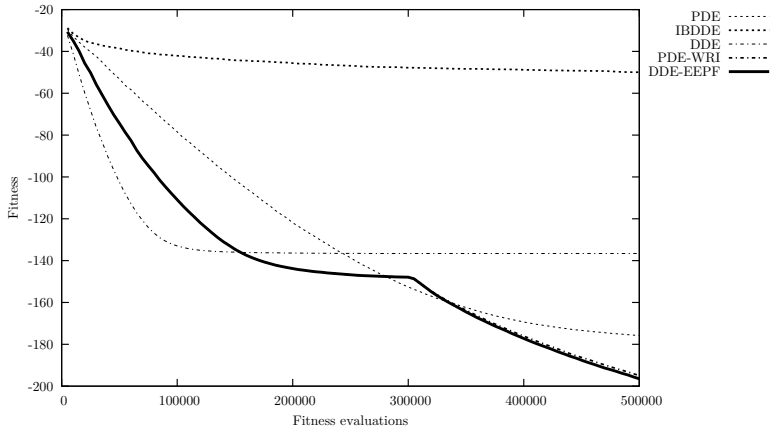
Fig. 8 Performance trends in 500 dimensions



(d) Schwefel



(e) Tirronen



(f) Rotated Michalewicz

Fig. 8 Performance trends in 500 dimensions (continued)

In 8(f), they are even slightly outperformed by PDE. However, at 300,000 fitness evaluations, PDE-WRI's and DDE-EEPF's first sub-population families start to be refreshed by individuals from the second sub-population family and by random new individuals, respectively. This shows on the curve as a "knee", a sudden change in the slope, which becomes again much steeper. In these four examples, at the end of the run, PDE-WRI and DDE-EEPF clearly outperform PDE and the other two algorithms by a large margin, DDE-EEPF also outperforming PDE-WRI, albeit only slightly in 8(a) and 8(f). Figure 8(b) shows the same behavior as described above, but PDE-WRI and DDE-EEPF lose very quickly their ability to make any improvement, until the time when the first family of sub-population is being refreshed. At this point, the slope of the curve becomes suddenly very much steeper. PDE-WRI is eventually outperformed by PDE, but DDE-EEPF outperforms PDE by a very large margin.

Figure 8(e) shows an example where PDE is on a par with DDE-EEPF and PDE-WRI: starting from 200,000 fitness evaluations, DDE-EEPF and PDE-WRI seem to become unable to improve on their solutions, while PDE continues improving. At 300,000 fitness evaluations, the sub-populations of the first family start receiving new individuals, causing DDE-EEPF and PDE-WRI to start improving again. DDE-EEPF eventually catches up with PDE around 500,000 fitness evaluations. One can also notice that on this function, IBDDE performs much better than on the other ones, without, however, being able to compete with the other algorithms.

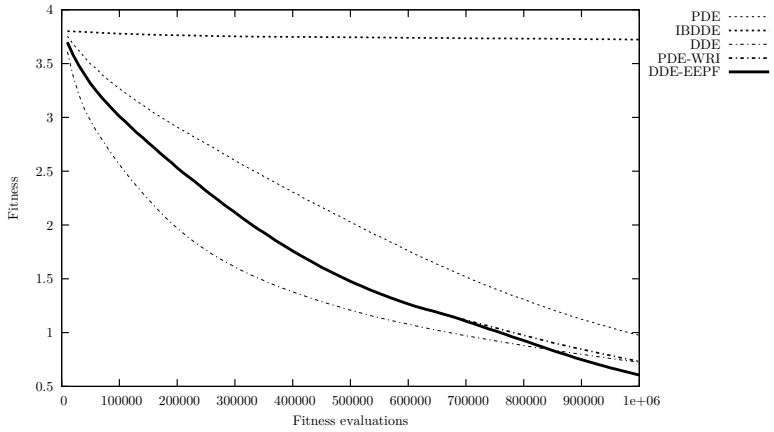
In these six examples of 500-dimensional problems, IBDDE, DDE and PDE show similar behaviors whatever the test problem when one considers the general shape of their curves. PDE-WRI and DDE-EEPF behave identically during the first age of the algorithms, while the improvement in DDE-EEPF brought by the solutions of the second family of sub-populations, compared to the introduction of new, random individuals in PDE-WRI, ranges from very small in two cases to considerably large in one other case.

Figure 9 shows average performance trends of the five considered algorithms over a selection of the test problems listed in Table 1, in 1,000 dimensions.

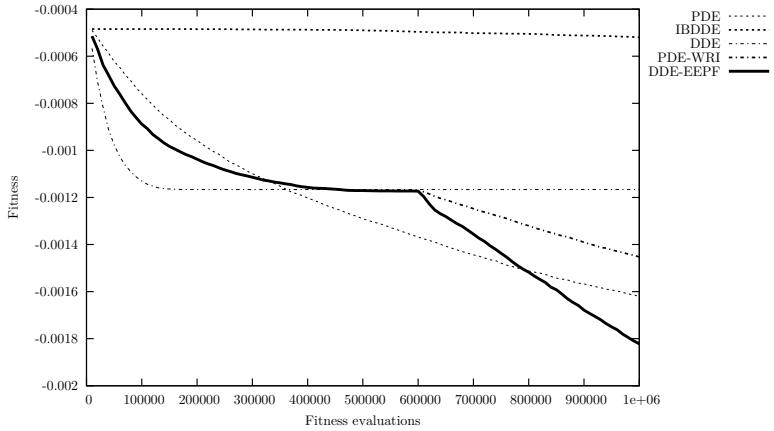
Similarly to the 500-dimensional problems, IBDDE in 1,000 dimensions improves on its solutions only marginally (see Figures 9(d), 9(e), 9(f)) or not at all (see Figures 9(a),9(b), 9(c)).

In Figure 9(b), PDE, DDE, PDE-WRI and DDE-EEPF present the same behavior as in 500 dimensions: DDE improves very quickly at first, but ceases to find any better solutions after about 200,000 fitness evaluations. DDE-EEPF and PDE-WRI seem to suffer from the same problem at first (although with a slower improvement rate than DDE), but at 600,000 fitness evaluations, the sub-populations of the first family start being refreshed, and the algorithms progress again, eventually finding better solutions than PDE. The difference between PDE-WRI and DDE-EEPF is unnoticeable on the graph.

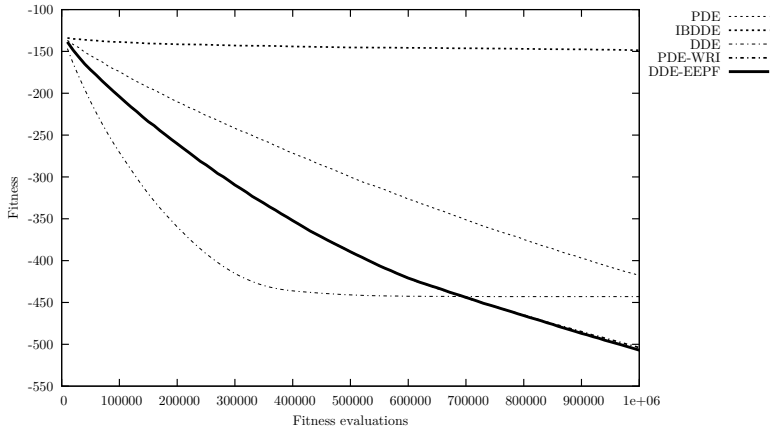
In Figures 9(c), 9(d), 9(e), and 9(f), PDE, DDE, PDE-WRI and DDE-EEPF exhibit very similar behaviors across the different test problems: DDE improves very quickly at the beginning, faster than the other two algorithms, but improves only marginally on its solutions after about 400,000 fitness evaluations. PDE, PDE-WRI and DDE-EEPF both show a steady rate of improvement, PDE-WRI's and DDE-EEPF's being better than PDE's, with a slight advantage for DDE-EEPF over PDE-WRI. Contrary to the 500-dimensional problems, DDE-EEPF's second age is not marked by a sharp change in the slope of the curve although the Figures do present a "knee" in the curve at 600,000 fitness evaluations, at the start of the second



(a) Ackley

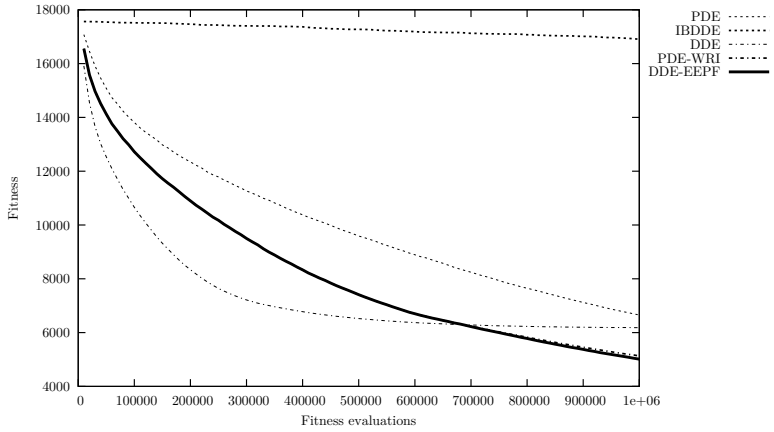


(b) Dropwave

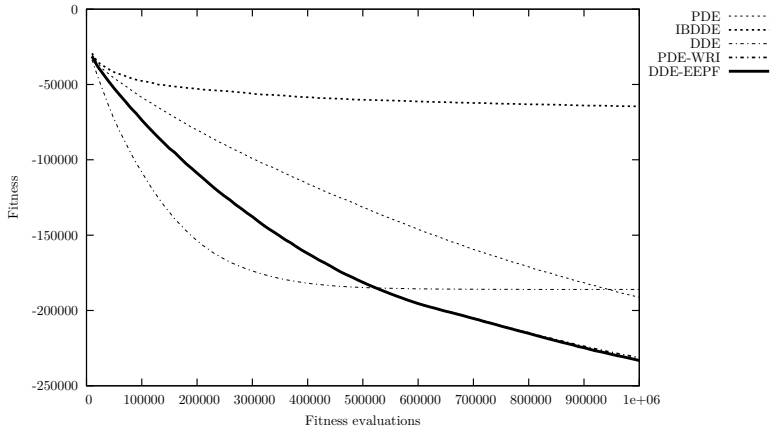


(c) Michalewicz

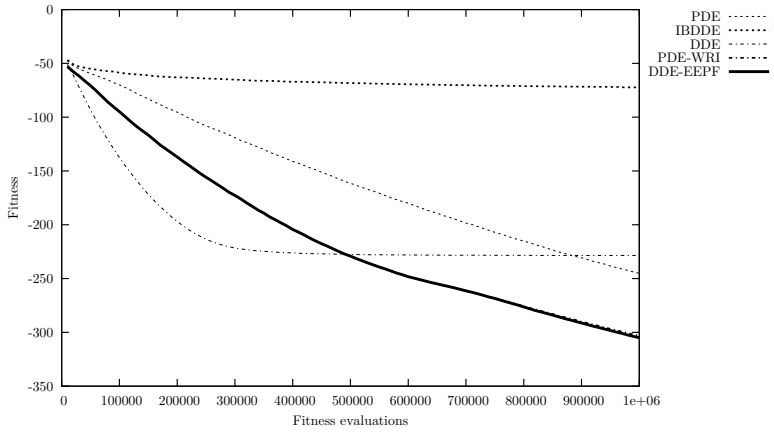
Fig. 9 Performance trends in 1,000 dimensions



(d) Rastrigin



(e) Schwefel



(f) Rotated Michalewicz

Fig. 9 Performance trends in 1,000 dimensions (continued)

age. The absence of a sharp "knee" is explained by the fact that in 500 dimensions, PDE-WRI and DDE-EEPF are not anymore improving on their solutions near the end of the first age, while in 1,000 dimensions they still are.

Figures 9(a), 9(c) and 9(d) show examples where DDE outperforms PDE and, in Figure 9(a), PDE-WRI as well: its sharper improvement rate in the beginning allows it to reach a better solution than the other algorithms, although all three reach similar solutions near the end. In Figure 9(a) one can notice that the difference between DDE-EEPF and PDE-WRI is perceptibly larger than it is in Figures 9(c), 9(d), 9(e), and 9(f).

In these six examples of 1,000-dimensional problems, as was the case with the 500-dimensional problems, the shapes of the curves presented by IBDDE, DDE and PDE are very similar across the various test problems. Moreover, while PDE was outperforming DDE on the 500-dimensional problems, the latter is better than the former in 1,000 dimensions.

The comparison between DDE-EEPF and PDE-WRI shows that although the random injection leads to satisfactory results both in 500 and 1,000 dimensions, and thus requires further investigation in the future, DDE-EEPF shows better results on average. This fact confirms that the injection of proper, high quality solutions is beneficial. We believe that a proper selection of these high quality solutions to be injected might have a great potential on a various set of problems.

6 Conclusion

The DDE-EEPF algorithm proposed in this paper is compared against three state-of-the-art, distributed DE algorithms and one designed ad hoc in this study. DDE-EEPF is characterized in that it uses two families of sub-populations instead of only one, as is the case in the three distributed DE algorithms. The first family has sub-populations of constant size organized into a ring topology, where the best individuals of a sub-population can migrate to another sub-population. The second family has sub-populations with a dynamic population size, which is reduced progressively. This scheme allows the first family to explore the search space, while the second family independently attempts to detect high performance solutions. These solutions then migrate to the first family in order to direct the exploration towards better solutions.

Numerical results show that DDE-EEPF outperforms the other algorithms in the majority of the twenty-four high-dimensional test problems. The injection of individuals from the second family of sub-populations into the first family has, in most of the cases, a beneficial effect as it gives a "second breath" to the process and allows it to explore the search space further. The numerical results presented in this paper additionally show that the use of a structured population improves greatly the performance of the DE algorithm compared to its original, serial version, when applied to high-dimensional problems.

Future development of this work will consider a generalization and extension of the two-family mechanism. A first direction of the research could aim at testing the proposed distributed system for Particle Swarm Optimizers and Covariance Matrix Adaptation Evolution Strategy. A second could involve different algorithms operating on the sub-populations, or include local search algorithms instead of the second family of sub-populations in order to generate high quality solutions.

References

1. E. Alba and B. Dorronsoro. The exploration/exploitation tradeoff in dynamic cellular genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 9(2):126–142, 2005.
2. E. Alba and S. Khuri. Sequential and distributed evolutionary algorithms for combinatorial optimization problems. In *Recent advances in intelligent paradigms and applications*, Studies in Fuzziness and Soft Computing, pages 211–233. Springer, 2003.
3. E. Alba and M. Tomassini. Parallelism and evolutionary algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5):443–462, 2002.
4. E. Alba and J. M. Troya. A survey of parallel distributed genetic algorithms. *Complexity*, 4(4), 1999.
5. E. Alba and J. M. Troya. Cellular evolutionary algorithms: Evaluating the influence of ratio. In *Parallel Problem Solving from Nature*, volume 1917 of *Lecture Notes in Computer Science*, pages 29–38. Springer, 2000.
6. J. Apolloni, G. Leguizamón, J. García-Nieto, and E. Alba. Island based distributed differential evolution: An experimental study on hybrid testbeds. In *Proceedings of the IEEE International Conference on Hybrid Intelligent Systems*, pages 696–701, 2008.
7. M. G. Arenas, P. Collet, A. E. Eiben, M. Jelasity, J. J. Merelo, B. Paechter, M. Preuß, and M. Schoenauer. A framework for distributed evolutionary algorithms. In *Parallel Problem Solving from Nature*, volume 2439 of *Lecture Notes in Computer Science*, pages 665–675. Springer, 2002.
8. T. C. Belding. The distributed genetic algorithm revisited. In *Proceedings of the International Conference on Genetic Algorithms*, pages 114–121. Morgan Kaufmann Publishers, 1995.
9. J. Brest and M. S. Maučec. Population size reduction for the differential evolution algorithm. *Applied Intelligence*, 29(3):228–247, 2008.
10. E. Cantú-Paz. A survey of parallel genetic algorithms. Technical Report 97003, IlliGAL, 1997.
11. E. Cantú-Paz. A survey of parallel genetic algorithms. *Calculateurs Paralleles, Reseaux et Systems Repartis*, 10(2):141–171, 1998.
12. E. Cantú-Paz. Topologies, migration rates, and multi-population parallel genetic algorithms. In *Proceedings of the Genetic and Evolutionary Computation Conference*, pages 91–98, 1999.
13. E. Cantú-Paz. *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, 2000.
14. E. Cantú-Paz. Migration policies, selection pressure, and parallel evolutionary algorithms. *Journal of Heuristics*, 7(4):311–334, 2001.
15. U. K. Chakraborty, editor. *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*. Springer, 2008.
16. A. E. Eiben and J. E. Smith. *Introduction to Evolutionary Computation*. Springer-verlag, Berlin, 2003.
17. I. D. Falco, A. D. Cioppa, D. Maisto, U. Scafuri, and E. Tarantino. Satellite image registration by distributed differential evolution. In *Applications of Evolutionary Computing*, volume 4448 of *Lectures Notes in Computer Science*, pages 251–260. Springer, 2007.
18. I. D. Falco, D. Maisto, U. Scafuri, E. Tarantino, and A. D. Cioppa. Distributed differential evolution for the registration of remotely sensed images. In *Proceedings of the IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 358–362, 2007.
19. I. D. Falco, U. Scafuri, E. Tarantino, and A. D. Cioppa. A distributed differential evolution approach for mapping in a grid environment. In *Proceedings of the IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pages 442–449, 2007.
20. H.-Y. Fan and J. Lampinen. A trigonometric mutation operation to differential evolution. *Journal of Global Optimization*, 27(1):105–129, 2003.
21. V. Feoktistov. *Differential Evolution in Search of Solutions*. Springer, 2006.
22. F. Fernández, M. Tomassini, and L. Vanneschi. An empirical study of multipopulation genetic programming. *Genetic Programming and Evolvable Machines*, 4(1):21–52, 2003.
23. F. Fernández, M. Tomassini, and L. Vanneschi. Saving computational effort in genetic programming by means of plagues. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2042–2049, 2003.

24. M. Giacobini, E. Alba, A. Tettamanzi, and M. Tomassini. Modeling selection intensity for toroidal cellular evolutionary algorithms. In *Genetic and Evolutionary Computation*, volume 3102 of *Lecture Notes in Computer Science*, pages 1138–1149. Springer, 2004.
25. M. Giacobini, E. Alba, and M. Tomassini. Selection intensity in asynchronous cellular evolutionary algorithms. In E. Cantú-Paz et al., editor, *Genetic and Evolutionary Computation*, volume 2723 of *Lecture Notes in Computer Science*, pages 955–966. Springer, 2003.
26. M. Giacobini, M. Tomassini, A. Tettamanzi, and E. Alba. Selection intensity in cellular evolutionary algorithms for regular lattices. *IEEE Transactions on Evolutionary Computation*, 9(5):489–505, 2005.
27. J. Grefenstette. Parallel adaptive algorithms for function optimization. Technical Report CS-81-19, Vanderbilt University, 1981.
28. R. Joshi and A. C. Sanderson. Minimal representation multisensor fusion using differential evolution. *IEEE Transactions on Systems, Man and Cybernetics, Part A*, 29(1):63–76, 1999.
29. K. N. Kozlov and A. M. Samsonov. New migration scheme for parallel differential evolution. In *Proceedings of the International Conference on Bioinformatics of Genome Regulation and Structure*, pages 141–144, 2006.
30. W. Kwedlo and K. Bandurski. A parallel differential evolution algorithm. In *Proceedings of the IEEE International Symposium on Parallel Computing in Electrical Engineering*, pages 319–324, 2006.
31. J. Lampinen. Differential evolution - new naturally parallel approach for engineering design optimization. In B. H. Topping, editor, *Developments in Computational Mechanics with High Performance Computing*, pages 217–228. Civil-Comp Press, 1999.
32. J. Lampinen and I. Zelinka. On stagnation of the differential evolution algorithm. In P. Oŝmera, editor, *Proceedings of 6th International Mendel Conference on Soft Computing*, pages 76–83, 2000.
33. P. Moscato and M. Norman. A competitive and cooperative approach to complex combinatorial search. Technical Report 790, 1989.
34. H. Mühlenbein, M. Schomisch, and J. Born. The parallel genetic algorithm as function optimizer. *Parallel Computing*, 17(6–7):619–632, 1991.
35. F. Neri and V. Tirronen. On memetic differential evolution frameworks: a study of advantages and limitations in hybridization. In *Proceedings of the IEEE World Congress on Computational Intelligence*, pages 2135–2142, 2008.
36. M. S. Nipteni, I. Valakos, and I. Nikolos. An asynchronous parallel differential evolution algorithm. In *Proceedings of the ERCOFTAC Conference on Design Optimisation: Methods and Application*, 2006.
37. NIST/SEMATECH. e-handbook of statistical methods. *http://www.itl.nist.gov/div898/handbook/*, 2003.
38. M. Nowostawski and R. Poli. Parallel genetic algorithm taxonomy. In *Proceedings of the International Conference on Knowledge-Based Intelligent Information Engineering Systems*, pages 88–92, 1999.
39. N. G. Pavlidis, D. K. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. N. Vrahatis. Spiking neural network training using evolutionary algorithms. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2190–2194, 2005.
40. V. P. Plagianakos, D. K. Tasoulis, and M. N. Vrahatis. A review of major application areas of differential evolution. In U. K. Chakraborty, editor, *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*, pages 197–238. Springer, 2008.
41. K. V. Price. Mechanical engineering design optimization by differential evolution. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 293–298. McGraw-Hill, 1999.
42. K. V. Price, R. Storn, and J. Lampinen. *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005.
43. W. Punch, E. Goodman, M. Pei, L. Chain-Shun, P. Hovland, and R. Enbody. Further research on feature selection and classification using genetic algorithms. In S. Forrest, editor, *Proceedings of the International Conference on Genetic Algorithms*, pages 557–564. Morgan Kaufmann, 1993.
44. A. K. Qin and P. N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791, 2005.

45. I. Rechemberg. *Evolutionstrategie: Optimierung Technisher Systeme nach prinzipien des Biologishen Evolution*. Fromman-Hozlboog Verlag, 1973.
46. G. Rudolph. Global optimization by means of distributed evolution strategies. In *Parallel Problem Solving from Nature*, volume 496 of *Lecture Notes in Computer Science*, pages 209–213. Springer, 1991.
47. G. D. Ruxton. The unequal variance t-test is an underused alternative to Student's t-test and the Mann-Whitney test. *Behavioral Ecology*, 17(4):688–690, 2006.
48. M. Salomon, G.-R. Perrin, F. Heitz, and J.-P. Armspach. Parallel differential evolution: Application to 3-d medical image registration. In K. V. Price, R. M. Storn, and J. A. Lampinen, editors, *Differential Evolution–A Practical Approach to Global Optimization*, Natural Computing Series, chapter 7, pages 353–411. Springer, 2005.
49. R. Storn. System design by constraint adaptation and differential evolution. *IEEE Transactions on Evolutionary Computation*, 3(1):22–34, 1999.
50. R. Storn. Designing nonstandard filters with differential evolution. *IEEE Signal Processing Magazine*, 22(1):103–106, 2005.
51. R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, ICSI, 1995.
52. R. Storn and K. Price. Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization*, 11:341–359, 1997.
53. D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis. Parallel differential evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 2023–2029, 2004.
54. V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi. A memetic differential evolution in filter design for defect detection in paper production. In *Applications of Evolutionary Computing*, volume 4448, pages 320–329. Springer, Berlin, Germany, 2007.
55. 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*, 16:529–555, 2008.
56. M. Tomassini. Parallel and distributed evolutionary algorithms: A review. In K. Miettinen, M. M. Mäkelä, P. Neittaanmäki, and J. Périaux, editors, *Evolutionary Algorithms in Engineering and Computer Science - Recent Advances in Genetic Algorithms, Evolution Strategies, Evolutionary Programming, Genetic Programming and Industrial Applications*. John Wiley and Sons, 1999.
57. M. Tomassini, L. Vanneschi, J. Cuendet, and F. Fernández. A new technique for dynamic size populations in genetic programming. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 486–493, 2004.
58. D. Whitley. Cellular genetic algorithms. In S. Forrest, editor, *Proceedings of the International Conference on Genetic Algorithms*, page 658, 1993.
59. D. Zaharie. Parameter adaptation in differential evolution by controlling the population diversity. In D. Petcu et al, editor, *Proceedings of the International Workshop on Symbolic and Numeric Algorithms for Scientific Computing*, pages 385–397, 2002.
60. D. Zaharie. Control of population diversity and adaptation in differential evolution algorithms. In D.Matousek and P. Osmera, editors, *Proceedings of MENDEL International Conference on Soft Computing*, pages 41–46, 2003.
61. D. Zaharie. A multipopulation differential evolution algorithm for multimodal optimization. In R. Matousek and P. Osmera, editors, *Proceedings of Mendel International Conference on Soft Computing*, pages 17–22, 2004.
62. D. Zaharie and G. Ciobanu. Distributed evolutionary algorithms inspired by membranes in solving continuous optimization problems. In *Membrane Computing*, volume 4361 of *Lecture Notes in Computer Science*, pages 536–553. Springer, 2006.
63. D. Zaharie and D. Petcu. Parallel implementation of multi-population differential evolution. In *Proceedings of the NATO Advanced Research Workshop on Concurrent Information Processing and Computing*, pages 223–232. IOS Press, 2003.
64. K. Zielinski and R. Laur. Stopping criteria for differential evolution in constrained single-objective optimization. In U. K. Chakraborty, editor, *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*, pages 111–138. Springer, 2008.
65. K. Zielinski, P. Weitkemper, R. Laur, and K.-D. Kammeyer. Parameter study for differential evolution using a power allocation problem including interference cancellation. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1857–1864, 2006.

PIII

SCALE FACTOR INHERITANCE MECHANISM IN DISTRIBUTED DIFFERENTIAL EVOLUTION

by

Matthieu Weber, Ville Tirronen and Ferrante Neri 2010

In Soft Computing - A Fusion of Foundations, Methodologies and Applications,
volume 14, number 11, pages 1187–1207

Soft Computing manuscript No.
(will be inserted by the editor)

Scale Factor Inheritance Mechanism in Distributed Differential Evolution

Matthieu Weber · Ville Tirronen · Ferrante Neri

Received: date / Accepted: date

Abstract This paper proposes a distributed differential evolution which employs a novel self-adaptive scheme, namely scale factor inheritance. In the proposed algorithm, the population is distributed over several sub-populations allocated according to a ring topology. Each sub-population is characterized by its own scale factor value. With a probabilistic criterion, that individual displaying the best performance is migrated to the neighbor population and replaces a pseudo-randomly selected individual of the target sub-population. The target sub-population inherits not only this individual but also the scale factor if it seems promising at the current stage of evolution. In addition, a perturbation mechanism enhances the exploration feature of the algorithm.

The proposed algorithm has been run on a set of various test problems and then compared to two sequential differential evolution algorithms and three distributed differential evolution algorithms recently proposed in literature and representing state-of-the-art in the field. Numerical results show that the proposed approach

This research is supported by the Academy of Finland, Akatemiatutkija 00853, Algorithmic Design Issues in Memetic Computing.

Matthieu Weber
Tel.: +358-14-2603056
E-mail: matthieu.weber@jyu.fi

Ville Tirronen
Tel.: +358-14-2604987
E-mail: ville.tirronen@jyu.fi

Ferrante Neri
Tel.: +358-14-2602764
E-mail: ferrante.neri@jyu.fi

University of Jyväskylä
Department of Mathematical Information Technology
P.O. Box 35 (Agora)
40014 University of Jyväskylä
Finland
Fax: +358-14-2604981

seems very efficient for most of the analyzed problems, and outperforms all other algorithms considered in this study.

Keywords Differential Evolution · Distributed Evolutionary Algorithms · Evolutionary Algorithms · Continuous Optimization

1 Introduction

Differential Evolution (DE, see [1], [2], and [3]) is a reliable and versatile function optimizer which displays a solid performance for diverse continuous optimization problems. DE is a very interesting population based metaheuristic having mixed features. Due to its recombination and selection features DE can be seen as an Evolutionary Algorithm (EA). On the other hand, a DE structure tends to generate an individual with an above average performance which leads the exploration search, similar to Swarm Intelligence Algorithms (SIA). In addition, DE, unlike EAs, generates offspring by perturbing the solutions with a scaled difference of two randomly selected population vectors, instead of recombining the solutions by means of a probabilistic criterion. Finally, DE employs a very peculiar survivor selection scheme, namely one to one spawning. This selection scheme allows replacement of an individual only if the offspring outperforms its corresponding parent.

Regardless of its classification, DE has proven to have a very good performance on various real-world problems. For example, in [4] a DE application to the multisensor fusion problem is given. In [5], and [6] DE applications to power electronics are presented. In [7] an application of DE to chemical engineering is proposed. In [8] a filter design is carried out by DE.

Reasons for success of the DE can be found in its simplicity and ease of implementation, while at the same time demonstrating reliability and high performance. In addition, the fact that only three parameters require tuning greatly contributes to the rapid diffusion of DE schemes among computer scientists and practitioners.

Although the DE undoubtedly has a great potential, setting of the control parameters is not a trivial task, since it has a heavy impact on the algorithmic performance. Thus, over the years, the DE community has intensively investigated the topic of parameter setting. Several studies have been reported, e.g. in [9], [10], [11], [12], and [13], and led to contradictory conclusions. In other words, in accordance with the No Free Lunch Theorem, [14], an efficient DE parameter setting is very prone to problems, as the studies in [15], [11], and [16] confirm.

In order to overcome the problem of the setting, some algorithms which employ adaptive and self-adaptive parameter settings have recently been proposed in literature. In [17] and [18], a variable population size is presented The adaptive population size approach has been recently improved in two different implementations reported in [19]. Another scheme, which proposes a progressive population size reduction in DE, has been proposed in [20]. A variable population size DE, based on a fitness diversity adaptation is proposed in [21]. An adaptive scheme for the DE scale factor is presented in [22]. An automatic update of the scale factor has been proposed in [23]. By following a similar line of thought, in [24], [25] and [26], a normal distribution is employed in order to perform a self-adaptation on the parameters F and CR. A Cauchy distribution in the self-adaptive scheme proposed in [27] and [28]. An alternative kind of self-adaptation which employs the so called chaos mutation is

proposed in [29]. A controlled randomization of scale factor and crossover rate has been proposed in [30].

In addition, since the DE algorithm suffers from many real world conditions, e.g. high dimensionality and noisy problems, some modifications on the standard DE scheme can significantly improve upon its performance. Modern DE based algorithms can be divided into the two following categories:

1. DE integrating an extra component. This class includes those algorithms which use the DE as an evolutionary framework in which it is assisted by additional algorithmic components, e.g. local searchers or extra operators (see [31], [32], [33], [34], [35], and [36]). The algorithms belonging to this class can be clearly decomposed as a DE framework and additional components.
2. Modified structures of DE. This class includes those algorithms which make a substantial modification within the DE structure, in the search logic, the selection etc. Some examples are given in [37] and [38]

A popular way to enhance the DE performance by structurally modifying the algorithmic functioning is through employment of structured populations. In other words, the population individuals are distributed over several sub-populations which evolve independently and interact by exchanging data and information details and contribute to a unique simultaneous evolution.

In [39] a distributed DE scheme employing a ring topology (the cores are interconnected in a circle and the migrations occur following the ring) has been proposed for the training of a neural network. In [40], an example of DE parallelization is given for a medical imaging application. A few famous examples of distributed DE are presented in [41], [42], and [43]; in these papers the migration mechanism as well as the algorithmic parameters are adaptively coordinated according to criterion based on genotypical diversity. In paper [44], a distributed DE for preserving diversity in the niches is proposed in order to solve multi-modal optimization problems. In [45], a distributed DE characterized by a ring topology and the migration of individuals with the best performance, to replace random individuals of the neighbor sub-population, has been proposed. An application of the algorithm in [45] for training of a neural network has been presented in [46]. Following similar logic, paper [47] proposes a distributed DE where the computational cores are arranged according to a ring topology and, during migration, the best individual of a sub-population replaces the oldest member of the neighboring population. In [48], [49], and [50] a distributed DE has been designed for the image registration problem. In these papers, a computational core acts as a master by collecting the best individuals detected by the various sub-populations running in slave cores. The slave cores are connected in a grid and a migration is arranged among neighbor sub-populations. In [51], a distributed DE which modifies the scheme proposed in [45] has been presented. In [51], the migration is based on a probabilistic criterion depending on five parameters. It is worthwhile mentioning that some parallel implementations of sequential (without structured population) DE are also available in literature, see [52]. An investigation of DE parallelization is given in [53].

This paper focuses on Distributed Differential Evolutions and proposes a novel distributed algorithm. The proposed algorithm distributes its individuals within sub-populations arranged according to a ring topology. Each sub-population is characterized by its own scale factor. According to a simple probabilistic criterion, the migration of individual with the best performance and its associated scale factor

occurs between neighbor sub-populations (following the ring topology). At each migration, the scale factor is also perturbed by means of a normal distribution. This paper is based on the idea that the scale factor is a determinant element within the DE search strategy. Thus, a successful search strategy can be inherited by the other sub-populations and propagated throughout the ring. A probabilistic perturbation enhances the exploration pressure of the algorithm.

The remainder of this paper is organized in the following way. Section 2 describes the working principles of DE. Section 3 gives a short description of recently presented distributed versions of DE and introduces algorithms employed for comparison in the experimental section. Section 4 describes the proposed algorithm and discusses its algorithmic principle of functioning. Section 5 shows the experimental setup and numerical results of the present study. Section 6 gives the conclusions of this paper.

2 Sequential Differential Evolution

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

According to its original definition given in [1], the DE consists of the following steps. An initial sampling of S_{pop} individuals is performed pseudo-randomly with a uniform distribution function within the decision space D. At each generation, for each individual x_i of the S_{pop}, three individuals x_r, x_s and x_t are pseudo-randomly extracted from the population. According to the DE logic, a provisional offspring x'_{off} is generated by mutation as:

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

where $F \in [0, 1+[$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated. With $F \in [0, 1+[$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1, see [2]. While there is no theoretical upper limit for F, effective values are rarely greater than 1.0. The mutation scheme shown in Equation (1) is also known as DE/rand/1. Other variants of the mutation rule have been subsequently proposed in literature, see [54]:

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

where x_{best} is the solution with the best performance among the individuals of the population, x_u and x_v are two additional pseudo-randomly selected individuals. It is worthwhile to mention the rotation invariant mutation shown in [55]:

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

where K is is the combination coefficient, which, as suggested in [55], should be chosen with a uniform random distribution from $[0, 1]$ and $F' = K \cdot F$. For this special mutation the mutated solution does not undergo the crossover operation described below.

```
generate S_pop individuals of the initial population pseudo-randomly
while budget condition do
  for i = 1 : S_pop do
    compute f(x_i)
  end for
  for i = 1 : S_pop do
    {** Mutation **}
    select three individuals x_r, x_s, and x_t
    compute x'_off = x_t + F(x_r − x_s)
    {** Crossover **}
    x_off = x'_off
    for j = 1 : n do
      generate rand(0, 1)
      if rand(0, 1) < CR then
        x_off,j = x_i,j
      end if
    end for
    {** Selection **}
    if f(x_off) ≤ f(x_i) then
      save index for replacement x_i = x_off
    end if
  end for
  perform replacements
end while
```

Fig. 1 Pseudo-code of DE/rand/1/bin

Recently, in [2], a new mutation strategy has been defined. This strategy, namely DE/rand/1/either-or, consists of the following:

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

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

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

$$x_{off,j} = \begin{cases} x_{i,j} & \text{if} \quad rand\,(0,1) < CR \\ x'_{off,j} & otherwise \end{cases} \tag{3}$$

where $rand\,(0,1)$ is a random number between 0 and 1; j is the index of the gene under examination. The crossover described in eq. (3) is known as binary crossover (simply indicated with ´´bin") and is the most common crossover scheme. It can be remarked that also the exponential crossover, see [2], is used in some cases.

The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) \leq f(x_i)$; otherwise no replacement occurs. It must be remarked that although the replacement indexes are saved one by one, during generation, actual replacements occur all at once at the end of the generation. For the sake of clarity, the pseudo-code highlighting working principles of the DE is shown in Figure 1.

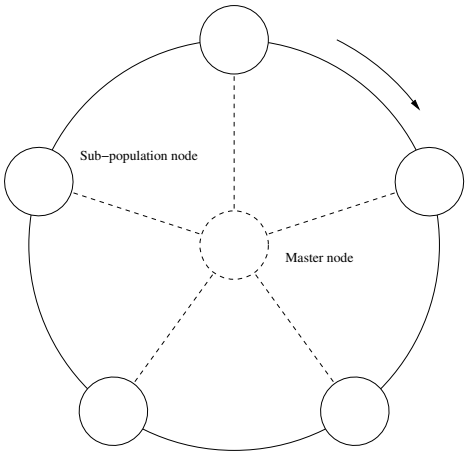


Fig. 2 Unidirectional ring topology in the Parallel Differential Evolution algorithm

3 Distributed Differential Evolution: Recently Developed Algorithms

This section describes three distributed algorithms based on a DE structure recently proposed in literature. The algorithms described in this section are, according to our judgement, representative the state-of-the-art structured DE algorithms and have been included in the benchmark for comparing the performance of the proposed approach. Although the notation can generate some confusion, i.e., all algorithms are distributed and can easily be parallelized, we decided to indicate these according to original terminology defined by their respective authors.

3.1 Parallel Differential Evolution

In [45], the problem of parallelization for DE schemes has been studied through an experimental analysis and an algorithm, namely Parallel Differential Evolution (indicated here with PDE) has been proposed.

The original PDE implementation uses the Parallel Virtual Machine (PVM), allowing multiple computers (called *nodes*) to be organized as a cluster and exchange arbitrary messages. The PDE algorithm is organized around one master node and m sub-populations running each on one node, and organized as a unidirectional ring, as illustrated in Figure 2. It must be noted that although the logical topology is a ring which does not contain the master node, the actual topology is a star, where all communications (i.e., the migrations of individuals) are passing through the master.

The S_{pop} individuals constituting the populations are distributed over the m sub-populations composing the ring. Each sub-population is composed of $\frac{S_{pop}}{m}$ individuals. Each sub-population runs a regular DE algorithm while the master node coordinates the migration of individuals between sub-populations. On each generation, the sub-population has a given probability ϕ to send a copy of its best individual to its next neighbor sub-population in the ring. When migration occurs, the migrating individual replaces a pseudo-randomly selected individual belonging to the target

```
spawn N sub-populations, each one on a different processor
for each generation do
   receive an individual from each sub-population
   for each received individual do
      if rand(0, 1) < φ then
         send the individual to the next sub-population in the ring
      end if
   end for
   if the stop criterion for the objective function is met then
      send a termination signal to all the sub-populations
   end if
end for
```

(a) At the master node

```
for each generation do
   perform a DE generation
   send a copy of the best individual to the master node
   if a migrated individual has been received then
      replace a random individual, different from the best, by this migrated individual
   end if
   if a termination signal has been received then
      terminate the execution
   end if
end for
```

(b) At each sub-population

Fig. 3 Pseudo-code of PDE

sub-population. Figure 3 describes the behavior of both the master node and the sub-populations in more detail.

The DE variant run by each sub-population is the same across all the sub-populations. In [45], six mutation strategies have been compared, namely DE/best/1, DE/rand/1, DE/cur-to-best/1, DE/best/2, DE/rand/2 described in Section 2, as well as the trigonometric operator described in [31]. Each strategy is used with different values of the migration constant ϕ and compared over seven test functions whose dimensions vary between 2 and 30. The results in [45] showed that DE/best/1 is the most efficient mutation strategy and quite stable across different values of ϕ for the low dimensional problems analyzed.

3.2 Island Based Distributed Differential Evolution

In [51] a distributed DE algorithm, namely Island Based Distributed Differential Evolution (IBDDE) has been proposed. The IBDDE algorithm is a modified version of PDE described in Subsection 3.1. In IBDDE, the population, having size S_{pop}, is structured in m sub-populations. Thus, each sub-population is composed of $\frac{S_{pop}}{m}$ individuals. The migration policy is then defined as a five-tuple $\mathcal{M} = (\gamma, \rho, \phi_s, \phi_r, \tau)$. $\gamma \in \mathbb{N}$ is the number of generations between two migrations, $\rho \in \mathbb{N}$ is the number of individuals which migrate from a sub-population P during each migration, ϕ_s is the selection function which, applied to a sub-population, returns the migrating individuals $v_g'^i$, ϕ_r is the replacement function that selects individuals to be replaced by the immigrants in the receiving sub-population, and τ is the topological

```
initialize(P)
while the stopping condition is not met do
   perform a DE generation
   if the last migration was γ generations ago then
      for each of the ρ individuals to send do
         v'_g^i ← φ_s(P)
         send v'_g^i to Q chosen by τ
      end for
   end if{** Asynchronous communication **}
   while individuals are arriving do
      receive v'_g^i from P
      replace an individual chosen from φ_r(Q) by v'_g^i
   end while
end while
```

Fig. 4 Pseudo-code of IBDDE for the sub-population P

rule, which selects the target sub-population Q. The individuals to be migrated are pseudo-randomly (uniformly) chosen by the selection function ϕ_s. Incoming individuals from other sub-populations replace pseudo-randomly chosen local individuals, only if the former are better, by the replacement function ϕ_r.

Figure 4 describes the algorithm as pseudo-code.

In [51], the experiments have been run with a population size S_{pop} equal to 20. The population is divided into two sub-populations of 10 individuals in one experiment, and into four sub-populations of 5 individuals in a second experiment. The migration parameters are set to $\gamma = 100$, $\rho = 1$, the functions ϕ_s and ϕ_r are defined to randomly select an individual, the topology τ is a unidirectional ring very similar to the logical topology used by PDE (see Subsection 3.1). The mutation strategy for DE is DE/rand/1, and the algorithm is tested on 25 different test functions in 30 and 50 dimensions, for a total of 50 test functions.

3.3 Distributed Differential Evolution

In [48], [49], and [50], in order to solve some image registration problems a distributed DE (indicated here with DDE) has been proposed. This algorithm differs from PDE and IBDDE by the topology it uses. Instead of a unidirectional ring, DDE uses a locally connected topology, where each node is connected to μ other nodes. Figure 5 represents such a topology where the nodes are arranged in a mesh folded into a torus.

In t[48], [49], and [50], it has been proposed to set $\mu = 4$, i.e., each node (such as the black disc in the Figure 5) has exactly four nearest neighbors (represented by the four grey discs). In DDE, each node represents one processor running a DE algorithm with a DE/rand/1 mutation strategy on a sub-population. Every M_I generations (the migration interval), each sub-population is allowed to exchange S_I (the migration rate) individuals with its nearest neighbors. In the experimental setup, each node sends a copy of its best individual to its neighbors. Figure 6 describes the algorithm as pseudo-code.

DDE also makes use of a master node, whose role it is to collect the best solutions found in each sub-population and to present these results to the user.

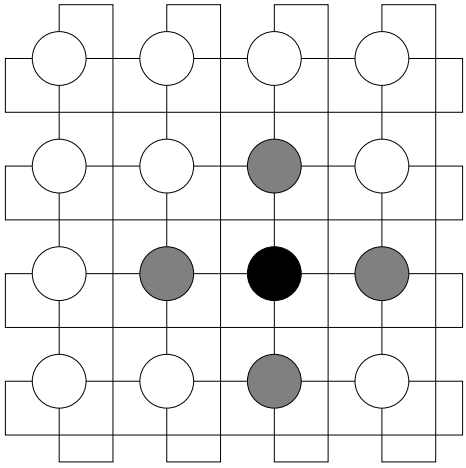


Fig. 5 Torus topology in Distributed Differential Evolution

```
initialize the sub-population
while the stopping condition is not met do
   perform a DE generation
   if the last migration was M_I generations ago then
      send a copy of the best individual to each neighbor
   end if
   if there are incoming individuals then
      replace the worst S_I × μ individuals by the S_I × μ incoming ones
   end if
end while
```

Fig. 6 Pseudo-code of the DDE algorithm at a sub-population

4 Distributed Differential Evolution with Scale Factor Inheritance

The proposed algorithm enhances the PDE structure described in Subsection 3.1 by means of the implementation, in a distributed logic, of the self-adaptive parameter control proposed in [30]. More specifically, an adaptive control of the scale factor "F" is proposed here. The novel mechanism proposed here in this paper is named scale factor inheritance. The proposed algorithm, namely "F" Adaptive Control Parallel Differential Evolution (FACPDE) consists of the following steps.

At the beginning of the optimization process, S_{pop} individuals are pseudo-randomly sampled within the decision space D. These S_{pop} are distributed over m sub-populations; each sub-population is composed of $\frac{S_{pop}}{m}$ individuals. For each (generic h^{th}) sub-population a scale factor F^k, for $k = 1, \ldots, m$, is assigned. Each scale factor is initially generated as pseudo-random by sampling a value from a uniform distribution between -1 and 1. The sub-populations are then arranged according to ring topology, as with the PDE topology represented in Figure 2.

At each generation, each sub-population performs a DE scheme. For each individual x_i of the $\frac{S_{pop}}{m}$, three individuals x_r, x_s and x_t are pseudo-randomly extracted from the population. The provisional offspring x'_{off} is generated by mutation as:

$$x'_{off} = x_t + F^k(x_r - x_s) \tag{4}$$

It is clear that a scale factor taking on a negative value means that the search direction is inverted.

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

$$x_{off,j} = \begin{cases} x_{i,j} & \text{if} \quad rand\,(0,1) < CR \\ x'_{off,j} & otherwise \end{cases} \tag{5}$$

where, as for the sequential DE, $rand\,(0,1)$ is a random number between 0 and 1; j is the index of the gene under examination. The standard one-to-one spawning is then applied and, at the end of each generation, the scheduled replacements are performed.

For each sub-population, between two subsequent generations, a pseudo-random number rand$(0,1)$ is generated by means of a uniform distribution. Analogous to what was explained about the PDE in Subsection 3.1, this pseudo-random number is then compared with a constant value ϕ, namely migration constant. If rand$(0,1) < \phi$, the individual with the best performance is selected to undergo migration. Thus, for the generic k^{th} sub-population, the individual x^k_{best} is duplicated and then replaces a pseudo-randomly selected individual of the neighbor (in the ring) sub-population. The scale factor inheritance mechanism occurs contextually with the migration. More specifically, when the migration occurs, performance of the individual x^k_{best} is compared with that of the best individual belonging to the target sub-population, indicated here with x^{k+1}_{best}. If the new immigrant has a better performance than the best individual of the target sub-population, i.e. if $f\left(x^k_{best}\right) < f\left(x^{k+1}_{best}\right)$, the $(k+1)^{th}$ sub-population inherits the scale factor F^k after a perturbation. More specifically, the scale factor F^{k+1} related to the $(k+1)^{th}$ sub-population is updated according to the following formula:

$$F^{k+1} = F^k + \alpha\mathcal{N}\,(0,1) \tag{6}$$

where $\mathcal{N}\,(0,1)$ is a pseudo-random value sampled from a normal distribution characterized by a zero mean and variance equal to 1. The constant value α has the role of controlling the range of perturbation values $\alpha\mathcal{N}\,(0,1)$. It must be observed that we did not impose any bounds for the variation of F. On the contrary, we decided to allow an unbounded variation of the control parameter and rely on the self-adaptation mechanism.

If the new immigrant does not outperform the best individual of the target sub-population, i.e. $f\left(x^k_{best}\right) \geq f\left(x^{k+1}_{best}\right)$, the scale factor inheritance mechanism is not activated and thus only the migration of the individual x^k_{best} occurs.

The described operations are repeated until the budget conditions are satisfied.

A graphical representation of FACPDE is given in Figure 7.

For the sake of clarity, the pseudo-code illustrating working principles of FACPDE is shown in Figure 8.

4.1 Scale Factor Inheritance: Algorithmic Philosophy

As shown in Section 2, DE is based on a very simple idea, i.e., a search by means of adding vectors and a one-to-one spawning for the survivor selection. Thus, DE is

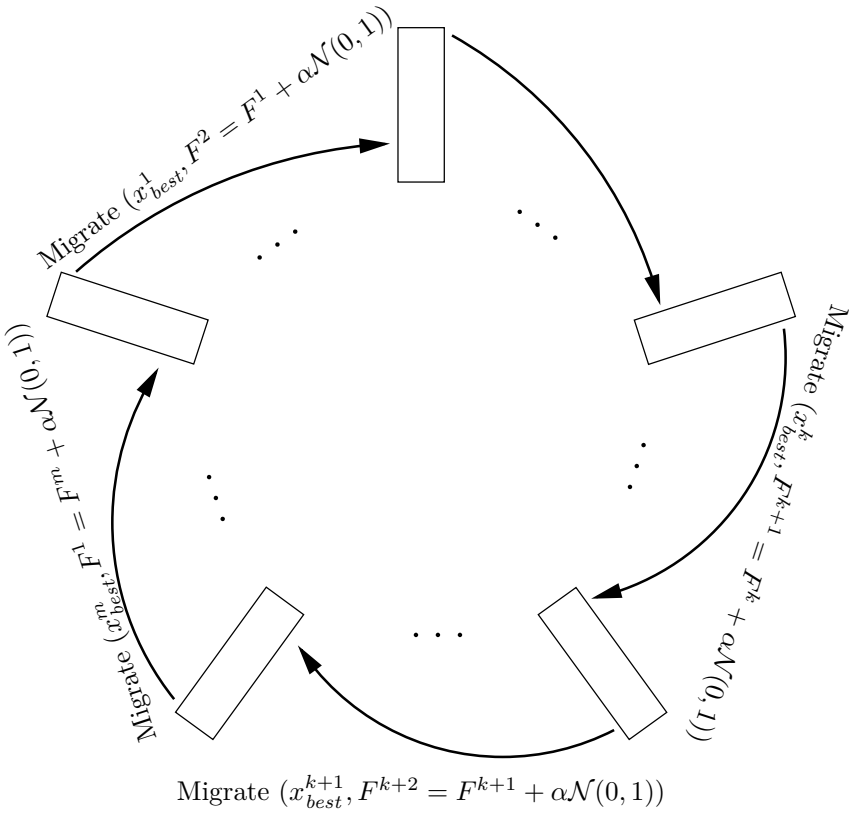


Fig. 7 Graphical Representation of FACPDE

```
while budget conditions do
  for each generation do
    for each sub-population k = 1 : m do
      if rand(0,1) < φ then
        select and copy x^k_best
        migrate x^k_best into the (k+1)^th sub-population by replacing a pseudo-randomly
        selected individual
        if f(x^k_best) < f(x^{k+1}_best) then
          F^{k+1} = F^k + αN(0,1)
        end if
      end if
    end for
  end for
end while
```

Fig. 8 Pseudo-code of the FACPDE algorithm

very simple to implement/code and contains a limited number of parameters to tune (only S_{pop}, F, and CR).

From an algorithmic viewpoint, reasons for the success of DE have been highlighted in [56]: success of DE is due to an implicit self-adaptation contained within the algorithmic structure. More specifically, since, for each candidate solution, the search rule depends on other solutions belonging to the population (e.g. x_t, x_r, and x_s), the capability of detecting new promising offspring solutions depends on the current distribution of the solutions within the decision space. During early stages of the optimization process, solutions tend to be spread out within the decision space. For a given scale factor value, this implies that the mutation appears to generate new solutions by exploring the space by means of a large step size (if x_r and x_s are distant solutions, $F\,(x_r - x_s)$ is a vector characterized by a large modulus). During the optimization process, the solutions of the population tend to concentrate on specific parts of the decision space. Therefore, step size in the mutation is progressively reduced and the search is performed in the neighborhood of the solutions. In other words, due to its structure, a DE scheme is highly explorative at the beginning of the evolution and subsequently becomes more exploitative during optimization.

Although this mechanism seems at first glance to be very efficient, it hides a limitation. If for some reason, the algorithm does not succeed in generating offspring solutions which outperform the corresponding parent, the search is repeated again with similar step size values and will likely fail by falling into an undesired stagnation condition (see [57]). Stagnation is the undesired effect which occurs when a population-based algorithm does not converge to a solution (even suboptimal) and the population diversity is still high. In the case of the DE, stagnation occurs when the algorithm does not manage to improve upon any solution of its population for a prolonged number of generations. In other words, the main drawback of the DE is that the scheme has, for each stage of the optimization process, a limited amount of exploratory moves. If these moves are not enough for generating new promising solutions the search can be heavily compromised.

Thus, in order to enhance the DE performance, alternative search moves should support the original scheme and promote a successful continuation of the optimization process. The use of multiple populations in distributed DE algorithms allows an observation of the decision space from various perspectives and, most importantly, decreases the risk of stagnation since each sub-population imposes a high exploitation pressure. In addition, the migration mechanism ensures that solutions with a high performance are included within the sub-populations during their evolution. This fact is equivalent to modifying the set of search moves. If the migration gives privilege to the best individuals, the new search moves promote the detection of new promising search directions and thus allow the DE search structure to be periodically "refurbished". Thus, migration is supposed to mitigate the risk of DE (sub-)populations stagnating and to enhance the global algorithmic performance.

As a countermeasure against stagnation, several recent DE versions propose a randomization in the search logic which increases the amount of potential exploration moves. For example in [30], scale factor and crossover rates are periodically updated by generating new random values. This simple operation seems to have a very relevant effect on the algorithmic performance. Also the DE scheme proposed in [37] enhances a previously proposed algorithm (presented in [54]) by introducing a randomization of the control parameters. Thus, two operations needed in order to

obtain significant improvements in DE performance seem to be the updating and randomization of control parameters.

In this paper we focus on the scale factor dynamics. Although DE schemes are characterized by the implicit self-adaptation described above, employment of a unique and constant scale factor value can be improper since the exploratory moves depend on distribution of the solution within the decision space. For example, for a highly multi-modal problem, a scale factor $F \approx 1$ can generate very long moving vectors $F(x_r - x_s)$ if the population is spread out within the decision space (see Figure 9(a)) and very short moving vectors if the population is concentrated in some areas of the decision space (see Figure 9(b)). This fact may lead to an excessively explorative behavior during some stages of the evolution and an excessively exploitative behavior during other stages. These two behaviors can cause, respectively, stagnation due to an incapability to detect a promising search direction and a premature convergence due to the excessive exploitation of a suboptimal basin of attraction. In other words, a proper choice of the scale factor depends not only on the optimization problem but also on the stage of the evolution.

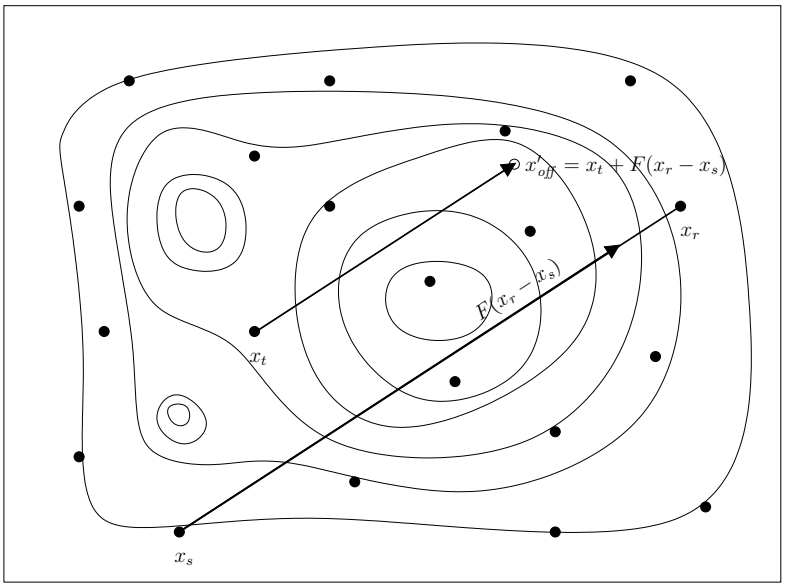
On the basis of this consideration, the scale factor inheritance mechanism has been designed. The initial pseudo-random assignment of the scale factors (one per sub-population) allows each FACPDE sub-population to explore the decision space from complementary perspectives. Subsequently, each evolving sub-population can be seen as a separate searcher which cooperates and competes with the other searcher, analogous to memetic algorithms, see [58], [59], [60], and [61]. The sub-populations cooperate with each other by means of migration of the individual with the best performance which can be seen as a suggestion of a promising search direction. The sub-populations compete with each other by means of the scale factor migration; this fact can be seen like the the imposition of the most successful search strategy to other sub-populations employing a weaker strategy. The ring topology assures propagation of this successful strategy to all sub-populations. On the other hand, this propagation is rather slow, so as to avoid too greedy an algorithmic behavior. In addition, randomization of the scale factor when the migration occurs guarantees a certain diversity of scale factors even in the event that a propagation throughout the entire ring occurs. Thus, we avoid that all the sub-population are characterized by the same scale factor.

In summary, the combined action of an exploration of the decision space from diverse and complementary perspectives, the cooperation mechanism with its suggestion of promising search directions, the competitive procedure of the most successful search strategies and their randomized updates should, together, compose a robust DE based algorithm which efficiently balances its explorative and exploitative resources in order to efficiently and robustly solve complex optimization problems.

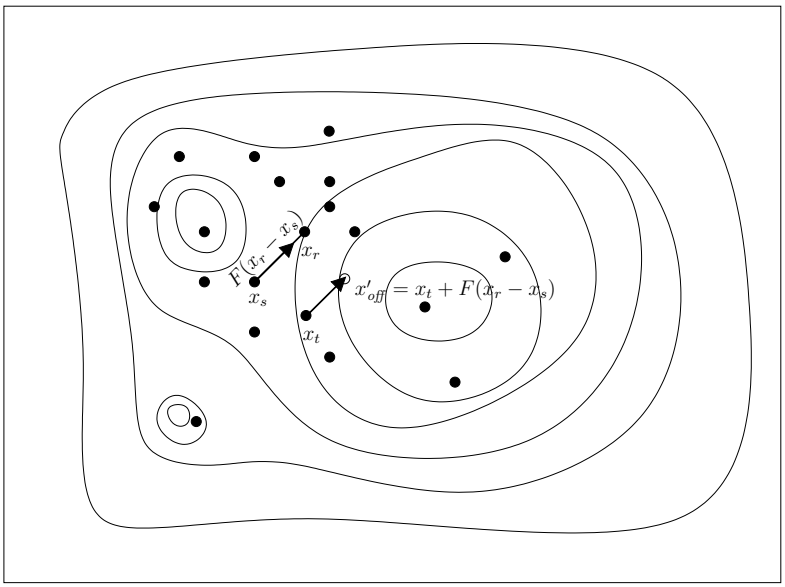
5 Experimental Results

In order to prove the viability of the FACPDE and test its performance with respect to modern distributed DE based algorithms, the following numerical experiments have been performed. The test problems listed in Table 1 have been considered in this study.

The rotated version of some of the test problems listed in Table 1 have been included into the benchmark set. These rotated problems have been generated through



(a) too explorative conditions



(b) too exploitative conditions

Fig. 9 Search mechanism in Differential Evolution

Table 1 Test Problems

Test Problem	Function	Decision Space
Ackley	$-20 + e + 20\exp\left(-\frac{0.2}{n}\sqrt{\sum_{i=1}^{n} x_i^2}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i)x_i\right)$	$[-1,1]^n$
Alpine	$\sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1x_i\rvert$	$[-10,10]^n$
Axis-parallel hyper-ellipsoid	$\sum_{i=1}^{n} ix_i^2$	$[-5.12,5.12]^n$
DeJong	$\lVert x\rVert^2$	$[-5.12,5.12]^n$
DropWave	$-\frac{1+\cos\left(12\sqrt{\lVert x\rVert^2}\right)}{\frac{1}{2}\lVert x\rVert^2+2}$	$[-5.12,5.12]^n$
Griewangk	$\frac{\lVert x\rVert^2}{4000} - \prod_{i=0}^{n}\cos\frac{x_i}{\sqrt{i}} + 1$	$[-600,600]^n$
Michalewicz	$-\sum_{i=1}^{n}\sin x_i\left(\sin\left(\frac{i\cdot x_i^2}{\pi}\right)\right)^{20}$	$[0,\pi]^n$
Pathological	$\sum_{i=1}^{n-1}\left(0.5 + \frac{\sin^2\left(\sqrt{100x_i^2+x_{i+1}^2}-0.5\right)}{1+0.001*\left(x_i^2-2x_ix_{i+1}+x_{i+1}^2\right)^2}\right)$	$[-100,100]^n$
Rastrigin	$10n + \sum_{i=0}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12,5.12]^n$
Rosenbrock valley	$\sum_{i=1}^{n-1}\left(100\left(x_{i+1}-x_i^2\right)^2 + (1-x_i)^2\right)$	$[-2.048,2.048]^n$
Schwefel	$\sum_{i=1}^{n} -x_i\sin\left(\sqrt{\lvert x_i\rvert}\right)$	$[-500,500]^n$
Sum of powers	$\sum_{i=1}^{n}\lvert x_i\rvert^{i+1}$	$[-1,1]^n$
Tirronen	$3\exp\left(-\frac{\lVert x\rVert^2}{10n}\right) - 10\exp\left(-8\lVert x\rVert^2\right)$ $+\frac{2.5}{n}\sum_{i=1}^{n}\cos\left(5\left(x_i + (1+i \bmod 2)cos\left(\lVert x\rVert^2\right)\right)\right)$	$[-10,5]^n$

the multiplication of the vector of variables by a randomly generated orthogonal rotation matrix. In total, twenty-four test problems have been considered in this study with both $n = 500$ and $n = 1000$. The FACPDE has been tested with these twenty-four test problems and its behavior and performance have been compared with those obtained by standard sequential DE/rand/1/bin (simply indicated here as DE), the improved DE algorithm employing Random Scale Factor proposed in [23] and indicated as DERSF, PDE introduced in [45], IBDDE given in [51] and DDE employed in [48], [49], and [50]. It must be remarked that all the algorithms chosen for comparison are modern distributed DE algorithms recently proposed in literature, and which are representative of the state-of-the-art in the field. Each algorithm has been run for $500,000$ fitness evaluations in the case of $n = 500$ and for $1,000,000$ fitness

evaluations when $n = 1000$. Fifty independent runs have been performed for each algorithm involved in this paper.

The algorithms considered in this study have been run with the following parameter setting.

- DE has been run with $F = 0.7$ and $CR = 0.3$ in accordance with the suggestions given in [13] and [62]. The population size has been set to $S_{pop} = 200$ for the 500-dimensional problems and to $S_{pop} = 400$ for the 1000-dimensional ones.
- DESRF has been run with $CR = 0.3$. Analogous to DE, the population size has been set to $S_{pop} = 200$ for the 500-dimensional problems and to $S_{pop} = 400$ for the 1000-dimensional ones. In DERSF the scale factor is randomized, at each generation, according to the rule $F = 0.5\,(1 + rand\,(0, 1))$.
- PDE has been run with populations of 200 or 400 individuals divided into 5 sub-populations of 40 or 80 individuals each, for the 500 and 1000-dimensional problems, respectively. Despite [45] showing better performance for the DE/best/1 mutation strategy in 30 and 50 dimensions, it has proven excessively exploitative and has led to premature convergence of the solutions when used on higher dimension problems. In order to perform a fair comparison, we have carried out an analysis on mutation strategies, leading to the choice of DE/rand/1 and setting the migration constant to $\phi = 0.2$. These settings proved to be the best choices in terms of algorithmic performance and have, thus, been chosen for the experiments described below.
- Similarly to PDE, IBDDE has been run with populations of 200 or 400 individuals divided into 5 sub-populations of 40 or 80 individuals each, depending on the dimensionality of the test problems. The other parameters have been chosen according to the values in [51]: the sub-populations exchange one individual ($\rho = 1$) every 100 generations ($\gamma = 100$). ϕ_s and ϕ_r have been defined so as to pseudo-randomly select an individual by means of a uniform distribution, and τ has been set to a unidirectional ring.
- For the 500-dimensional problems, the DDE has been run with a population of 200 individuals divided into 16 sub-populations of alternatively 12 or 13 individuals. In the case of the 1000-dimensional problems, the population has been set to 400 individuals divided into 16 sub-populations of 25 individuals. Following the suggestions in [49] the sub-populations have been organized into a 4×4 grid folded into a torus ($\mu = 4$). Migration occurred in each sub-population with only its best individual ($S_I = 1$) every $M_I = 5$ generations.
- Similarly to PDE and IBDDE, FACPDE has been run with populations of 200 and 400 individuals divided into 5 sub-populations of 40 and 80 individuals each for the 500 and 1000-dimensional cases, respectively. The migration constant ϕ has been set equal to 0.2 and the constant α in formula (6) has been set equal to 0.1.

It is worthwhile commenting on choice of the population sizes $S_{pop} = 200$ and $S_{pop} = 400$. Although in [9] it is suggested that the DE population size be set equal to about ten times the dimensionality of the problem, this indication is not confirmed by a recent study in [63] where it is shown that a population size lower than the dimensionality of the problem can be optimal in many cases.

Table 2 shows the average of the final results detected by each algorithm $\pm$ the standard deviations, with the 500 dimension case. Table 3 shows the results for the 1000 dimension case. The best results are highlighted in bold face.

Table 2 Average final fitness values ± standard deviations for 500 dimensions problems

	DE	DERSF	PDE	IBDDE	DDE	FACPDE
Ackley	$3.62e+00 \pm 9.69e-03$	$3.58e+00 \pm 1.40e-02$	$1.62e-01 \pm 1.67e-02$	$3.55e+00 \pm 2.96e-02$	$1.51e-01 \pm 6.96e-02$	$\mathbf{2.67e-02 \pm 1.25e-02}$
Alpine	$1.30e+03 \pm 1.19e+01$	$1.24e+03 \pm 1.53e+01$	$8.88e+01 \pm 1.26e+01$	$1.21e+03 \pm 3.43e+01$	$1.50e+02 \pm 4.35e+01$	$\mathbf{3.09e+01 \pm 1.60e+01}$
Ax.-par. hyp.-ell.	$8.67e+05 \pm 1.07e+04$	$7.06e+05 \pm 1.50e+04$	$3.68e+03 \pm 6.84e+02$	$7.27e+05 \pm 3.70e+04$	$4.09e+03 \pm 4.73e+03$	$\mathbf{3.28e+02 \pm 2.96e+02}$
DeJong	$3.86e+03 \pm 5.73e+01$	$3.34e+03 \pm 4.98e+01$	$1.92e+01 \pm 3.57e+00$	$3.19e+03 \pm 2.67e+02$	$1.61e+01 \pm 1.15e+01$	$\mathbf{1.80e+00 \pm 2.02e+00}$
DropWave	$-9.83e-04 \pm 2.23e-05$	$-9.90e-04 \pm 1.91e-05$	$-4.11e-03 \pm 3.96e-04$	$-1.13e-03 \pm 8.03e-05$	$-2.56e-03 \pm 2.69e-04$	$\mathbf{-5.02e-03 \pm 8.78e-04}$
Griewangk	$1.38e+04 \pm 2.15e+02$	$1.19e+04 \pm 1.73e+02$	$5.62e+02 \pm 1.09e+01$	$1.13e+04 \pm 7.65e+02$	$5.68e+02 \pm 4.86e+01$	$\mathbf{5.04e+02 \pm 1.19e+01}$
Michalewicz	$-8.84e+01 \pm 1.61e+00$	$-8.85e+01 \pm 9.05e-01$	$-3.06e+02 \pm 5.68e+00$	$-9.13e+01 \pm 2.09e+00$	$-2.60e+02 \pm 8.56e+00$	$\mathbf{-3.54e+02 \pm 3.12e+01}$
Pathological	$-1.05e+02 \pm 1.96e+00$	$-1.21e+02 \pm 2.40e+00$	$-3.34e+02 \pm 5.98e+00$	$-3.34e+02 \pm 7.40e+00$	$-3.06e+02 \pm 7.28e+00$	$\mathbf{-3.55e+02 \pm 3.05e+01}$
Rastrigin	$8.21e+03 \pm 5.26e+01$	$7.90e+03 \pm 6.55e+01$	$1.91e+03 \pm 9.94e+01$	$7.78e+03 \pm 1.55e+02$	$2.73e+03 \pm 2.28e+02$	$\mathbf{9.91e+02 \pm 3.93e+02}$
Rosenbrock	$1.94e+05 \pm 5.44e+03$	$1.79e+05 \pm 4.47e+03$	$2.11e+03 \pm 1.77e+02$	$1.35e+05 \pm 1.61e+04$	$1.82e+03 \pm 6.14e+02$	$\mathbf{1.63e+03 \pm 6.18e+02}$
Schwefel	$-4.42e+04 \pm 7.07e+02$	$-4.57e+04 \pm 9.54e+02$	$-1.30e+05 \pm 3.17e+03$	$-4.48e+04 \pm 8.15e+02$	$-1.06e+05 \pm 4.19e+03$	$\mathbf{-1.54e+05 \pm 1.83e+04}$
Sum of powers	$1.82e+00 \pm 3.99e-01$	$1.82e+00 \pm 3.99e-01$	$1.06e-05 \pm 5.19e-05$	$3.09e-01 \pm 1.67e-01$	$1.02e-03 \pm 2.51e-03$	$\mathbf{4.82e-06 \pm 1.15e-05}$
Tirronen	$-9.92e-01 \pm 2.02e-02$	$-1.10e+00 \pm 1.77e-02$	$-1.57e+00 \pm 3.44e-02$	$-1.01e+00 \pm 1.90e-02$	$-1.38e+00 \pm 6.76e-02$	$\mathbf{-1.63e+00 \pm 3.41e-02}$
Rt. Ackley	$3.65e+00 \pm 1.54e-02$	$3.56e+00 \pm 1.99e-02$	$2.15e-01 \pm 2.50e-02$	$3.53e+00 \pm 3.82e-02$	$1.94e-01 \pm 7.21e-02$	$\mathbf{6.92e-02 \pm 1.92e-02}$
Rt. Alpine	$1.32e+03 \pm 1.60e+01$	$1.28e+03 \pm 1.54e+01$	$1.03e+02 \pm 9.74e+00$	$1.23e+03 \pm 4.13e+01$	$1.39e+02 \pm 2.64e+01$	$\mathbf{6.26e+01 \pm 2.87e+01}$
Rt. Ax.-par. hyp.-ell.	$8.41e+05 \pm 1.53e+04$	$7.43e+05 \pm 1.43e+04$	$4.90e+03 \pm 7.92e+02$	$7.12e+05 \pm 3.97e+04$	$3.87e+03 \pm 2.20e+03$	$\mathbf{1.07e+03 \pm 4.10e+02}$
Rt. Griewangk	$1.28e+04 \pm 2.16e+02$	$1.12e+04 \pm 2.05e+02$	$5.66e+02 \pm 1.03e+01$	$1.10e+04 \pm 6.18e+02$	$5.56e+02 \pm 3.75e+01$	$\mathbf{5.10e+02 \pm 5.04e+00}$
Rt. Michalewicz	$-4.93e+01 \pm 1.90e+00$	$-4.82e+01 \pm 1.18e+00$	$-1.76e+02 \pm 7.76e+00$	$-4.99e+01 \pm 1.62e+00$	$-1.37e+02 \pm 7.98e+00$	$\mathbf{-1.92e+02 \pm 1.20e+01}$
Rt. Pathological	$-2.11e+01 \pm 1.36e+00$	$-2.09e+01 \pm 7.80e-01$	$\mathbf{-1.21e+02 \pm 7.30e+00}$	$-2.11e+01 \pm 1.05e+00$	$-1.01e+02 \pm 1.20e+01$	$-1.13e+02 \pm 1.26e+01$
Rt. Rastrigin	$8.39e+03 \pm 6.89e+01$	$8.11e+03 \pm 5.68e+01$	$1.95e+03 \pm 1.51e+02$	$7.85e+03 \pm 2.27e+02$	$2.70e+03 \pm 2.60e+02$	$\mathbf{1.10e+03 \pm 1.85e+02}$
Rt. Rosenbrock	$1.95e+05 \pm 6.04e+03$	$1.75e+05 \pm 4.47e+03$	$1.66e+03 \pm 1.53e+02$	$1.47e+05 \pm 1.56e+04$	$1.45e+03 \pm 4.68e+02$	$\mathbf{9.74e+02 \pm 1.74e+02}$
Rt. Schwefel	$-5.16e+04 \pm 1.80e+03$	$-5.45e+04 \pm 1.35e+03$	$-1.65e+05 \pm 4.74e+03$	$-5.31e+04 \pm 1.87e+03$	$-1.27e+05 \pm 7.80e+03$	$\mathbf{-1.67e+05 \pm 1.22e+04}$
Rt. Sum of powers	$3.81e+16 \pm 1.28e+17$	$8.50e+16 \pm 4.18e+17$	$\mathbf{1.06e-05 \pm 5.20e-05}$	$2.13e+15 \pm 7.15e+15$	$2.27e-03 \pm 6.51e-03$	$2.47e-04 \pm 1.71e-03$
Rt. Tirronen	$-5.06e-01 \pm 2.09e-02$	$-5.02e-01 \pm 2.34e-02$	$-1.24e+00 \pm 7.73e-02$	$-5.08e-01 \pm 2.45e-02$	$-9.31e-01 \pm 9.41e-02$	$\mathbf{-1.28e+00 \pm 7.60e-02}$

Table 3 Average final fitness values ± standard deviations for 1000 dimensions problems

	DE	DERSF	PDE	IBDDE	DDE	FACPDE
Ackley	$3.75e+00 \pm 7.55e-03$	$3.72e+00 \pm 7.94e-03$	$9.75e-01 \pm 4.77e-02$	$3.72e+00 \pm 1.61e-02$	$7.25e-01 \pm 8.17e-02$	$\mathbf{2.00e-02 \pm 1.02e-02}$
Alpine	$2.77e+03 \pm 2.71e+01$	$2.73e+03 \pm 1.66e+01$	$6.24e+02 \pm 3.28e+01$	$2.66e+03 \pm 6.16e+01$	$3.85e+02 \pm 4.99e+01$	$\mathbf{1.05e+02 \pm 2.33e+02}$
Ax.-par. hyp.-ell.	$3.96e+06 \pm 4.94e+04$	$3.89e+06 \pm 4.28e+04$	$1.81e+05 \pm 1.23e+04$	$3.20e+06 \pm 1.62e+05$	$1.06e+05 \pm 1.86e+04$	$\mathbf{6.43e+02 \pm 9.09e+02}$
DeJong	$8.01e+03 \pm 9.20e+01$	$8.00e+03 \pm 9.03e+01$	$4.66e+02 \pm 2.81e+01$	$6.29e+03 \pm 3.74e+02$	$2.83e+02 \pm 5.12e+01$	$\mathbf{1.75e+00 \pm 1.94e+00}$
DropWave	$-4.84e-04 \pm 7.42e-06$	$-4.84e-04 \pm 7.42e-06$	$-1.62e-03 \pm 9.29e-05$	$-5.20e-04 \pm 2.21e-05$	$-1.17e-03 \pm 9.73e-05$	$\mathbf{-2.06e-03 \pm 2.47e-04}$
Griewangk	$2.85e+04 \pm 3.16e+02$	$2.84e+04 \pm 2.72e+02$	$2.58e+03 \pm 1.03e+02$	$2.28e+04 \pm 1.55e+03$	$1.96e+03 \pm 1.80e+02$	$\mathbf{1.02e+03 \pm 2.12e+02}$
Michalewicz	$-1.45e+02 \pm 1.72e+00$	$-1.49e+02 \pm 1.72e+00$	$-4.18e+02 \pm 9.73e+00$	$-1.49e+02 \pm 2.24e+00$	$-4.43e+02 \pm 1.22e+01$	$\mathbf{-6.03e+02 \pm 1.57e+02}$
Pathological	$-1.57e+02 \pm 2.41e+00$	$-1.98e+02 \pm 3.95e+00$	$-6.39e+02 \pm 8.43e+00$	$-1.87e+02 \pm 3.77e+01$	$-5.70e+02 \pm 1.30e+01$	$\mathbf{-7.17e+02 \pm 6.26e+00}$
Rastrigin	$1.75e+04 \pm 9.32e+01$	$1.71e+04 \pm 7.23e+01$	$6.65e+03 \pm 2.16e+02$	$1.69e+04 \pm 3.40e+02$	$6.18e+03 \pm 3.96e+02$	$\mathbf{1.99e+03 \pm 1.35e+03}$
Rosenbrock	$4.06e+05 \pm 7.22e+03$	$4.06e+05 \pm 7.19e+03$	$1.34e+04 \pm 9.17e+02$	$2.90e+05 \pm 3.35e+04$	$8.96e+03 \pm 1.82e+03$	$\mathbf{2.94e+03 \pm 6.19e+02}$
Schwefel	$-6.43e+04 \pm 1.08e+03$	$-6.51e+04 \pm 9.96e+02$	$-1.91e+05 \pm 4.04e+03$	$-6.45e+04 \pm 1.08e+03$	$-1.86e+05 \pm 8.61e+03$	$\mathbf{-3.24e+05 \pm 5.25e+04}$
Sum of powers	$1.90e+00 \pm 3.55e-01$	$1.90e+00 \pm 3.55e-01$	$\mathbf{3.07e-06 \pm 1.01e-05}$	$1.07e+00 \pm 3.69e-01$	$4.68e-04 \pm 1.64e-03$	$3.50e-06 \pm 1.59e-05$
Tirronen	$-8.49e-01 \pm 1.27e-02$	$-9.72e-01 \pm 1.46e-02$	$-1.45e+00 \pm 2.04e-02$	$-8.63e-01 \pm 1.40e-02$	$-1.35e+00 \pm 5.11e-02$	$\mathbf{-1.59e+00 \pm 3.57e-02}$
Rt. Ackley	$3.78e+00 \pm 1.04e-02$	$3.74e+00 \pm 1.21e-02$	$9.35e-01 \pm 5.37e-02$	$3.69e+00 \pm 4.53e-02$	$6.90e-01 \pm 7.83e-02$	$\mathbf{6.45e-02 \pm 1.91e-02}$
Rt. Alpine	$2.83e+03 \pm 2.14e+01$	$2.79e+03 \pm 2.22e+01$	$6.13e+02 \pm 2.58e+01$	$2.70e+03 \pm 8.77e+01$	$4.01e+02 \pm 4.19e+01$	$\mathbf{1.13e+02 \pm 3.59e+01}$
Rt. Ax.-par. hyp.-ell.	$3.87e+06 \pm 5.89e+04$	$3.76e+06 \pm 4.55e+04$	$1.59e+05 \pm 1.25e+04$	$3.31e+06 \pm 2.20e+05$	$8.52e+04 \pm 1.60e+04$	$\mathbf{3.31e+03 \pm 1.24e+03}$
Rt. Griewangk	$2.80e+04 \pm 3.56e+02$	$2.76e+04 \pm 2.48e+02$	$2.24e+03 \pm 9.40e+01$	$2.33e+04 \pm 1.52e+03$	$1.75e+03 \pm 1.60e+02$	$\mathbf{1.01e+03 \pm 5.59e+00}$
Rt. Michalewicz	$-7.20e+01 \pm 2.82e+00$	$-7.51e+01 \pm 1.83e+00$	$-2.45e+02 \pm 7.54e+00$	$-7.24e+01 \pm 2.19e+00$	$-2.29e+02 \pm 1.23e+01$	$\mathbf{-3.08e+02 \pm 5.12e+01}$
Rt. Pathological	$-2.95e+01 \pm 1.45e+00$	$-2.98e+01 \pm 9.77e-01$	$-1.45e+02 \pm 6.58e+00$	$-2.99e+01 \pm 1.56e+00$	$\mathbf{-1.80e+02 \pm 1.89e+01}$	$-1.60e+02 \pm 2.12e+01$
Rt. Rastrigin	$1.76e+04 \pm 1.42e+02$	$1.74e+04 \pm 1.06e+02$	$6.71e+03 \pm 2.04e+02$	$1.66e+04 \pm 5.33e+02$	$5.90e+03 \pm 4.07e+02$	$\mathbf{2.18e+03 \pm 6.85e+02}$
Rt. Rosenbrock	$4.32e+05 \pm 9.18e+03$	$4.28e+05 \pm 8.25e+03$	$1.08e+04 \pm 7.14e+02$	$3.18e+05 \pm 3.03e+04$	$6.95e+03 \pm 9.68e+02$	$\mathbf{2.08e+03 \pm 4.17e+02}$
Rt. Schwefel	$-7.09e+04 \pm 2.21e+03$	$-7.24e+04 \pm 1.97e+03$	$-2.48e+05 \pm 7.65e+03$	$-7.18e+04 \pm 1.83e+03$	$-2.48e+05 \pm 1.06e+04$	$\mathbf{-3.16e+05 \pm 2.25e+04}$
Rt. Sum of powers	$1.14e+52 \pm 4.68e+52$	$1.12e+54 \pm 7.06e+54$	$\mathbf{1.00e-07 \pm 4.88e-07}$	$8.67e+50 \pm 4.47e+51$	$2.02e-02 \pm 8.35e-02$	$2.43e-01 \pm 1.70e+00$
Rt. Tirronen	$-3.73e-01 \pm 1.49e-02$	$-3.98e-01 \pm 1.24e-02$	$-9.73e-01 \pm 4.86e-02$	$-3.74e-01 \pm 1.89e-02$	$-8.12e-01 \pm 6.75e-02$	$\mathbf{-1.04e+00 \pm 6.09e-02}$

Results in Tables 2 and 3 show that the sequential DE algorithms are outperformed by the distributed algorithms. This result confirms that a structured population can enhance performance of the DE, see [64]. In addition, FACPDE seems to have a very good performance with the test problems considered in this study since it detects those solutions with best performance for twenty-three and twenty-one test problems out of the twenty-four considered in 500 and 1000 dimensions, respectively. It must be remarked that in the cases where FACPDE does not detect the solutions with best performance, the algorithm still detects, in any case, competitive solutions; these solutions usually have, on average, the second best performance. In this sense, FACPDE seems to be a high quality algorithm for various test problems.

In order to prove statistical significance of the results, the Wilcoxon Rank-Sum test has also been applied according to the description given in [65], where the confidence level has been fixed to 0.95. Tables 4 and 5 show results of the test. A "+" indicates the case in which FACPDE statistically outperforms, for the corresponding test problem, the algorithm mentioned in that column; a "=" indicates that a pairwise comparison leads to success of the Wilcoxon Rank-Sum test, i.e., the two algorithms have the same performance; a "-" indicates that FACPDE is outperformed.

In the case of 500-dimensional problems, the Wilcoxon test results in Table 4 show that FACPDE, out of the one hundred twenty pair-wise comparisons performed, loses out in only one case, obtains the same results in four cases, and wins in ninety-one cases. Thus, FACPDE comes out behind in only 0.83% of the comparisons and and comes out ahead in 95.0% of the considered comparisons. In 1000 dimensions, the Wilcoxon test results displayed in Table 5 show that FACPDE, out of the one hundred twenty pair-wise comparisons performed, loses in one case, obtains the same results in seven cases, and wins in one hundred twelve cases. Thus, FACPDE loses only 0.83% of the comparisons and and is shown to be superior in 93.3% of the considered comparisons. The statistical test carried out confirms that FACPDE has a very good performance in regard to the studied test problems. In addition, the comparison between PDE and FACPDE shows that the proposed scale factor inheritance mechanism seems to be very beneficial.

In order to strengthen the statistical significance of the results, the Holm procedure [66] has been applied by following the description in [67]. The Holm procedure consists of the following. Considering the results in tables 2 and 3, the six algorithms under analysis have been ranked on the basis of their average performance calculated over the twenty-four test problems. Thus, or in other words, a score R_i for $i = 1, \ldots, N_A$ (where N_A is the number of algorithms under analysis, $N_A = 6$ in our case) has been assigned. With the calculated R_i values, the FACPDE has been taken as a reference algorithm. Indicating with R_0 the rank of FACPDE, and with R_j for $j = 1, \ldots, N_A - 1$ the rank of one of the remaining five algorithms, the values z_j have been calculated as

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

where N_{TP} is the number of test problems in consideration ($N_{TP} = 24$ 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 $\delta/(N_A - j)$ where δ is the level of confidence, set to 0.05 in our case. Tables 6 and 7 display z_j values, p_j values, and corresponding $\delta/(N_A - j)$. The values of z_j and p_j are expressed in terms of z_{N_A-j} and p_{N_A-j} for $j = 1, \ldots, N_A -$

Table 4 Results of the Wilcoxon rank-sum test for 500 dimensions problems

	DE	DERSF	PDE	IBDDE	DDE
Ackley	+	+	+	+	+
Alpine	+	+	+	+	+
Ax.-par. hyp.-ell.	+	+	+	+	+
DeJong	+	+	+	+	+
DropWave	+	+	+	+	+
Griewangk	+	+	+	+	+
Michalewicz	+	+	+	+	+
Pathological	+	+	+	+	+
Rastrigin	+	+	+	+	+
Rosenbrock	+	+	+	+	=
Schwefel	+	+	+	+	+
Sum of powers	+	+	=	+	+
Tirronen	+	+	+	+	+
Rt. Ackley	+	+	+	+	+
Rt. Alpine	+	+	+	+	+
Rt. Ax.-par. hyp.-ell.	+	+	+	+	+
Rt. Griewangk	+	+	+	+	+
Rt. Michalewicz	+	+	+	+	+
Rt. Pathological	+	+	-	+	+
Rt. Rastrigin	+	+	+	+	+
Rt. Rosenbrock	+	+	+	+	+
Rt. Schwefel	+	+	=	+	+
Rt. Sum of powers	+	=	=	+	+
Rt. Tirronen	+	+	+	+	+

1. Moreover, it is indicated whether the null-hypothesis (that the two algorithms have indistinguishable performances) is "Rejected" i.e., the FACPDE statistically outperforms the algorithm under consideration, or "Accepted" if the distribution of values can be considered the same (there is no outperformance).

The Holm procedure confirms that the FACPDE displays a significantly better performance with respect to the other algorithms in this study for both 500- and 1000-dimensional cases.

In order to carry out a numerical comparison of the convergence speed performance, for each test problem, the average final fitness value returned by the best performing algorithm G has been considered. Subsequently, the average fitness value at the beginning of the optimization process J has also been computed. The threshold value $THR = J - 0.95(J - G)$ has then been calculated. The value THR represents 95% of the decay in the fitness value of the algorithm with the best performance. If an algorithm succeeds during a certain run to reach the value THR, the run is said to be successful. For each test problem, the average amount of fitness evaluations $\bar{ne}$ required, for each algorithm, to reach THR has been computed. Subsequently, the Q-test (Q stands for Quality) described in [56] has been applied. For each test problem and each algorithm, the Q measure is computed as:

$$Q = \frac{\bar{ne}}{Rob} \tag{7}$$

where the robustness Rob is the percentage of successful runs. It is clear that, for each test problem, the smallest value equals the best performance in terms of convergence

Table 5 Results of the Wilcoxon rank-sum test for 1000 dimensions problems

	DE	DERSF	PDE	IBDDE	DDE
Ackley	+	+	+	+	+
Alpine	+	+	+	+	+
Ax.-par. hyp.-ell.	+	+	+	+	+
DeJong	+	+	+	+	+
DropWave	+	+	+	+	+
Griewangk	+	+	+	+	+
Michalewicz	+	+	+	+	+
Pathological	+	+	+	+	+
Rastrigin	+	+	+	+	+
Rosenbrock	+	+	+	+	+
Schwefel	+	+	+	+	+
Sum of powers	+	+	=	+	=
Tirronen	+	+	+	+	+
Rt. Ackley	+	+	+	+	+
Rt. Alpine	+	+	+	+	+
Rt. Ax.-par. hyp.-ell.	+	+	+	+	+
Rt. Griewangk	+	+	+	+	+
Rt. Michalewicz	+	+	+	+	+
Rt. Pathological	+	+	+	+	-
Rt. Rastrigin	+	+	+	+	+
Rt. Rosenbrock	+	+	+	+	+
Rt. Schwefel	+	+	+	+	+
Rt. Sum of powers	=	=	=	=	=
Rt. Tirronen	+	+	+	+	+

Table 6 Results of the Holm procedure for 500-dimensional problems (FACPDE is the control algorithm)

$N_A - j$	Optimizer	z_{N_A-j}	p_{N_A-j}	$\delta/(N_A - j)$	Hypothesis
5	DE	-8.72e+00	1.41e-18	1.00e-02	Rejected
4	DERSF	-7.33e+00	1.16e-13	1.25e-02	Rejected
3	IBDDE	-5.63e+00	8.90e-09	1.67e-02	Rejected
2	DDE	-3.09e+00	1.01e-03	2.50e-02	Rejected
1	PDE	-2.08e+00	1.86e-02	5.00e-02	Rejected

Table 7 Results of the Holm procedure for 1000-dimensional problems (FACPDE is the control algorithm)

$N_A - j$	Optimizer	z_{N_A-j}	p_{N_A-j}	$\delta/(N_A - j)$	Hypothesis
5	DE	-8.72e+00	1.41e-18	1.00e-02	Rejected
4	DERSF	-6.71e+00	9.59e-12	1.25e-02	Rejected
3	IBDDE	-5.86e+00	2.27e-09	1.67e-02	Rejected
2	PDE	-2.62e+00	4.36e-03	2.50e-02	Rejected
1	DDE	-2.01e+00	2.24e-02	5.00e-02	Rejected

Table 8 Results of the Q-test for 500 dimensions problems

	THR	DE	DERSF	PDE	IBDDE	DDE	FACPDE
Ackley	2.15e-01	∞	∞	4.58e+03	∞	4.65e+03	**7.37e+02**
Alpine	9.89e+01	∞	∞	5.82e+03	∞	2.54e+04	**1.79e+03**
Ax.-par. hyp.-ell.	4.99e+04	∞	∞	2.14e+03	∞	1.20e+03	**4.43e+02**
DeJong	2.02e+02	∞	∞	2.36e+03	∞	1.35e+03	**4.96e+02**
DropWave	-4.81e-03	∞	∞	9.31e+04	∞	∞	**3.50e+03**
Griewangk	1.19e+03	∞	∞	2.34e+03	∞	1.40e+03	**5.29e+02**
Michalewicz	-3.39e+02	∞	∞	∞	∞	∞	**3.38e+03**
Pathological	-3.37e+02	∞	∞	8.43e+03	1.65e+04	∞	**1.22e+03**
Rastrigin	1.38e+03	∞	∞	∞	∞	∞	**1.23e+03**
Rosenbrock	1.17e+04	∞	∞	1.57e+03	∞	7.23e+02	**4.71e+02**
Schwefel	-1.46e+05	∞	∞	∞	∞	∞	**1.17e+03**
Sum of powers	1.24e-01	∞	∞	2.32e+02	3.52e+04	**7.97e+01**	1.81e+02
Tirronen	-1.55e+00	∞	∞	5.04e+03	∞	∞	**2.04e+03**
Rt. Ackley	2.55e-01	∞	∞	4.98e+03	∞	4.59e+03	**8.62e+02**
Rt. Alpine	1.31e+02	∞	∞	4.35e+03	∞	8.47e+03	**1.50e+03**
Rt. Ax.-par. hyp.-ell.	4.97e+04	∞	∞	2.04e+03	∞	1.04e+03	**4.84e+02**
Rt. Griewangk	1.19e+03	∞	∞	2.17e+03	∞	1.18e+03	**4.91e+02**
Rt. Michalewicz	-1.83e+02	∞	∞	2.27e+04	∞	∞	**3.87e+03**
Rt. Pathological	-1.15e+02	∞	∞	**5.24e+03**	∞	3.16e+04	1.01e+04
Rt. Rastrigin	1.49e+03	∞	∞	∞	∞	∞	**1.76e+03**
Rt. Rosenbrock	1.17e+04	∞	∞	1.41e+03	∞	6.18e+02	**4.70e+02**
Rt. Schwefel	-1.60e+05	∞	∞	4.70e+03	∞	∞	**3.39e+03**
Rt. Sum of powers	6.33e+20	2.27e+01	1.31e+01	5.24e+00	1.44e+01	4.92e+00	**4.12e+00**
Rt. Tirronen	-1.21e+00	∞	∞	5.21e+03	∞	∞	**3.72e+03**

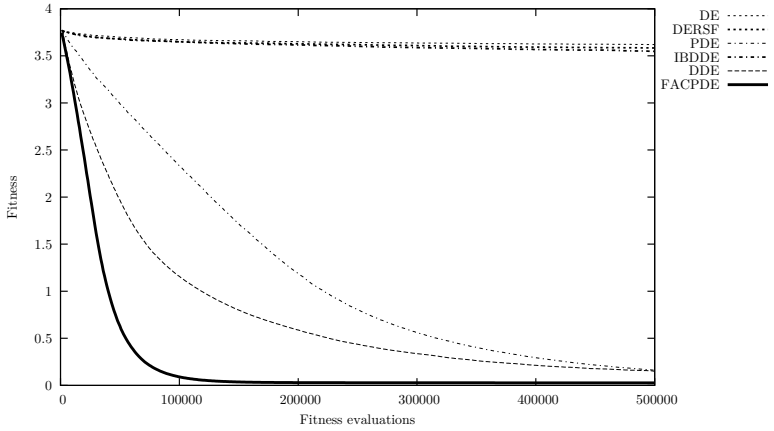
speed. The value "∞" means that $Rob = 0$, i.e., the algorithm never reached the THR.

Tables 8 and 9 show the Q values for 500-dimensional problems and 1000-dimensional problems respectively, as well as the associated THR values. The best results are highlighted in bold face.

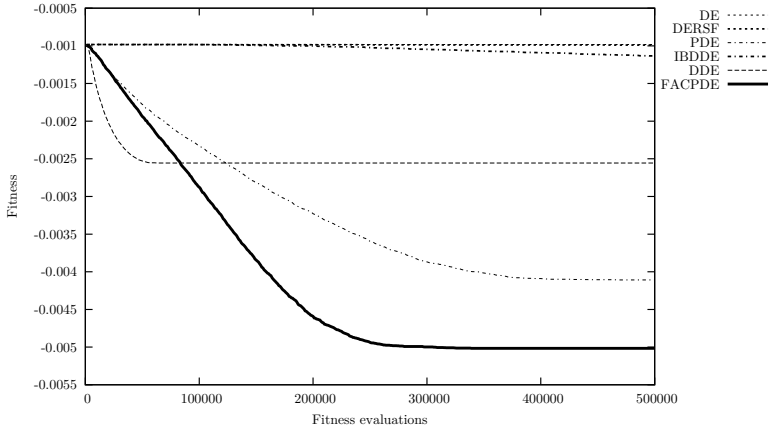
The Q-test results, listed in Table 8 and 9, show that in 500 and 1000-dimensional cases, the proposed FACPDE algorithm has the best performance in terms of convergence speed in twenty-two and twenty-one cases out of the twenty-four considered, respectively. Most importantly, the FACPDE algorithm, throughout all considered test problems, is never characterized by an ∞ value of Q-measure. This fact shows that the proposed algorithm is always competitive with the other algorithms in the benchmark and is never relevantly outperformed. In summary, the algorithmic behavior of FACPDE, thanks to its scale factor inheritance mechanism, is extremely promising in terms of algorithmic robustness.

Figure 9 shows average performance trends of the five considered algorithms over a selection of the test problems listed in Table 1 in 500 dimensions.

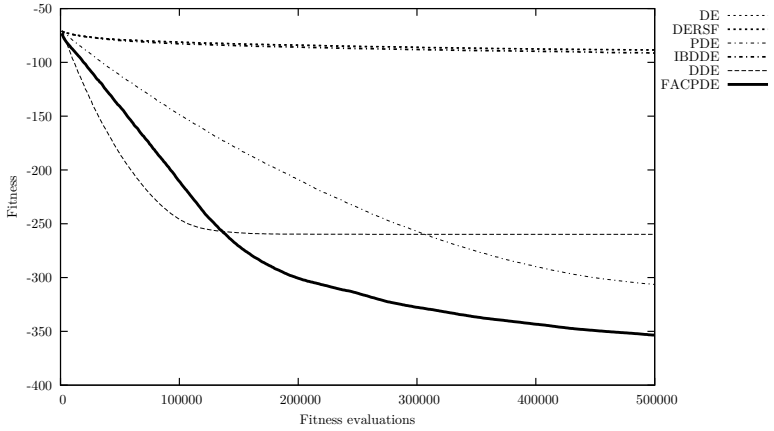
Figures 9(c)–9(k) show that in 500 dimensions, the serial DE algorithms improve only marginally. IBDDE has a slightly better performance than DE and DERSF (see for example Figure 9(i)), but is still not competitive compared to PDE, DDE and FACPDE for the high dimensional problems considered in this study. In Figures 9(d)–9(k), DDE improves its solutions very quickly in the beginning, before ceasing to make any significant improvement. The transition occurs around 100,000



(c) Ackley

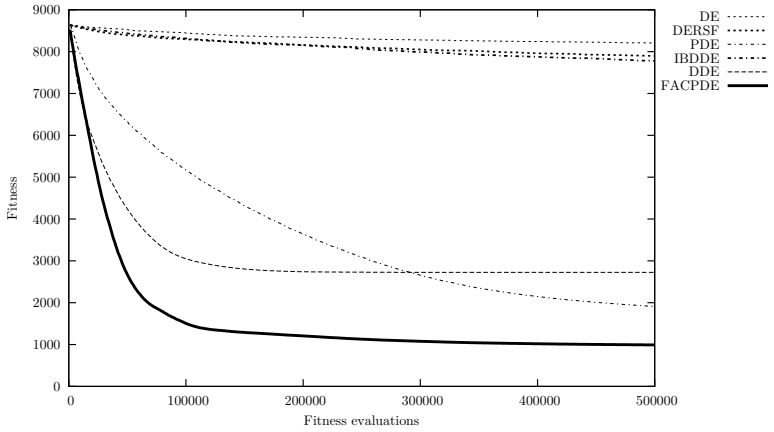


(d) DropWave

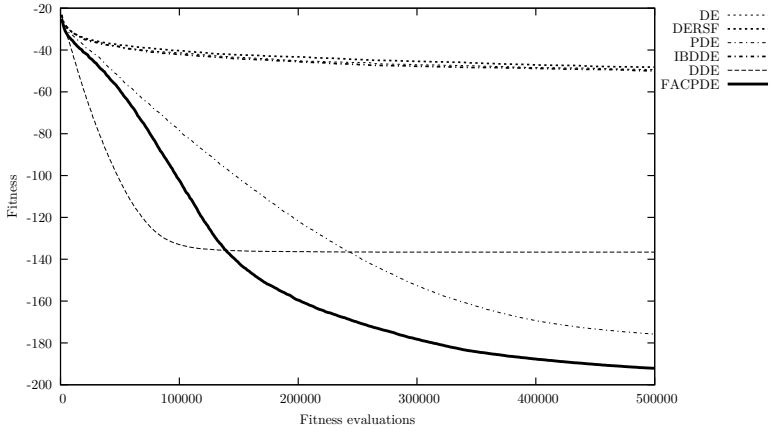


(e) Michalewicz

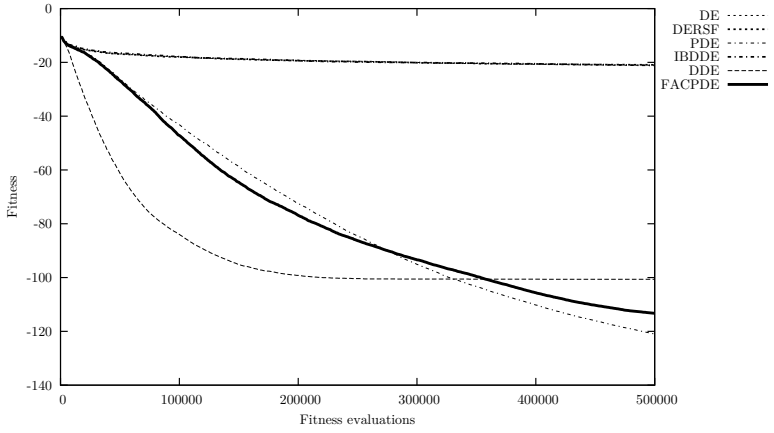
Fig. 9 Performance trends in 500 dimensions



(f) Rastrigin

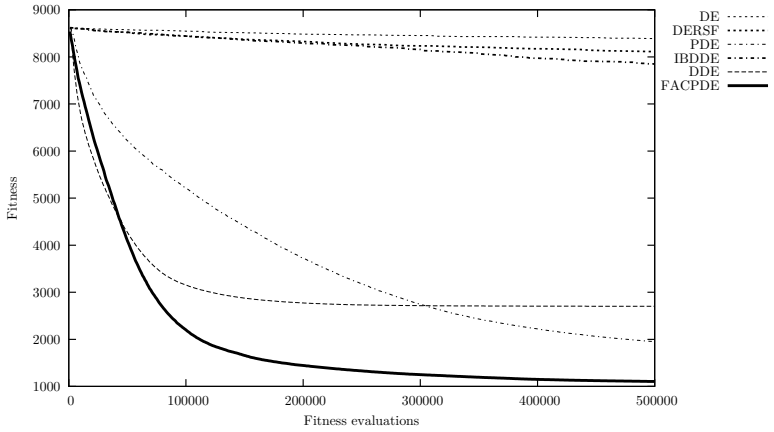


(g) Rotated Michalewicz

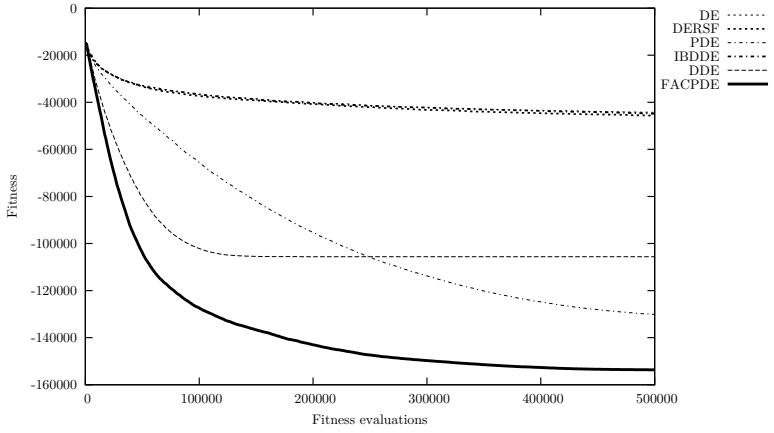


(h) Rotated Pathological

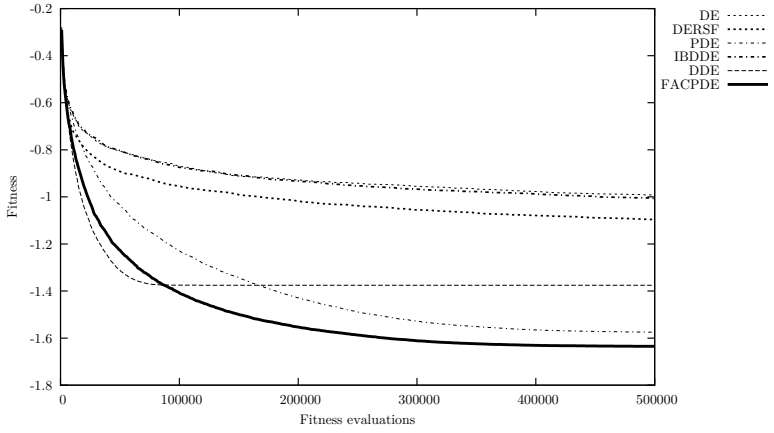
Fig. 9 Performance trends in 500 dimensions (continued)



(i) Rotated Rastrigin



(j) Schwfel



(k) Tirronen

Fig. 9 Performance trends in 500 dimensions (continued)

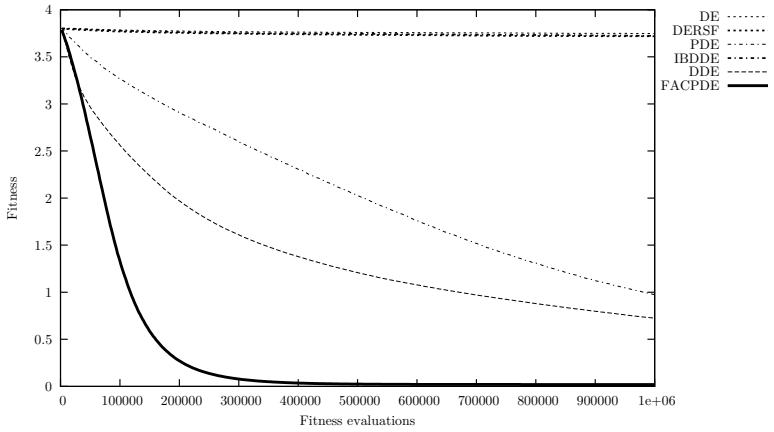
Table 9 Results of the Q-test for 1000 dimensions problems

	THR	DE	DERSF	PDE	IBDDE	DDE	FACPDE
Ackley	2.10e-01	∞	∞	∞	∞	∞	**2.14e+03**
Alpine	2.41e+02	∞	∞	∞	∞	∞	**2.81e+03**
Ax.-par. hyp.-ell.	2.02e+05	∞	∞	9.45e+03	∞	6.10e+03	**1.43e+03**
DeJong	4.11e+02	∞	∞	∞	∞	7.61e+03	**1.40e+03**
DropWave	-1.98e-03	∞	∞	∞	∞	∞	**1.13e+04**
Griewangk	2.43e+03	∞	∞	1.22e+05	∞	7.43e+03	**1.45e+03**
Michalewicz	-5.79e+02	∞	∞	∞	∞	∞	**7.94e+03**
Pathological	-6.81e+02	∞	∞	∞	∞	∞	**4.39e+03**
Rastrigin	2.78e+03	∞	∞	∞	∞	∞	**2.59e+03**
Rosenbrock	2.38e+04	∞	∞	6.57e+03	∞	3.37e+03	**1.54e+03**
Schwefel	-3.09e+05	∞	∞	∞	∞	∞	**2.57e+03**
Sum of powers	1.23e-01	∞	∞	6.84e+02	∞	**1.90e+02**	4.86e+02
Tirronen	-1.51e+00	∞	∞	∞	∞	∞	**5.03e+03**
Rt. Ackley	2.52e-01	∞	∞	∞	∞	∞	**2.60e+03**
Rt. Alpine	2.52e+02	∞	∞	∞	∞	∞	**3.68e+03**
Rt. Ax.-par. hyp.-ell.	2.02e+05	∞	∞	8.78e+03	∞	5.15e+03	**1.49e+03**
Rt. Griewangk	2.40e+03	∞	∞	1.01e+04	∞	6.09e+03	**1.37e+03**
Rt. Michalewicz	-2.94e+02	∞	∞	∞	∞	∞	**8.87e+03**
Rt. Pathological	-1.72e+02	∞	∞	∞	∞	**7.56e+03**	3.30e+04
Rt. Rastrigin	2.96e+03	∞	∞	∞	∞	∞	**4.03e+03**
Rt. Rosenbrock	2.43e+04	∞	∞	5.65e+03	∞	2.52e+03	**1.30e+03**
Rt. Schwefel	-3.01e+05	∞	∞	∞	∞	∞	**6.91e+03**
Rt. Sum of powers	2.08e+56	**8.00e+00**	1.30e+01	8.00e+00	8.00e+00	8.00e+00	9.28e+00
Rt. Tirronen	-9.88e-01	∞	∞	2.10e+04	∞	∞	**9.92e+03**

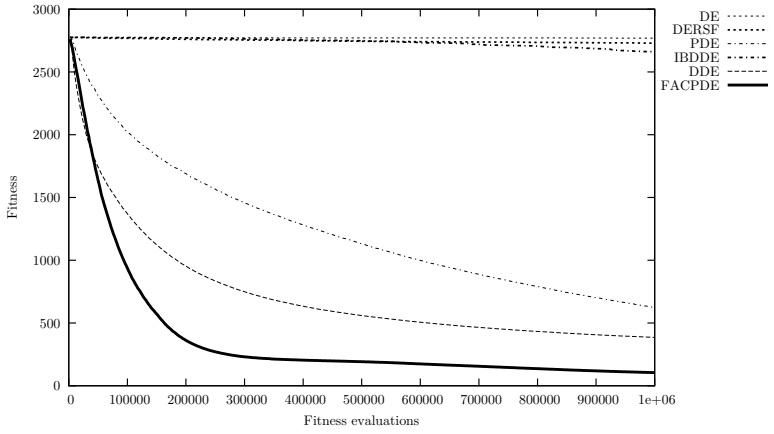
fitness evaluations in all cases except 9(h), where it occurs around 200,000 fitness evaluations. PDE's improvement rate is slower than DDE's, but contrary to the latter, it continuously improves its solutions and in all but one of the examples listed above, finds a better solution than DDE; Figure 9(c) is the exception, but PDE's solution is still very close to DDE's. Finally, FACPDE's improvement rate at the onset is steeper than DDE's in Figures 9(c), 9(f) and 9(j), similar in 9(i) and 9(k), and softer in Figures 9(d), 9(e), 9(g) and 9(h). In all cases however, FACPDE does not show symptoms of premature convergence as DDE does. Moreover, FACPDE's final solutions are, in all above examples except 9(h), better than PDE's.

Figure 10 shows average performance trends of the five considered algorithms over a selection of the test problems listed in Table 1 in 1000 dimensions.

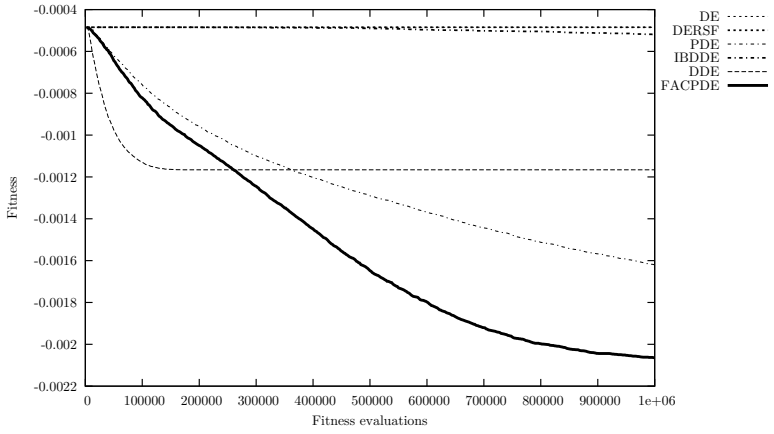
Qualitative results represented in Figure 10 confirm the findings in 500 dimensions. For large scale problems, the sequential DE seems to suffer from the curse of dimensionality. The IBDDE algorithm succeeds at improving upon the performance of sequential DE and DERSF. The other algorithms demonstrate a better behavior for the high dimensional problems. The DDE has good convergence speed performance during early stages of the evolution since it often manages to detect high quality solutions during the initial generations. However, the DDE tends to stop improving upon previous achievements quite quickly, see e.g. Figures 10(c) and 10(i). It is interesting to observe that in 1000 dimensions PDE is slower at detecting high quality solutions with respect to the 500-dimensional case e.g. compare the behavior with Michalewicz function in Figures 9(e) and 10(e). Thus, more often than in the



(a) Ackley

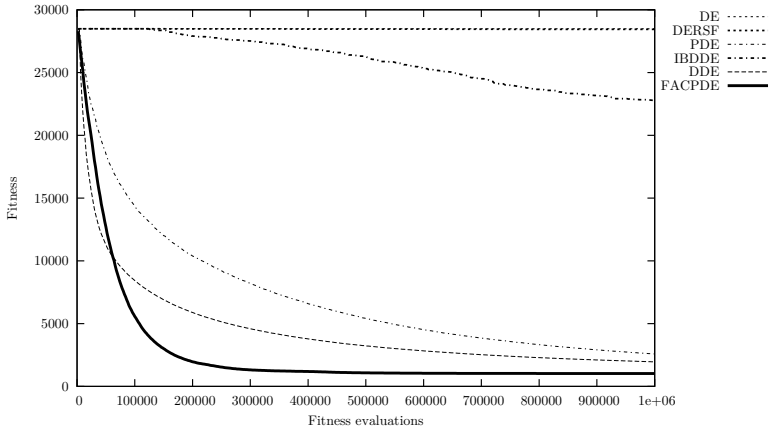


(b) Alpine

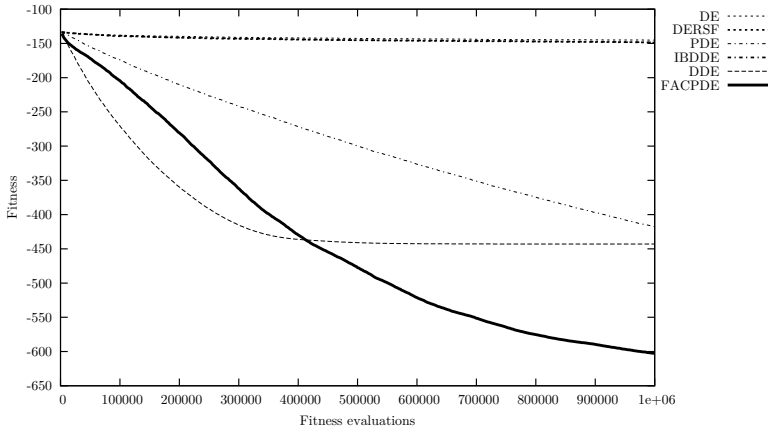


(c) DropWave

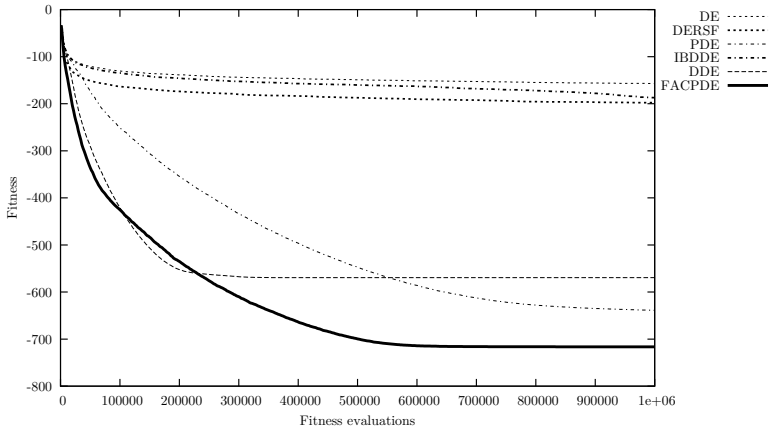
Fig. 10 Performance trends in 1000 dimensions



(d) Griewangk

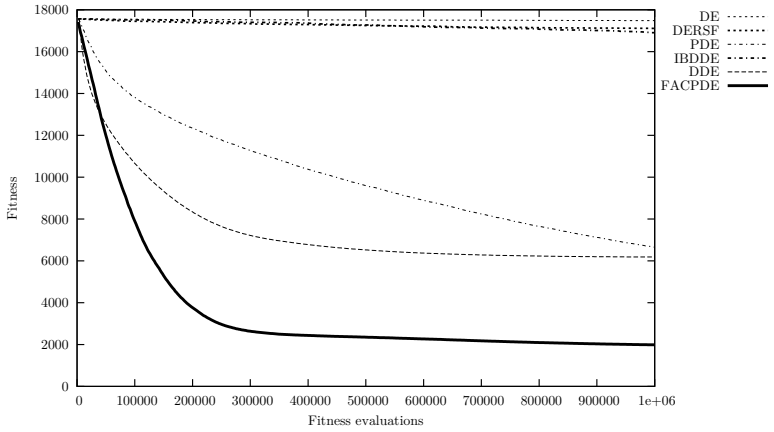


(e) Michalewicz

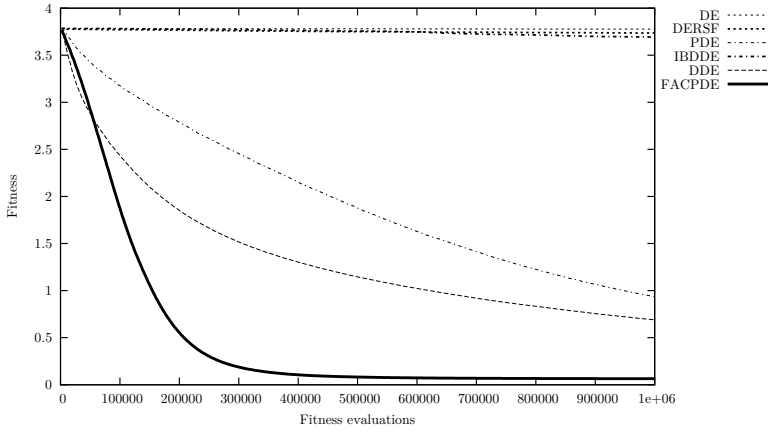


(f) Pathological

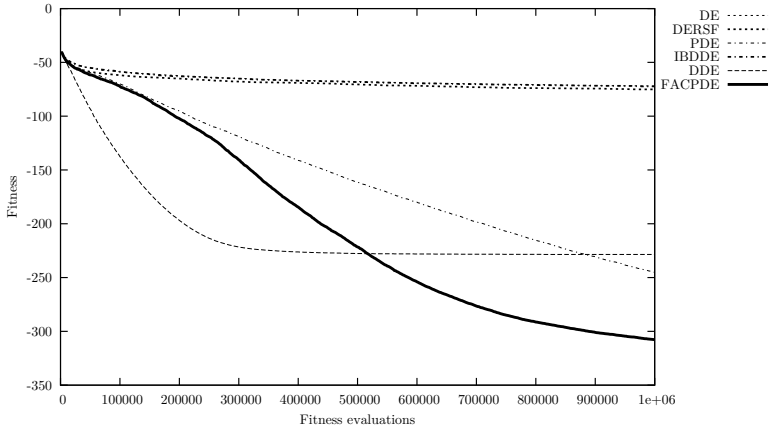
Fig. 10 Performance trends in 1000 dimensions (continued)



(g) Rastrigin



(h) Rotated Ackley



(i) Rotated Michalewicz

Fig. 10 Performance trends in 1000 dimensions (continued)

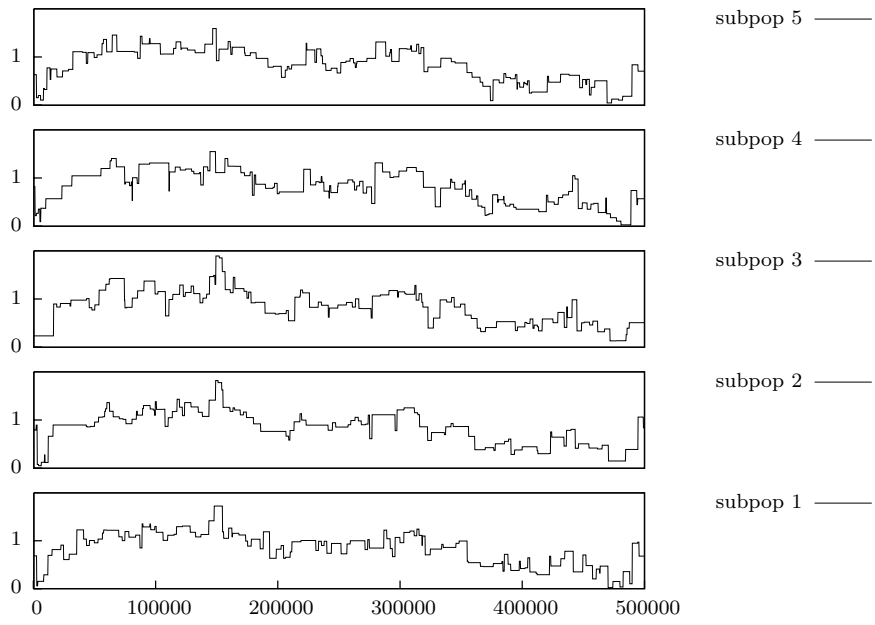


Fig. 11 Example of the scale factor behavior over the five sub-populations

500-dimensional case, the DDE outperforms PDE in high dimensions. The FACPDE algorithm, on the contrary, seems to be very efficient and outperforms, in most cases, both PDE and DDE. In addition, the gap between FACPDE and the second best algorithm performance seems to be systematically larger for the 1000-dimensional case with respect to the results in 500 dimensions, see e.g. Figures 10(a), 10(c), 10(e) and 10(h). In other words, the scale factor inheritance mechanism seems to be very beneficial in order to tackle large scale problems.

In order to better understand the working principle of the scale factor inheritance, Figure 11 has been included. Figure 11 shows an example, based on a single run, of the trend that the absolute value of F takes during the FACPDE evolution for the five distributed sub-populations.

It can be noticed from Figure 11 how the scale factor inheritance mechanism works. For example, at around 150,000 a sudden increase in the scale factor value occurs throughout all the sub-populations. This phenomenon is clearly visible in sub-populations 1,2, and 3 and proves how promising scale factor values are propagated within the ring of sub-populations. The most important finding in Figure 11 is related to the variation of the scale factor values. It can be observed that although the variation of F is, in principle, unbounded, there is no divergence in the trends. On the contrary, the scale factors never take a value greater than 2. In addition, the scale factor trends do not converge to a constant value. On the contrary, the trends continue to oscillate throughout the entire evolution. This fact can be seen as confirmation that there is no optimal scale factor for a given problem, but that a dynamic mechanism is required in order to satisfy the necessities of the evolution. Finally, since the evolution is biased by randomization, according to our interpretation, the

Table 10 Experimental Results for the IEEE CEC08 Benchmark

	$n = 100$	$n = 500$	$n = 1000$
F1	$3.94e+01 \pm 1.41e+02$	$2.45e+02 \pm 2.42e+02$	$3.76e+02 \pm 3.54e+02$
F2	$8.97e+01 \pm 7.68e+00$	$1.13e+02 \pm 7.20e+00$	$1.28e+02 \pm 9.92e+00$
F3	$8.49e+06 \pm 3.05e+07$	$2.39e+08 \pm 3.40e+08$	$3.35e+08 \pm 3.09e+08$
F4	$1.89e+02 \pm 7.18e+01$	$8.31e+02 \pm 1.66e+02$	$1.65e+03 \pm 3.32e+02$
F5	$8.64e-01 \pm 1.17e+00$	$4.18e+00 \pm 2.89e+00$	$1.36e+01 \pm 4.96e+01$
F6	$1.01e+01 \pm 3.60e+00$	$2.83e+00 \pm 2.24e+00$	$4.24e+00 \pm 4.41e+00$
F7	$-1.32e+03 \pm 5.09e+01$	$-6.31e+03 \pm 2.03e+02$	$-1.19e+04 \pm 9.28e+02$

scale factor control should also contain some randomization, as empirically shown in many papers on DE e.g. [23], [30], [37] and [38].

5.1 Experimental Results for a Large Scale Optimization Benchmark

The proposed FACPDE has been finally tested on the large scale benchmark settled for the 2008 IEEE Congress on Evolutionary Computation (IEEE CEC08) described in [68]. Following the suggestions given in [68], each function has been considered in 100, 500, and 1000 dimensions. For each problem, FACPDE has been has been run 25 times (25 independent runs) and the computational budget has been fixed equal to $5000 \times n$. Regarding the parameter setting, the FACPDE has been in 500 and 1000 dimensions with the same setting mentioned above. Following the same rate $\frac{n}{S_{pop}}$, in 100 dimensions, the population size S_{pop} has been set equal to 40. The final results, expressed in terms of fitness difference between the detected value and the actual minimum ($\pm$ related standard deviation) are given for the seven test problems contained in [68] for the three levels of dimensionality in Table 10

Although the results displayed in Table 10 are outperformed in several cases by the algorithms which took part in the CEC08 competition, FACPDE is still competitive in most cases. This fact is, in our opinion, very relevant since FACPDE, unlike most of those algorithms, does not employ local search components and did not undergo a parameter setting in order to have a high quality performance for these specific test problems.

6 Conclusion

This paper proposes an adaptive mechanism for the scale factor in distributed differential evolution schemes. The proposed mechanism, namely, scale factor inheritance, consists of the perturbation, by means of a random number and migration, of promising scale factor values throughout sub-populations arranged according to a ring topology. This mechanism is integrated within a distributed differential evolution which employs migration of those individuals demonstrating the best performance.

The resulting algorithm has been tested on a broad and various set of optimization problems and then compared with two sequential differential evolution algorithms and three distributed differential evolution schemes, recently proposed in literature. Numerical results show that the proposed approach is very promising and that the resulting algorithm displays excellent performance in terms of detected final

solutions, convergence speed, and robustness for all test problems and comparisons considered in this study.

On the basis of the obtained results and an analysis of the algorithmic working principles, some additional conclusions have been drawn. In distributed differential evolution a cooperative/competitive adaptation of the scale factor is beneficial to algorithmic performance and, more generally, the employment of multiple scale factors factors, updating of which can greatly improve performance of the distributed algorithm. In addition, a constant scale factor value is inadequate since it restricts the amount of search moves in differential evolution (both sequential and distributed). In other words, for a given problem there is no optimal scale factor since the optimal setting varies during the various stages of evolution. An optimal setting dynamically varies with distribution of the solutions within the decision space during evolution. Although an understanding of a proper control dynamic of the scale factor variation is not yet complete, according to our interpretation, it does not follow a progressive increase or decrease, but takes on an oscillatory behavior. Finally, since the evolution is affected by random events, mainly in the selection of individuals composing the moving vectors within the mutation operation, a certain randomization of the scale factor seems to be beneficial for a successful enhancement of the differential evolution performance.

References

1. R. Storn and K. Price, "Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces," Tech. Rep. TR-95-012, ICSI, 1995.
2. K. V. Price, R. Storn, and J. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization*. Springer, 2005.
3. U. K. Chakraborty, ed., *Advances in Differential Evolution*, vol. 143 of *Studies in Computational Intelligence*. Springer, 2008.
4. 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.
5. C.-T. Su and C.-S. Lee, "Network reconfiguration of distribution systems using improved mixed-integer hybrid differential evolution," *IEEE Transactions on Power Delivery*, vol. 18, no. 3, pp. 1022–1027, 2003.
6. J.-P. Chiou, C.-F. Chang, and C.-T. Su, "Ant direction hybrid differential evolution for solving large capacitor placement problems," *IEEE Transactions on Power Systems*, vol. 19, no. 4, pp. 1794–1800, 2004.
7. F.-S. Wang and H.-J. Jang, "Parameter estimation of a bioreaction model by hybrid differential evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 1, pp. 410–417, 2000.
8. R. Storn, "Designing nonstandard filters with differential evolution," *IEEE Signal Processing Magazine*, vol. 22, no. 1, pp. 103–106, 2005.
9. R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.
10. K. Price and R. Storn, "Differential evolution: A simple evolution strategy for fast optimization," *Dr. Dobb's J. Software Tools*, vol. 22, no. 4, pp. 18–24, 1997.
11. J. Liu and J. Lampinen, "On setting the control parameter of the differential evolution algorithm," in *Proceedings of the 8th international Mendel conference on soft computing*, pp. 11–18, 2002.
12. J. Rönkkönen, S. Kukkonen, and K. V. Price, "Real-parameter optimization with differential evolution," in *Proceedings of IEEE International Conference on Evolutionary Computation*, vol. 1, pp. 506–513, 2005.

13. K. Zielinski, P. Weitkemper, R. Laur, and K.-D. Kammeyer, "Parameter study for differential evolution using a power allocation problem including interference cancellation," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1857–1864, 2006.
14. D. Wolpert and W. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.
15. R. Gämperle, S. D. Müller, and P. Koumoutsakos, "A parameter study for differential evolution," in *Proceedings of the Conference in Neural Networks and Applications (NNA), Fuzzy Sets and Fuzzy Systems (FSFS) and Evolutionary Computation (EC)*, pp. 293–298, WSEAS, 2002.
16. R. Mallipeddi and P. N. Suganthan, "Empirical study on the effect of population size on differential evolution algorithm," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 3663–3670, 2008.
17. J. Teo, "Differential evolution with self-adaptive populations," in *Knowledge-Based Intelligent Information and Engineering Systems*, vol. 3681 of *Lecture Notes in Computer Science*, pp. 1284–1290, Springer, 2005.
18. J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Computing – A Fusion of Foundations, Methodologies and Applications*, vol. 10, no. 8, pp. 673–686, 2006.
19. 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.
20. J. Brest and M. S. Maučec, "Population size reduction for the differential evolution algorithm," *Applied Intelligence*, vol. 29, no. 3, pp. 228–247, 2008.
21. V. Tirronen and F. Neri, "Differential evolution with fitness diversity self-adaptation," in *Nature-Inspired Algorithms for Optimisation* (R. Chiong, ed.), vol. 193 of *Studies in Computational Intelligence*, pp. 199–234, Springer, 2009.
22. M. M. Ali and A. Törn, "Population set-based global optimization algorithms: Some modifications and numerical studies," *Computer Operational Research*, vol. 31, no. 10, pp. 1703–1725, 2004.
23. 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*, pp. 991–998, ACM, 2005.
24. J. Rönkkönen and J. Lampinen, "On using normally distributed mutation step length for the differential evolution algorithm," in *Proceedings of Ninth International MENDEL Conference on Soft Computing* (R. Matousek and P. Osmera, eds.), pp. 11–18, 2003.
25. M. G. Omran, A. Salman, and A. P. Engelbrecht, "Self-adaptive differential evolution," in *Computational Intelligence and Security*, vol. 3801 of *Lecture Notes in Computer Science*, pp. 192–199, Springer, 2005.
26. A. Salman, A. P. Engelbrecht, and M. G. Omran, "Empirical analysis of self-adaptive differential evolution," *European Journal of Operational Research*, vol. 183, no. 2, pp. 785–804, 2007.
27. O. S. Soliman, L. T. Bui, and H. A. Abbass, "The effect of a stochastic step length on the performance of the differential evolution algorithm," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2850–2857, 2007.
28. O. S. Soliman and L. T. Bui, "A self-adaptive strategy for controlling parameters in differential evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2837–2842, 2008.
29. G. Zhenyu, C. Bo, Y. Min, and C. Binggang, "Self-adaptive chaos differential evolution," in *Advances in Natural Computation*, vol. 4221 of *Lecture Notes in Computer Science*, pp. 972–975, Springer, 2006.
30. 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.
31. H.-Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *Journal of Global Optimization*, vol. 27, no. 1, pp. 105–129, 2003.
32. J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Computing - A Fusion of Foundations, Methodologies and Applications, Springer*, vol. 9, pp. 448–462, June 2005.
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. V. Tirronen, F. Neri, T. Kärkkäinen, K. Majava, and T. Rossi, "An enhanced memetic differential evolution in filter design for defect detection in paper production," *Evolutionary Computation*, vol. 16, pp. 529–555, 2008.
35. 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.
36. F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Computing Journal*, vol. 1, no. 2, pp. 153–171, 2009.
37. 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, pp. 398–417, 2009.
38. 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, pp. 526–553, 2009.
39. W. Kwedlo and K. Bandurski, "A parallel differential evolution algorithm," in *Proceedings of the IEEE International Symposium on Parallel Computing in Electrical Engineering*, pp. 319–324, 2006.
40. M. Salomon, G.-R. Perrin, F. Heitz, and J.-P. Armspach, "Parallel differential evolution: Application to 3-d medical image registration," in *Differential Evolution–A Practical Approach to Global Optimization* (K. V. Price, R. M. Storn, and J. A. Lampinen, eds.), Natural Computing Series, ch. 7, pp. 353–411, Springer, 2005.
41. D. Zaharie, "Parameter adaptation in differential evolution by controlling the population diversity," in *Proceedings of the International Workshop on Symbolic and Numeric Algorithms for Scientific Computing* (D. Petcu et al, ed.), pp. 385–397, 2002.
42. D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in *Proceedings of MENDEL International Conference on Soft Computing* (D.Matousek and P. Osmera, eds.), pp. 41–46, 2003.
43. D. Zaharie and D. Petcu, "Parallel implementation of multi-population differential evolution," in *Proceedings of the NATO Advanced Research Workshop on Concurrent Information Processing and Computing*, pp. 223–232, IOS Press, 2003.
44. D. Zaharie, "A multipopulation differential evolution algorithm for multimodal optimization," in *Proceedings of Mendel International Conference on Soft Computing* (R. Matousek and P. Osmera, eds.), pp. 17–22, 2004.
45. D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2023–2029, 2004.
46. N. G. Pavlidis, D. K. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. N. Vrahatis, "Spiking neural network training using evolutionary algorithms," in *Proceedings of the IEEE International Joint Conference on Neural Networks*, pp. 2190–2194, 2005.
47. K. N. Kozlov and A. M. Samsonov, "New migration scheme for parallel differential evolution," in *Proceedings of the International Conference on Bioinformatics of Genome Regulation and Structure*, pp. 141–144, 2006.
48. I. D. Falco, A. D. Cioppa, D. Maisto, U. Scafuri, and E. Tarantino, "Satellite image registration by distributed differential evolution," in *Applications of Evolutionary Computing*, vol. 4448 of *Lectures Notes in Computer Science*, pp. 251–260, Springer, 2007.
49. I. D. Falco, D. Maisto, U. Scafuri, E. Tarantino, and A. D. Cioppa, "Distributed differential evolution for the registration of remotely sensed images," in *Proceedings of the IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pp. 358–362, 2007.
50. I. D. Falco, U. Scafuri, E. Tarantino, and A. D. Cioppa, "A distributed differential evolution approach for mapping in a grid environment," in *Proceedings of the IEEE Euromicro International Conference on Parallel, Distributed and Network-Based Processing*, pp. 442–449, 2007.
51. J. Apolloni, G. Leguizamón, J. García-Nieto, and E. Alba, "Island based distributed differential evolution: An experimental study on hybrid testbeds," in *Proceedings of the IEEE International Conference on Hybrid Intelligent Systems*, pp. 696–701, 2008.
52. M. S. Nipteni, I. Valakos, and I. Nikolos, "An asynchronous parallel differential evolution algorithm," in *Proceedings of the ERCOFTAC Conference on Design Optimisation: Methods and Application*, 2006.
53. J. Lampinen, "Differential evolution - new naturally parallel approach for engineering design optimization," in *Developments in Computational Mechanics with High Performance Computing* (B. H. Topping, ed.), pp. 217–228, Civil-Comp Press, 1999.

54. 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, pp. 1785–1791, 2005.
55. K. V. Price, "Mechanical engineering design optimization by differential evolution," in *New Ideas in Optimization* (D. Corne, M. Dorigo, and F. Glover, eds.), pp. 293–298, McGraw-Hill, 1999.
56. V. Feoktistov, *Differential Evolution in Search of Solutions*. Springer, 2006.
57. J. Lampinen and I. Zelinka, "On stagnation of the differential evolution algorithm," in *Proceedings of 6th International Mendel Conference on Soft Computing* (P. Ôsmera, ed.), pp. 76–83, 2000.
58. 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.
59. 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.
60. A. Caponio, G. L. Cascella, F. Neri, N. Salvatore, and M. Sumner, "A fast adaptive memetic algorithm for on-line and off-line control design of pmsm drives," *IEEE Transactions on System Man and Cybernetics-part B*, vol. 37, no. 1, pp. 28–41, 2007.
61. 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.
62. K. Zielinski and R. Laur, "Stopping criteria for differential evolution in constrained single-objective optimization," in *Advances in Differential Evolution* (U. K. Chakraborty, ed.), vol. 143 of *Studies in Computational Intelligence*, pp. 111–138, Springer, 2008.
63. F. Neri and V. Tirronen, "On memetic differential evolution frameworks: a study of advantages and limitations in hybridization," in *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 2135–2142, 2008.
64. E. Alba and M. Tomassini, "Parallelism and evolutionary algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 5, pp. 443–462, 2002.
65. F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
66. S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.
67. S. García, A. Fernández, 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.
68. 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," tech. rep., Nature Inspired Computation and Applications Laboratory, USTC, China, 2007.

PIV

PARALLEL RANDOM INJECTION DIFFERENTIAL EVOLUTION

by

Matthieu Weber, Ferrante Neri and Ville Tirronen 2010

In Applications of Evolutionary Computation, volume 6024/2010 of *Lecture Notes in Computer Science*, pages 471-480

Parallel Random Injection Differential Evolution*

Matthieu Weber, Ferrante Neri, and Ville Tirronen

Department of Mathematical Information Technology,
University of Jyväskylä, P.O. Box 35 (Agora), FI-40014, Finland
{matthieu.weber, ferrante.neri, ville.tirronen}@jyu.fi

Abstract. This paper proposes the introduction of a generator of random individuals within the ring topology of a Parallel Differential Evolution algorithm. The generated random individuals are then injected within a sub-population. A crucial point in the proposed study is that a proper balance between migration and random injection can determine the success of a distributed Differential Evolution scheme. An experimental study of this balance is carried out in this paper. Numerical results show that the proposed Parallel Random Injection Differential Evolution seems to be a simple, robust, and efficient algorithm which can be used for various applications. An important finding of this paper is that premature convergence problems due to an excessively frequent migration can be overcome by the injection of random individuals. In this way, the resulting algorithm maintains the high convergence speed properties of a parallel algorithm with high migration but differs in that it is able to continue improving upon the available genotypes and detect high quality solutions.

1 Introduction

Differential Evolution (DE), see [1], is a versatile optimizer which has shown high performance in several applications, especially continuous problems, for example [2]. As highlighted in [3], the success of DE is contained in its implicit self-adaptation which allows an extensive domain exploration during early stages of the evolution and a progressive narrowing of the search within the most promising areas of the decision space. Although this mechanism is effective, it conceals a drawback: the DE contains a limited amount of search moves which could cause the population to fail at enhancing upon the available genotypes, thus resulting in a stagnation condition. In order to overcome this drawback, computer scientists in recent years have attempted to improve the DE performance by modifying the basic DE. Some popular examples of this scientific trend are contained in [4] where multiple search logics are employed, in [5] where the offspring are generated by combining two mating pools (one global and one local,

* This research is supported by the Academy of Finland, Akatemiatutkija 00853, Algorithmic Design Issues in Memetic Computing

respectively), and in [6] where a randomization of the parameters increases the variability of the potential search moves.

Another popular way to enhance the DE performance is through employment of structured populations. In [7] a distributed DE scheme employing a ring topology (the cores are interconnected in a circle and the migrations occur following the ring) has been proposed for the training of a neural network. In [8], an example of DE parallelization is given for a medical imaging application. A few famous examples of distributed DE are presented in [9] and [10]; in these papers the migration mechanism and the algorithmic parameters are adaptively coordinated according to criterion based on genotypical diversity. In paper [9], a distributed DE that preserves diversity in the niches is proposed in order to solve multi-modal optimization problems. In [11], a distributed DE characterized by a ring topology and the migration of individuals with the best performance, to replace random individuals of the neighbor sub-population, has been proposed. Following similar logic, paper [12] proposes a distributed DE where the computational cores are arranged according to a ring topology and, during migration, the best individual of a sub-population replaces the oldest member of the neighboring population. In [13] a distributed DE has been designed for the image registration problem. In these papers, a computational core acts as a master by collecting the best individuals detected by the various sub-populations running in slave cores. The slave cores are connected in a grid and a migration is arranged among neighbor sub-populations. In [14], a distributed DE which modifies the scheme proposed in [11] has been presented. In [14], the migration is based on a probabilistic criterion depending on five parameters. It is worthwhile mentioning that some parallel implementations of sequential (without structured population) DE are also available in literature, see [15]. An investigation of DE parallelization is given in [16].

This paper focuses on distributed DE and in particular on the ring topology scheme presented in [11]. The main novelty in Parallel Differential Evolution (PDE) described in [11] consists of the migration scheme and its related probability: the DE is independently performed on each sub-population composing the ring and, at the end of each generation, with a certain probability the individual with the best performance is copied in the neighbor sub-population and replaces a randomly selected individual from the target sub-population. In [11] a compromise value of migration probability is proposed. In this paper we propose a novel algorithm based on PDE, namely Parallel Random Injection Differential Evolution (PRIDE). The PRIDE algorithm includes within the ring topology a random generator of candidate solutions: at the end of each generation a random individual replaces a randomly selected individual from a sub-population of the ring topology. This simple mechanism, if properly used, can have a major impact on the algorithmic performance of the original PDE scheme. In addition, the resulting effect of migration (with its probability levels) and random injection is a very interesting topic and has thus been analyzed in this paper.

The remainder of this article is organized in the following way. Section 2 describes the working principles of DE, PDE and PRIDE. Section 3 shows the

experimental setup and numerical results of the present study. Section 4 gives the conclusions of this paper.

2 Parallel Random Injection Differential Evolution

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

At the beginning of the optimization process S_{pop} individuals are pseudo-randomly sampled with a uniform distribution function within the decision space D (for simplicity, the term random will be used instead of pseudo-random in the reminder of this paper). The S_{pop} individuals constituting the populations are distributed over the m sub-populations composing the ring. Each sub-population is composed of $\frac{S_{pop}}{m}$ individuals.

Within each sub-population a standard DE, following its original definition, is performed. At each generation, for each individual x_i 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 as:

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

where $F \in [0, 1+[$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated. With $F \in [0, 1+[$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1, see [1]. While there is no theoretical upper limit for F, effective values are rarely greater than 1.0. The mutation scheme shown in Equation (1) is also known as DE/rand/1. It is worth mentioning that there exist many other mutation variants, see [4].

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

$$x_{off,j} = \begin{cases} x_{off,j} & \text{if } rand(0,1) \leq CR \\ x'_{i,j} & otherwise \end{cases} \tag{2}$$

where $rand(0, 1)$ is a random number between 0 and 1; j is the index of the gene under examination.

The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) \leq f(x_i)$; otherwise no replacement occurs. It must be remarked that although the replacement indexes are saved one by one during generation, actual replacements occur all at once at the end of the generation.

According to its original implementation, PDE uses the Parallel Virtual Machine (PVM), allowing multiple computers (called *nodes*) to be organized as a cluster and exchange arbitrary messages. PDE is organized around one master

node and m sub-populations running each on one node, and organized as a unidirectional ring. It must be noted that although the logical topology is a ring which does not contain the master node, the actual topology is a star, where all communications (i.e., the migrations of individuals) are passing through the master.

Each sub-population runs a regular DE algorithm while the master node coordinates the migration of individuals between sub-populations. On each generation, the sub-population has a given probability ϕ to send a copy of its best individual to its next neighbor sub-population in the ring. When migration occurs, the migrating individual replaces a randomly selected individual belonging to the target sub-population.

The PRIDE introduces an apparently minor modification to the PDE scheme. At the end of each generation, according to a certan probability ψ a random individual is generated and inserted within a randomly selected sub-population by replacing a randomly selected individual. However, the replacement cannot involve the individual with the best performance in that sub-population, which is saved in an elitist fashion. For the sake of clarity a scheme highlighting the working principles of PRIDE is shown in Figure 1.

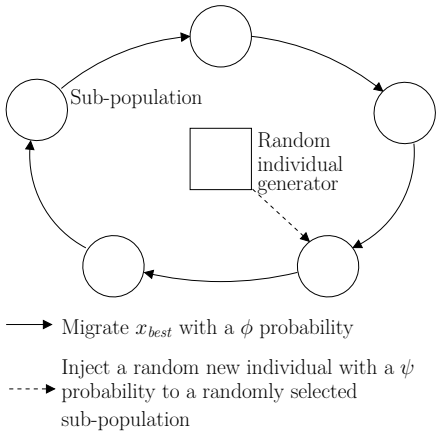


Fig. 1. Working principle of the Parallel Random Injection Differential Evolution

In order to understand the rationale behind the proposed mechanism, it is important to analyze the concept of parallelism and migration in a PDE scheme. As highlighted above, in panmictic DE, stagnation occurs when the algorithm does not manage to improve upon any solution of its population for a prolonged number of generations. In other words, for each stage of the optimization process, DE can generate a limited amount of exploratory moves. If these moves are not enough for generating new promising solutions, the search can be heavily compromised.

Thus, in order to enhance the DE performance, alternative search moves should support the original scheme and promote a successful continuation of the optimization process. The use of multiple populations in distributed DE algorithms allows an observation of the decision space from various perspectives and, most importantly, decreases the risk of stagnation since each sub-population imposes a high exploitation pressure. In addition, the migration mechanism ensures that solutions with a high performance are included within the sub-populations during their evolution. This fact is equivalent to modifying the set of search moves. If the migration privileges the best solutions, the new search moves pro-

mote detection of new promising search directions and thus allow the DE search structure to be periodically "refurbished". Thus, the migration is supposed to mitigate risk of stagnation of the DE sub-populations and to enhance global algorithmic performance.

Migration in PDE has a very interesting effect. Low migration probability values (low values of ϕ) make the algorithm rather explorative, i.e. the PDE slowly improves upon the available solutions and eventually detects individuals characterized by a (relatively) high performance. On the other hand, high migration probability values produce very quick improvements during the early stages of the optimization but eventually cause a premature convergence. The final values detected in these cases may likely have an unsatisfactory performance. This fact can be easily explained as the consequence of a diversity loss due to a high migration: the best individuals are too often copied into the target sub-populations thus causing an excessively aggressive/exploitative algorithmic behavior. In order to obtain an algorithm which produces high quality results with a reasonably good convergence speed, in [11] a compromise on the ϕ value (subsequent to a tuning) has been implemented.

This paper proposes the idea of making use of the highly exploitative behavior of a PDE with a high migration probability, but to inhibit premature convergence by including new genotypes within the ring structure. The algorithmic philosophy employed here is a balance between two opposite and conflicting actions: the migration exploits available search directions by promoting quick improvements upon available genotypes (ϕ action) while the random injection mechanism is supposed to generate completely new genotypes which assist the PDE framework in testing novel promising search directions (ψ action) thus avoiding diversity loss. Analogous to the balance between exploration and exploitation in Evolutionary Algorithms, a proper balance between ϕ and ψ actions determine the success of the proposed PRIDE algorithm.

3 Experimental Results

The test problems listed in Table 1 have been considered in this study.

The rotated version of some of the test problems listed in Table 1 have been included into the benchmark set. These rotated problems have been generated through the multiplication of the vector of variables by a sparse orthogonal rotation matrix, created by composing n rotations of random angles (uniformly sampled in $[-\pi, \pi]$), one around each of the n axes of the search space. The boundaries of rotated problems are kept the same as for the non-rotated problems. In total, ten test problems have been considered in this study with $n = 500$. The choice of the relatively high dimensionality for the test problems in this study is due to the fact that the advantages of PDE-based algorithms with respect to panmictic DE is not evident for low dimensional problems, see [17].

In order to prove the effectiveness of the random injection and its relationship with the migration constant, for each test problem the PDE and PRIDE algorithms have been run with ϕ equal to 0.2 (suggested value in [11]), 0.5, and

Table 1. Test Problems

Test Problem	Function	Decision Space	Features
Ackley	$-20 + e + 20\exp\left(-\frac{0.2}{n}\sqrt{\sum_{i=1}^{n} x_i^2}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i)x_i\right)$	$[-1,1]^n$	multimodal, non-separable
Alpine	$\sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1x_i\rvert$	$[-10,10]^n$	multimodal, separable
DeJong's Sphere	$\lVert x\rVert^2$	$[-5.12,5.12]^n$	unimodal, separable
Michalewicz	$-\sum_{i=1}^{n} \sin x_i \left(\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)^{20}$	$[0,\pi]^n$	multimodal, separable
Rastrigin	$10n + \sum_{i=0}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12,5.12]^n$	multimodal, separable
Schwefel	$-\sum_{i=1}^{n} x_i \sin\left(\sqrt{\lvert x_i\rvert}\right)$	$[-500,500]^n$	multimodal, separable

1 respectively. On the basis of a preliminary tuning, we decided to set $\psi = 1$ for all the versions of PRIDE contained in this paper. All the algorithms in this study have been run with populations of 200 individuals divided into 5 subpopulations of 40 individuals each. Regarding scale factor and crossover rate, F has been set equal to 0.7 and CR equal to 0.1 for all the algorithms in this study. Each algorithm has undergone 50 independent runs for each test problem; rotated problems have been rotated using the same matrix on each run. Each single run has been performed for $500,000$ fitness evaluations.

Table 2 shows the average of the final results detected by each algorithm $\pm$ the standard deviations, with the 500 dimension case. Results are organized in PDE vs PRIDE pairs. The best results of each pairwise comparison are highlighted in bold face. The best overall results for each problem are also underlined. With the notation PDE-ϕ and PRIDE-ϕ we mean the PDE and PRIDE algorithms, respectively, running with the corresponding ϕ value. In order to prove statistical significance of the results, the Wilcoxon Rank-sum test has been applied according to the description given in [18] for a confidence level of 0.95. Table 3 shows results of the test. A "+" indicates the case in which PRIDE statistically outperforms its corresponding PDE variant; a "=" indicates that a pairwise comparison leads to success of the Wilcoxon test, i.e., the two algorithms have the same performance; a "-" indicates that PRIDE is outperformed. In order to carry out a numerical comparison of the convergence speed performance for each test problem, the average final fitness value returned by the best performing algorithm (of each pair PDE vs PRIDE under consideration) G has been considered. Subsequently, the average fitness value at the beginning of the optimization process J has also been computed. The threshold value $THR = J - 0.95(J - G)$ has then been calculated. The value THR represents 95% of the decay in the fitness value of the algorithm displaying the best performance. If during a certain run an algorithm succeeds in reaching the value THR, the run is said to be successful. For each test problem, the average amount of fitness evaluations $\bar{ne}$ required for

Table 2. Fitness ± standard deviation

	PDE-0.2	PRIDE-0.2	PDE-0.5
Ackley	$\mathbf{1.62e-01 \pm 1.67e-02}$	$2.86e-01 \pm 1.92e-02$	$1.31e-01 \pm 5.49e-02$
Alpine	$\mathbf{8.88e+01 \pm 1.26e+01}$	$1.22e+02 \pm 1.36e+01$	$8.45e+01 \pm 1.64e+01$
DeJong's Sphere	$\mathbf{1.92e+01 \pm 3.57e+00}$	$4.82e+01 \pm 6.00e+00$	$\mathbf{1.17e+01 \pm 5.17e+00}$
Michalewicz	$\mathbf{-3.06e+02 \pm 5.68e+00}$	$-2.95e+02 \pm 6.50e+00$	$-2.86e+02 \pm 9.56e+00$
Rastrigin	$\mathbf{1.91e+03 \pm 9.94e+01}$	$1.92e+03 \pm 9.04e+01$	$2.24e+03 \pm 1.60e+02$
Schwefel	$-1.30e+05 \pm 3.17e+03$	$\mathbf{-1.32e+05 \pm 2.35e+03}$	$-1.23e+05 \pm 4.11e+03$
Rt. Ackley	$\mathbf{2.15e-01 \pm 2.50e-02}$	$3.67e-01 \pm 3.33e-02$	$\mathbf{1.72e-01 \pm 5.72e-02}$
Rt. Michalewicz	$\mathbf{-1.76e+02 \pm 7.76e+00}$	$-1.76e+02 \pm 5.86e+00$	$-1.52e+02 \pm 7.48e+00$
Rt. Rastrigin	$\mathbf{1.95e+03 \pm 1.51e+02}$	$2.09e+03 \pm 1.06e+02$	$2.34e+03 \pm 1.96e+02$
Rt. Schwefel	$-1.65e+05 \pm 4.74e+03$	$\mathbf{-1.65e+05 \pm 4.78e+03}$	$-1.53e+05 \pm 6.10e+03$

	PRIDE-0.5	PDE-1.0	PRIDE-1.0
Ackley	$\mathbf{1.24e-01 \pm 1.17e-02}$	$2.80e-01 \pm 9.08e-02$	$\mathbf{9.99e-02 \pm 1.10e-02}$
Alpine	$\mathbf{5.56e+01 \pm 7.21e+00}$	$1.59e+02 \pm 2.47e+01$	$\mathbf{4.08e+01 \pm 5.34e+00}$
DeJong's Sphere	$1.32e+01 \pm 2.35e+00$	$4.85e+01 \pm 1.89e+01$	$\mathbf{8.41e+00 \pm 1.70e+00}$
Michalewicz	$\mathbf{-3.55e+02 \pm 5.45e+00}$	$-2.56e+02 \pm 9.29e+00$	$\mathbf{-3.83e+02 \pm 4.21e+00}$
Rastrigin	$\mathbf{1.40e+03 \pm 7.85e+01}$	$2.82e+03 \pm 1.88e+02$	$\mathbf{1.17e+03 \pm 6.07e+01}$
Schwefel	$\mathbf{-1.48e+05 \pm 1.94e+03}$	$-1.12e+05 \pm 4.54e+03$	$\mathbf{-1.57e+05 \pm 1.86e+03}$
Rt. Ackley	$1.95e-01 \pm 2.66e-02$	$3.11e-01 \pm 8.27e-02$	$\mathbf{1.65e-01 \pm 2.40e-02}$
Rt. Michalewicz	$\mathbf{-2.09e+02 \pm 4.57e+00}$	$-1.34e+02 \pm 7.50e+00$	$\mathbf{-2.24e+02 \pm 4.47e+00}$
Rt. Rastrigin	$\mathbf{1.67e+03 \pm 9.08e+01}$	$2.95e+03 \pm 2.32e+02$	$\mathbf{1.51e+03 \pm 6.93e+01}$
Rt. Schwefel	$\mathbf{-1.71e+05 \pm 4.55e+03}$	$-1.36e+05 \pm 6.61e+03$	$\mathbf{-1.74e+05 \pm 4.35e+03}$

Table 3. Wilcoxon Rank-Sum test (PDE vs. corresponding PRIDE)

	PDE-0.2	PDE-0.5	PDE-1.0
Ackley	-	=	+
Alpine	-	+	+
DeJong's Sphere	-	-	+
Michalewicz	-	+	+
Rastrigin	=	+	+
Schwefel	+	+	+
Rt. Ackley	-	-	+
Rt. Michalewicz	=	+	+
Rt. Rastrigin	-	+	+
Rt. Schwefel	=	+	+

each algorithm to reach *THR* has been computed. Subsequently, the Q-test (Q stands for Quality) described in [3] has been applied. For each test problem and each algorithm, the Q measure is computed as $Q = \frac{\bar{n}e}{R}$ where the robustness R is the percentage of successful runs. It is clear that, for each test problem, the smallest value equals the best performance in terms of convergence speed. The value "∞" means that $R = 0$, i.e., the algorithm never reached the *THR*. Results of the Q-test are given in Table 4. Best results are highlighted in bold face.

Table 4. Q-test

	PDE-0.2	PRIDE-0.2	PDE-0.5	PRIDE-0.5	PDE-1.0	PRIDE-1.0
Ackley	**3.75e+03**	4.59e+03	3.43e+03	**3.29e+03**	7.19e+03	**3.05e+03**
Alpine	**3.84e+03**	4.74e+03	3.33e+03	**3.22e+03**	∞	**2.93e+03**
DeJong's Sphere	**2.28e+03**	2.82e+03	**1.69e+03**	1.92e+03	1.75e+03	**1.63e+03**
Michalewicz	**4.36e+03**	8.51e+03	∞	**4.39e+03**	∞	**4.25e+03**
Rastrigin	**3.74e+03**	4.18e+03	∞	**3.64e+03**	∞	**3.61e+03**
Schwefel	**4.27e+03**	4.30e+03	∞	**4.02e+03**	∞	**4.03e+03**
Rt. Ackley	**3.79e+03**	5.79e+03	**3.40e+03**	3.70e+03	5.60e+03	**3.39e+03**
Rt. Michalewicz	**4.57e+03**	4.84e+03	∞	**4.30e+03**	∞	**4.19e+03**
Rt. Rastrigin	**3.87e+03**	4.60e+03	6.44e+04	**3.61e+03**	∞	**3.49e+03**
Rt. Schwefel	**4.06e+03**	4.43e+03	4.73e+04	**3.91e+03**	∞	**3.92e+03**

Results can be analyzed from two complementary perspectives: by considering the pairwise comparisons for fixed values of migration constant ϕ and by considering the entire experimental setup. By analyzing the pairwise comparisons, it is clear that in the case $\phi = 0.2$ the random injection does not lead to any improvement; on the contrary in most cases it leads to a worsening of the algorithmic performance and a slowdown in the convergence. In the case $\phi = 0.5$, employment of random injection is advisable since it succeeds at improving upon the PDE performance in seven cases out of the ten considered in this study. However, not in all cases is this extra component beneficial. Finally in the case $\phi = 1$, there is a clear benefit in the employment of random injection in terms of both final results and convergence speed. In particular Q-test results show that the PDE algorithm in the comparison PDE-1.0 vs PRIDE-1.0 displays many ∞ values, i.e. it is not competitive with respect to its PRIDE variant. This fact can be seen as a confirmation that the introduction of novel (random) individuals within the ring topology can be an efficient countermeasure against premature convergence.

An analysis of the entire experimental setup shows that the PRIDE with $\phi = \psi = 1$ is a very efficient algorithm which outperforms (at least for those tests carried out in this paper) all the other algorithms in this study, in particular the PDE with its most promising parameter tuning ($\phi = 0.2$). More specifically the PRIDE, analogous to the PDE-1.0, tends to reach improvements very quickly during early stages of the evolution, but instead of converging prematurely continues generating solutions with a high performance and eventually detects good results. According to our interpretation, this algorithmic behavior is due to an efficient balance in the exploitation of promising search directions

resulting from the frequent migrations and testing of unexplored areas of the decision space resulting from the random injections. Both these components within a DE structure seem to compensate for the lack of potential search moves and the stagnation inconvenience.

For sake of clarity an example of the performance trend is shown in Figure 2(a). In order to highlight the benefit of random injection, the trend of the average fitness ± standard deviation is shown in Figure 2(b).

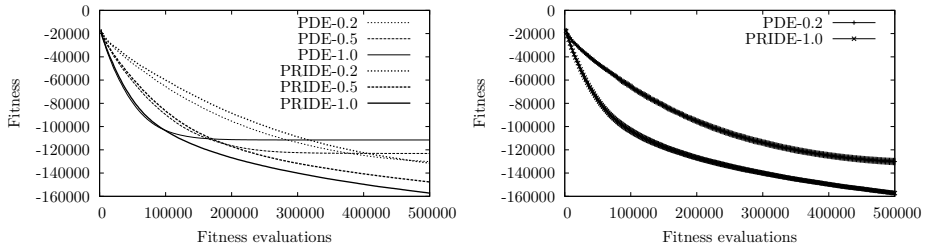


Fig. 2. Performance trend (Schwefel)

4 Conclusion

This paper proposes the generation of random solutions during the evolution of a Parallel Differential Evolution algorithm previously proposed in literature. These random solutions are intended to propose new promising search directions to the sub-populations during the optimization process. An analysis of the algorithmic performance, dependant on the value of migration probability, has been carried out. Experimental results show that while random injections do not lead to benefits for low values of migration probabilities, they could be extremely beneficial in conjunction with high migration probabilities. In particular, an algorithm combining frequent migrations and random injections seems very robust and efficient for various problems and seems to outperform the original Parallel Differential Evolution regardless of its parameter setting. An important finding of this research is the relationship between high migration and random injections which can be seen as parametrization, in relation to Parallel Differential Evolution, of the general concept of balance between exploration and exploitation in Evolutionary Computation.

References

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

2. Tirronen, V., Neri, F., Kärkkäinen, T., Majava, K., Rossi, T.: An enhanced memetic differential evolution in filter design for defect detection in paper production. Evolutionary Computation **16** (2008) 529–555
3. Feoktistov, V. In: Differential Evolution in Search of Solutions. Springer (2006) 83–86
4. Qin, A.K., Huang, V.L., Suganthan, P.N.: Differential evolution algorithm with strategy adaptation for global numerical optimization. IEEE Transactions on Evolutionary Computation **13** (2009) 398–417
5. Das, S., Abraham, A., Chakraborty, U.K., Konar, A.: Differential evolution with a neighborhood-based mutation operator. IEEE Transactions on Evolutionary Computation (2009) to appear.
6. Brest, J., Greiner, S., Bošković, B., Mernik, M., Žumer, V.: Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. IEEE Transactions on Evolutionary Computation **10**(6) (2006) 646–657
7. Kwedlo, W., Bandurski, K.: A parallel differential evolution algorithm. In: Proceedings of the IEEE International Symposium on Parallel Computing in Electrical Engineering. (2006) 319–324
8. Salomon, M., Perrin, G.R., Heitz, F., Armspach, J.P.: Parallel differential evolution: Application to 3-d medical image registration. In Price, K.V., Storn, R.M., Lampinen, J.A., eds.: Differential Evolution–A Practical Approach to Global Optimization. Natural Computing Series. Springer (2005) 353–411
9. Zaharie, D.: Parameter adaptation in differential evolution by controlling the population diversity. In D. Petcu et al, ed.: Proceedings of the International Workshop on Symbolic and Numeric Algorithms for Scientific Computing. (2002) 385–397
10. Zaharie, D., Petcu, D.: Parallel implementation of multi-population differential evolution. In: Proceedings of the NATO Advanced Research Workshop on Concurrent Information Processing and Computing, IOS Press (2003) 223–232
11. Tasoulis, D.K., Pavlidis, N.G., Plagianakos, V.P., Vrahatis, M.N.: Parallel differential evolution. In: Proceedings of the IEEE Congress on Evolutionary Computation. (2004) 2023–2029
12. Kozlov, K.N., Samsonov, A.M.: New migration scheme for parallel differential evolution. In: Proceedings of the International Conference on Bioinformatics of Genome Regulation and Structure. (2006) 141–144
13. De Falco, I., Della Cioppa, A., Maisto, D., Scafuri, U., Tarantino, E.: Satellite image registration by distributed differential evolution. In: Applications of Evolutionary Computing. Volume 4448 of Lectures Notes in Computer Science. Springer (2007) 251–260
14. Apolloni, J., Leguizamón, G., García-Nieto, J., Alba, E.: Island based distributed differential evolution: An experimental study on hybrid testbeds. In: Proceedings of the IEEE International Conference on Hybrid Intelligent Systems. (2008) 696–701
15. Nipteni, M.S., Valakos, I., Nikolos, I.: An asynchronous parallel differential evolution algorithm. In: Proceedings of the ERCOFTAC Conference on Design Optimisation: Methods and Application. (2006)
16. Lampinen, J.: Differential evolution - new naturally parallel approach for engineering design optimization. In Topping, B.H., ed.: Developments in Computational Mechanics with High Performance Computing, Civil-Comp Press (1999) 217–228
17. Weber, M., Neri, F., Tirronen, V.: Distributed differential evolution with explorative-exploitative population families. Genetic Programming and Evolvable Machines **10**(4) (2009) 343–371
18. Wilcoxon, F.: Individual comparisons by ranking methods. Biometrics Bulletin **1**(6) (1945) 80–83

PV

PARALLEL DIFFERENTIAL EVOLUTION WITH ENDEMIC RANDOMIZED CONTROL PARAMETERS

by

Matthieu Weber, Ferrante Neri and Ville Tirronen 2010

In Proceedings of the Fourth International Conference on Bioinspired Optimization Methods and their Applications, pages 19–29

PARALLEL DIFFERENTIAL EVOLUTION WITH ENDEMIC RANDOMIZED CONTROL PARAMETERS

Matthieu Weber
University of Jyväskylä
matthieu.weber@jyu.fi

Ferrante Neri*
University of Jyväskylä
ferrante.neri@jyu.fi

Ville Tirronen
University of Jyväskylä
ville.tirronen@jyu.fi

Abstract This paper proposes the use of endemic control parameters within a Parallel Differential Evolution algorithm. The Differential Evolution running at each subpopulation is associated with randomly initialized scale factor and crossover rate, which are then repeatedly updated during the optimization process. Numerical results show that the Endemic Randomized Control Parameter Parallel Differential Evolution seems to be a simple, robust, and efficient algorithm suited for various applications. An important finding of this study is that randomized initial values of both control parameters and repeated updates of the scale factor are beneficial to the optimization process, whereas repeated updates of the crossover rate are detrimental.

1. Introduction

Differential Evolution (DE), see [8], is an optimization algorithm which has shown high performance in various types of applications particularly

*This work is supported by Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing.

when applied to continuous problems, for example [11]. DE is implicitly self-adaptive (see [3]), which allows it to extensively explore the problem space during the early stages of the evolution and progressively narrow the search within the most promising areas of the decision space. Although this mechanism is effective, there is a hidden drawback: the DE contains a limited amount of search moves and the population could fail at enhancing upon the available genotypes, thus resulting in a stagnation condition. In order to overcome this drawback, computer scientists in recent years have attempted to improve the DE performance by modifying the basic DE. Some popular examples of this scientific trend can be found in [9] where multiple search strategies are employed, in [2] where the offspring are generated by combining two mating pools (one global and one local, respectively), and in [1] where a randomization of the parameters increases the variability of the potential search moves.

Another popular method of enhancing the DE performance is through employment of structured populations. In [6], a distributed DE scheme employing a ring topology (the cores are interconnected in a circle and the migrations occur following the ring) has been proposed for the training of a neural network. [10] proposes a distributed DE characterized by a ring topology and the migration of individuals with the best performance to replace random individuals of the neighbor subpopulation; an application of this algorithm for training of a neural network has been presented in [7].

This paper focuses on distributed DE and in particular on the ring topology scheme presented in [10]. The main novelty in Parallel Differential Evolution (PDE) described in [10] consists of the migration scheme and the related probability: the DE is independently performed on each subpopulation composing the ring and, at the end of each generation, with a certain probability the individual with the best performance is copied into the neighbor subpopulation and replaces a randomly selected individual from the target subpopulation. In [10], a compromise value of migration probability is proposed. In this paper we propose a novel algorithm based on PDE, namely Endemic Randomized Control Parameters Parallel Differential Evolution. Instead of the fixed values of the control parameters used in PDE, ERCPDE uses pseudo-randomly generated values, inspired by the Self-Adapting Control Parameters method described in [1]. Scale factors and crossover rates are endemic (local) to each subpopulation and they are updated over time according to a probabilistic scheme.

The remainder of this article is organized in the following way. Section 2 describes the working principles of DE, PDE and ERCPDE. Sec-

tion 3 shows the experimental setup and numerical results of the present study. Section 4 gives the conclusions of this paper.

2. Endemic Randomized Control Parameters Parallel Differential Evolution

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

At the beginning of the optimization process S_{pop} individuals are pseudo-randomly sampled with a uniform distribution function within the decision space D (for simplicity, the term random will be used instead of pseudo-random in the reminder of this paper). The S_{pop} individuals constituting the populations are distributed over m subpopulations P_k, $k = 1, \ldots, m$, organized into a unidirectional ring. Each subpopulation is therefore composed of S_{pop}/m individuals.

Within each subpopulation a original DE, following its original definition, is performed. At each generation, for each individual x_i 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 as:

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

where $F \in [0, 1+]$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated. With $F \in [0, 1+]$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1, see [8]. While there is no theoretical upper limit for F, effective values are rarely greater than 1.0. The mutation scheme shown in Equation (1) is also known as DE/rand/1. It is worthwhile mentioning that there exist many other mutation variants, see [9].

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

$$x_{off,j} = \begin{cases} x'_{off,j} & \text{if } rand(0,1) \leq CR \\ x_{i,j} & \text{otherwise} \end{cases} \tag{2}$$

where $rand(0, 1)$ is a random number between 0 and 1; j is the index of the gene under examination.

The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, it replaces x_i if and only if $f(x_{off}) \leq f(x_i)$; otherwise no replacement occurs. It must be remarked that although

the replacement indexes are saved one by one during generation, actual replacements occur all at once at the end of the generation.

In PDE, each subpopulation P_k runs the DE algorithm described above. On each generation, the subpopulation has a given probability ϕ to send a copy of its best individual to its next neighbor subpopulation in the ring. When a migration occurs, the migrating individual replaces a randomly selected individual belonging to the target subpopulation, with an exception being made of the subpopulation's best performing individual, which can never be replaced. For the sake of clarity a scheme highlighting the working principles of ERCPDE is shown in Figure 1.

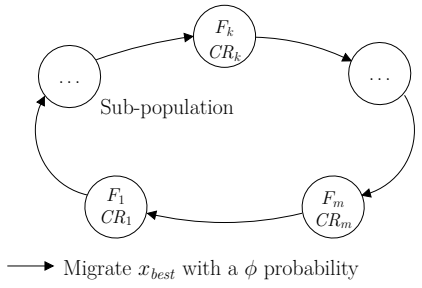


Figure 1. Working principle of the ERCPDE

In PDE, the control parameters of DE (namely the scale factor F and the crossover rate CR) are global for all the subpopulations, meaning that whichever subpopulation an individual x belongs to at a given time the same values of F and CR are used when the DE algorithm is applied to it. In ERCPDE however, each subpopulation P_k has its own scale factor F_k and crossover rate CR_k. Thus, when an individual x resides within population P_k, DE is applied to it using the local F_k and CR_k control parameters. During the initialization phase of the algorithm, the values of F_k and CR_k are randomly set in each subpopulation, such that $F_k \in [F_l, F_l + F_u]$ and $CR_k \in [0, 1]$. Additionally, the values of F_k and CR_k of each subpopulation vary in time; more precisely, on each generation F_k and CR_k have a probability to be updated of τ_1 and τ_2, respectively, see [1]:

$$F_k = \begin{cases} F_l + F_u \times rand_1, & \text{if} \quad rand_2 < \tau_1 \\ F_k, & \text{otherwise} \end{cases} \tag{3}$$

$$CR_k = \begin{cases} rand_3, & \text{if} \quad rand_4 < \tau_2 \\ CR_k, & \text{otherwise} \end{cases} \tag{4}$$

where $rand_j$, $j \in \{1, 2, 3, 4\}$, are uniform pseudo-random values between 0 and 1; τ_1 and τ_2 are constant values which represent the probabilities that parameters are updated, F_l and F_u are constant values which represent the minimum value that F could take and the maximum variable contribution to F, respectively.

To understand the rationale behind the proposed mechanism, it is important to analyze the concept of parallelism and migration in a PDE

scheme. In the classical DE, for each stage of the optimization process the algorithm can generate only a limited amount of exploratory moves. If these moves are not enough for generating new promising solutions, stagnation occurs as the algorithm does not manage to improve upon any solution of its population for a prolonged number of generations.

The use of multiple populations in distributed DE algorithms allows an observation of the decision space from various perspectives and, most importantly, decreases the risk of stagnation. In addition, the migration mechanism ensures that solutions with a high performance are introduced into the subpopulations during their evolution, which modifies the set of search moves and promote detection of new promising search directions. Thus, the migration is supposed to mitigate the risk of stagnation of the DE subpopulations and to enhance the global algorithmic performance.

Since the number of search moves allowed to an individual depends not only on other individuals in the population, but also on the values of the scale factor and crossover rate, allowing for different values of these two control parameters within the framework of a structured population such as the one used by PDE leads to a higher number of possible search moves, which increases the explorative capacity of the algorithm and leads to a quicker improvement. This paper proposes to achieve such a setup by assigning multiple values of the scale factor and the crossover rate to each subpopulation in PDE and by updating them over time so as to yet increase the number of search moves a given individual is allowed.

3. Experimental Results

The test problems listed in Table 1 have been considered in this study. The rotated version of some of the test problems listed in Table 1 have been included into the benchmark set. These rotated problems have been generated through the multiplication of the vector of variables by a randomly generated sparse orthogonal rotation matrix, created by composing n rotations of random angles (uniformly sampled in $[-\pi, \pi]$), one around each of the n axes of the search space. In total, ten test problems have been considered in this study with $n = 500$.

In order to prove the effectiveness of varying control parameters, PDE has been used as a reference algorithm and run with a value of ϕ set to 0.2 (suggested value in [10]). For the sake of comparison, the jDE algorithm described in [1] has been run on the same function, but preliminary experiments show it to be clearly inferior to PDE (as is illustrated in Figure 2 on page 9) and is therefore not included in the results presented below. ERCPDE has been run with the same value of ϕ; F_l and

Table 1. Test Problems

Test Problem	**Function**	**Decision Space**	**Optimum**
Ackley	$-20 + e + 20\exp\left(-\frac{0.2}{n}\sqrt{\sum_{i=1}^{n} x_i^2}\right)$ $-\exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi \cdot x_i)x_i\right)$	$[-1, 1]^n$	0
Alpine	$\sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1x_i \rvert$	$[-10, 10]^n$	0
Sphere	$\lVert x \rVert^2$	$[-5.12, 5.12]^n$	0
Michalewicz	$-\sum_{i=1}^{n} \sin x_i \left(\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)^{20}$	$[0, \pi]^n$	unknown
Rastrigin	$10n + \sum_{i=0}^{n}\left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12, 5.12]^n$	0
Schwefel	$-\sum_{i=1}^{n} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]^n$	$-418.9829n$

F_u were set to 0.1 and 0.9, respectively. Regarding the values of τ_1 and τ_2, two versions are presented here. The first version, indicated with ERCPDE-CR+F, has $\tau_1 = 0.1$ and $\tau_2 = 0.1$, the second indicated with ERCPDE-F has $\tau_1 = 0.1$ and $\tau_2 = 0$. In the last case, crossover rates are randomly sampled values and remain unchanged for the duration of the optimization process.

All the algorithms in this study have been run with populations of 200 individuals divided into 5 subpopulations of 40 individuals each, a setup which, according to our preliminary study, leads to the best average performance over the test functions. Each algorithm has undergone 50 independent runs for each test problem. Each single run has been performed for $500,000$ fitness evaluations. Table 2 shows the average of the final results detected by each algorithm $\pm$ the standard deviations, with the 500 dimension case. The algorithm achieving the best result for each test problem is highlighted in bold face.

In order to prove the statistical significance of the results, the Wilcoxon Rank-sum test has been applied according to the description given in [12]

Table 2. Average Fitness $\pm$ standard deviation at the end of the optimization

	PDE	ERCPDE CR+F	ERCPDE-F
Ackley	$1.62e-01 \pm 1.67e-02$	$1.52e-02 \pm 7.50e-03$	$\mathbf{6.47e-03 \pm 4.88e-03}$
Alpine	$8.88e+01 \pm 1.26e+01$	$7.05e+00 \pm 3.95e+00$	$\mathbf{1.98e+00 \pm 2.36e+00}$
DeJong	$1.92e+01 \pm 3.57e+00$	$3.37e-01 \pm 5.80e-01$	$\mathbf{8.82e-02 \pm 1.95e-01}$
Michalewicz	$-3.06e+02 \pm 5.68e+00$	$-3.11e+02 \pm 1.49e+01$	$\mathbf{-3.51e+02 \pm 3.58e+01}$
Rastrigin	$1.91e+03 \pm 9.94e+01$	$1.08e+03 \pm 1.42e+02$	$\mathbf{8.64e+02 \pm 1.38e+02}$
Schwefel	$-1.30e+05 \pm 3.17e+03$	$-1.33e+05 \pm 3.27e+03$	$\mathbf{-1.50e+05 \pm 1.14e+04}$
Rt. Ackley	$2.15e-01 \pm 2.50e-02$	$4.45e-02 \pm 9.44e-03$	$\mathbf{3.36e-02 \pm 1.03e-02}$
Rt. Michalewicz	$-1.76e+02 \pm 7.76e+00$	$-1.56e+02 \pm 7.13e+00$	$\mathbf{-1.83e+02 \pm 2.58e+01}$
Rt. Rastrigin	$1.95e+03 \pm 1.51e+02$	$1.16e+03 \pm 1.55e+02$	$\mathbf{9.61e+02 \pm 1.65e+02}$
Rt. Schwefel	$-1.65e+05 \pm 4.74e+03$	$-1.54e+05 \pm 6.07e+03$	$\mathbf{-1.66e+05 \pm 8.11e+03}$

Table 3. Results of the Wilcoxon Rank-Sum test (Comparison with ERCPDE-F)

	PDE	ERCPDE-CR+F
Ackley	+	+
Alpine	+	+
DeJong	+	+
Michalewicz	+	+
Rastrigin	+	+
Schwefel	+	+
Rt. Ackley	+	+
Rt. Michalewicz	=	+
Rt. Rastrigin	+	+
Rt. Schwefel	=	+

for a confidence level of 0.95. Table 3 shows results of the test. A "+" indicates the case in which ERCPDE-F statistically outperforms, for the corresponding test problem, the algorithm mentioned in that column and a "=" indicates that a pairwise comparison leads to success of the Wilcoxon test, i.e., the two algorithms have the same performance. The results presented in Table 3 show that ERCPDE-F outperforms PDE in eight out of the ten test problems, and has comparable results in two cases. ERCPDE-F also outperforms ERCPDE-CR+F on all ten test problems.

In order to strengthen the statistical significance of the results, the Holm procedure [5] has been applied by following the description in [4]. Considering the results in Table 2, the three algorithms under analysis have been ranked based on their average performance over the ten test problems, assigning to each algorithm a score R_i for $i = 0, \ldots, N_A - 1$ (where N_A is the number of algorithms under analysis, $N_A = 3$ in our case). With the calculated R_i values, the ERCPDE-F has been taken as the reference algorithm. The values z_i have been calculated as

$$z_i = (R_0 - R_i)/\sqrt{N_A(N_A + 1)/(6N_{TP})}$$

where R_0 is the rank of ERCPDE-F and N_{TP} is the number of test problems in consideration ($N_{TP} = 10$ in our case). The corresponding cumulative normal distribution values p_i corresponding to the z_i values have been calculated and compared with the corresponding α/i where α is the confidence threshold, set to 0.05 in our case. Table 4 displays z_i val-

Table 4. Results of the Holm procedure (Comparison with ERCPDE-F)

i	Optimizer	z	p	α/i	Hypothesis
2	PDE	-4.02e+00	2.85e-05	2.50e-02	Rejected
1	ERCPDE CR+F	-2.68e+00	3.65e-03	5.00e-02	Rejected

ues, p_i values, and corresponding α/i. Moreover, it is indicated whether the null-hypothesis (when the two algorithms have indistinguishable performances) is "Rejected" i.e., the ERCPDE-F statistically outperforms the algorithm under consideration, or "Accepted" if the distribution of values can be considered the same (no algorithm is outperformed). The Holm procedure confirms that the ERCPDE-F displays a significantly better performance with respect to the other algorithms in this study.

In order to carry out a numerical comparison of the convergence speed performance for each test problem, the average fitness values J and G returned by the best performing algorithm respectively at the beginning and at the end of the optimization process have been computed. The threshold value $THR = J - 0.95(J - G)$ has then been calculated and represents 95% of the decay in the fitness value of the best performing algorithm. If during a certain run an algorithm succeeds in reaching the value THR, the run is said to be successful. For each test problem, the average amount of fitness evaluations $\bar{ne}$ required for each algorithm to reach THR has been computed. Subsequently, the Q-test (Q stands for Quality) described in [3] has been applied. For each test problem and each algorithm, the Q measure is computed as: $Q = \bar{ne}/R$ where the robustness R is the percentage of successful runs. It is clear that, for each test problem, the smallest value equals the best performance in terms of convergence speed. The value "∞" means that $R = 0$, i.e., the algorithm never reached the THR. Table 5 shows the Q values for the ten problems; the best results are highlighted in bold face. They show that the ERCPDE-F variant of the proposed ERCPDE algorithm has the best performance in terms of convergence speed in nine cases out of the ten test problems considered. Most importantly, the ERCPDE-F algorithm, throughout all considered test problems, is never characterized by an ∞ value of Q-measure. This fact demonstrates that the proposed algorithm is always competitive with the other algorithms in the benchmark and is

Table 5. Results of the Q-test

	PDE	ERCPDE-CR+F	ERCPDE-F
Ackley	4.91e+03	2.24e+03	**1.96e+03**
Alpine	8.14e+04	2.18e+03	**2.04e+03**
DeJong	2.49e+03	**9.27e+02**	9.67e+02
Michalewicz	∞	7.00e+04	**5.69e+03**
Rastrigin	∞	3.64e+03	**2.71e+03**
Schwefel	∞	∞	**4.86e+03**
Rt. Ackley	8.39e+03	2.35e+03	**2.05e+03**
Rt. Michalewicz	7.44e+03	∞	**5.21e+03**
Rt. Rastrigin	∞	3.88e+03	**2.75e+03**
Rt. Schwefel	4.35e+03	1.23e+04	**3.48e+03**

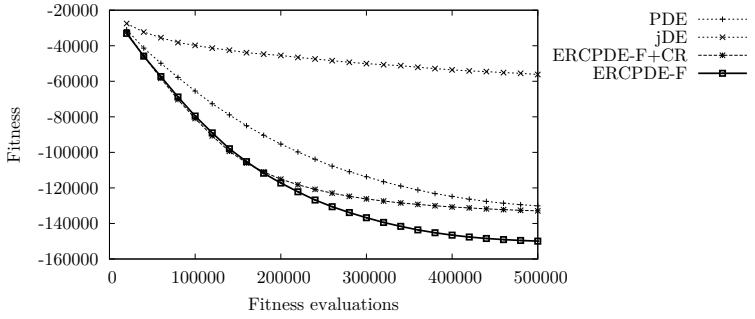


Figure 2. Performance trend (Schwefel)

never outperformed. In summary, the algorithmic behavior of ERCPDE-F is extremely promising in terms of algorithmic robustness.

Results show that endemic control parameters are an improvement over the original PDE algorithm. In addition, the fact that ERCPDE-F is superior to ERCPDE-CR+F indicates that repeated updates of the scale factor is beneficial to the performance of the algorithm, whereas the same kind of update applied to crossover rate is detrimental. In this sense, our study partly confirms and extends the self-adaptive control parameters strategy presented in [1] to PDE systems. The fact that in our case a crossover rate update is detrimental to the algorithmic performance is, according to our conjectures, explained by the fact that it appears to lead to an excessively frequent variation in the search strategy of each subpopulation, thus not allowing for an efficient exploitation of the available genotypes. Instead, as results show, it makes the algorithm too explorative, which seems to lead to stagnation (see Figure 2).

4. Conclusion

This paper proposes a novel distributed algorithm based on a parallel differential evolution scheme previously proposed in literature, namely Endemic Randomized Control parameters Parallel Differential Evolution. Each population is characterized by its own control parameter values. The individuals displaying the best performance migrate across the subpopulations, thus conforming to various search strategies. The endemic (belonging to the subpopulation) control parameters are updated over time according to probabilistic criteria. Numerical results show that the proposed algorithmic strategy leads to significant improvements in terms of algorithmic performance with respect to original Parallel Differential Evolution. In addition, the scheme employing only the update of the scale factor seems more promising with respect to the

scheme that updates both control parameters, which seems to indicate that while Parallel Differential Evolution structure requires a certain degree of randomization in order to highly enhance its performance, excessive randomization may lead to a too explorative algorithmic behavior and therefore stagnation.

References

[1] 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.

[2] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar. Differential evolution with a neighborhood-based mutation operator. *IEEE Transactions on Evolutionary Computation*, 13(3):526–553, 2009.

[3] V. Feoktistov. *Differential Evolution in Search of Solutions*, pages 83–86. Springer, 2006.

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

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

[6] W. Kwedlo and K. Bandurski. A parallel differential evolution algorithm. In *Proceedings of the IEEE International Symposium on Parallel Computing in Electrical Engineering*, pages 319–324, 2006.

[7] N. G. Pavlidis, D. K. Tasoulis, V. P. Plagianakos, G. Nikiforidis, and M. N. Vrahatis. Spiking neural network training using evolutionary algorithms. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 2190–2194, 2005.

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

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

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

[11] 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*, 16:529–555, 2008.

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

PVI

COMPONENT DECOMPOSITION IN PARALLEL DIFFERENTIAL EVOLUTION

by

Matthieu Weber, Ferrante Neri and Ville Tirronen 2010

In Proceedings of the Fourth International Conference on Bioinspired Optimization Methods and their Applications, pages 43–53

COMPONENT DECOMPOSITION IN PARALLEL DIFFERENTIAL EVOLUTION

Matthieu Weber
University of Jyväskylä
matthieu.weber@jyu.fi

Ferrante Neri*
University of Jyväskylä
ferrante.neri@jyu.fi

Ville Tirronen
University of Jyväskylä
ville.tirronen@jyu.fi

Abstract This paper proposes decomposing the search space of large-scale problems into lower-dimensionality subspaces, and associating each of these to one subpopulation of a Parallel Differential Evolution algorithm. Each subpopulation is running a modified Differential Evolution algorithm, where the crossover function is limited to components of the subpopulation's associated subspace. According to numerical results the Parallel Component Decomposition Differential Evolution seems to be a clear improvement over the original Parallel Distributed Evolution, making it a simple, robust, and efficient algorithm suited for various applications.

1. Introduction

Differential Evolution (DE), see [7], has shown high performance in various types of optimization problems, and more particularly in continuous problems, for example [10]. Like all optimization algorithms, DE suffers from the so-called "curse of dimensionality", which refers to the

*This work is supported by Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing.

fact that the complexity of a multidimensional problem increases with its dimensionality in an exponential fashion. In [12] we have shown that the structured population characterizing the Parallel Differential Evolution (PDE) described in [9] is one enhancement that allows to break this curse: instead of running the DE over a single population, PDE runs multiple DE algorithms over multiple subpopulations which are organized in a unidirectional ring; with a given probability, the best individual of a subpopulation is migrated to the next subpopulation in the ring, where it replaces a random individual. In order to enhance PDE, [12] then proposes to use two families of subpopulations, the first one running a regular PDE while the second runs independent instances of the population-size reduction DE proposed in [2]; after an observation period, individuals from the second family are injected into the subpopulations of the first family.

The randomization of DE's control parameter during the course of the optimization process, such as in [1], is another strategy which allows to improve the performance of the algorithm. In [13], we propose to blend the randomization of the control parameters with the migration mechanism of the PDE, resulting in a variant of the latter where the scale factor used by the DE at one given subpopulation is migrated (with a small perturbation) along with the best individual.

The rationale behind these variants of PDE and DE lies in the fact that DE contains a limited amount of search moves and the population could fail at enhancing upon the available genotypes thus resulting in stagnation. The variation over time of the control parameters or the injection of independently optimized individuals increases the number of available search moves, thus allowing for a more exhaustive exploration of the problem space.

Yet another possibility of enhancing the DE is to subdivide the search space into subspaces and optimize only the components of the solution that belong to one subspace, while keeping the others constant. This approach, which effectively breaks the curse of dimensionality by locally reducing the dimensionality of the problem, was first proposed in [6]. Here each component of the solution was optimized separately by a dedicated subpopulation, while the fitness of one solution was evaluated over the whole set of components by taking random individuals from the other subpopulations in order to reconstruct a whole solution; this process was called cooperative co-evolution. This decomposition scheme is, however, reputed to be inefficient on non-separable functions (see [6, 15]) i.e., where the variables of the problem interact with each other. To overcome this problem, [17] uses randomly selected sub-components which change over time, [15] proposes using a set of weights, itself evolved us-

ing DE, to select the sub-components, [16] makes use of the DE with neighborhood search, while [11] selects components showing the highest variance across the population as the ones to be optimized at a given time. The three above cited articles made use of non-standard DE algorithms in order to yet increase their efficiency.

The algorithm proposed in this paper, the Parallel Component Decomposition Differential Evolution (PaCoDDE) borrows the structured population and best individual migration from the Parallel Differential Evolution (PDE) described in [9] and merges it with a static subcomponent decomposition: the search space is decomposed into subspaces of near-equal dimensionality, and each subspace is assigned to one subpopulation. The subpopulations focus on optimizing their solutions within their own subspaces, while keeping the rest of the components constant. On every generation, each subpopulation has a given probability of migrating its best-performing individual to the next subpopulation in the ring, allowing to the optimized sub-components to incrementally propagate the other subpopulations.

The remainder of this article is organized in the following way. Section 2 describes the working principles of DE, PDE and PaCoDDE. Section 3 shows the experimental setup and numerical results of the present study. Section 4 gives the conclusions of this paper.

2. Parallel Component Decomposition Differential Evolution

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

At the beginning of the optimization process, S_{pop} individuals are pseudo-randomly sampled with a uniform distribution function within the decision space D (for simplicity, the term random will be used instead of pseudo-random in the reminder of this paper). The S_{pop} individuals constituting the populations are distributed over m subpopulations $P_k, k = 1, \ldots, m$ arranged in a unidirectional ring. Each subpopulation is therefore composed of S_{pop}/m individuals. Additionally, the set of dimensions of the search space D is decomposed into m subsets C_k of approximately equal sizes n_k, with the constraints that $\sum_{k=1}^{m} n_k = n$ and $\forall (k_1, k_2) \in \{1, \ldots, m\}^2, k_1 \neq k_2, C_{k_1} \cap C_{k_2} = \emptyset$. In other words, C_k represents the sub-component of x that is to be optimized by subpopulation P_k.

Within each subpopulation P_k, a modified DE is performed. At each generation, for each individual x_i of P_k, three individuals x_r, x_s and x_t

are randomly extracted from the subpopulation. According to the DE logic, a provisional offspring x'_{off} is generated by mutation as:

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

where $F \in [0, 1+]$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated. With $F \in [0, 1+]$, it is meant here that the scale factor should be a positive value which cannot be much greater than 1, see [7]. The mutation scheme shown in Equation (1) is also known as DE/rand/1. It is worthwhile mentioning that there exist many other mutation variants, see [8].

When the provisional offspring has been generated by mutation, each gene $x_{i,j}, j \in C_k$ of x_i is exchanged with the corresponding gene of x'_{off} with a uniform probability and the final offspring x_{off} is generated:

$$x_{off,j} = \begin{cases} x'_{off,j} & \text{if} \quad j \in C_k \text{ and } rand\,(0,1) \leq CR \\ x_{i,j} & \text{otherwise} \end{cases} \tag{2}$$

where $rand\,(0,1)$ is a random number between 0 and 1; $j = 1, \ldots, n$ is the index of the gene under examination. This modified crossover function always keeps the parent's genes which are not part of the considered sub-component, and actually crosses over only the genes of the offspring and of the parent which are part of the sub-component.

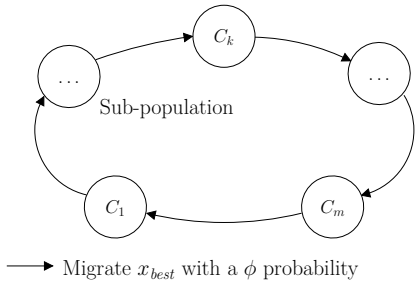


Figure 1. Working principle of the PaCoDDE

The resulting offspring x_{off} is evaluated and, according to a one-to-one spawning strategy, replaces x_i if and only if $f(x_{off}) \leq f(x_i)$; otherwise no replacement occurs. It must be remarked that although the replacement indexes are saved one by one during generation, actual replacements occur all at once at the end of the generation.

In PDE, each subpopulations P_k runs a regular DE algorithm, which is replaced in PaCoDDE by the modified DE algorithm described above. On each generation, the subpopulation has a given probability ϕ to send a copy of its best individual to its next neighbor subpopulation in the ring. When a migration occurs, the migrating individual replaces a randomly selected individual belonging to the target subpopulation, with an exception being made of the subpopulation's best performing individual, which can never be replaced. For the sake of clarity, a schema highlighting the working principles of PaCoDDE is shown in Figure 1.

Table 1. Test Problems

Test Problem	**Function**	**Decision Space**	**Optimum**
Ackley	$-20 + e + 20 \exp\left(-\frac{0.2}{n}\sqrt{\sum_{i=1}^{n} x_i^2}\right)$ $- \exp\left(\frac{1}{n}\sum_{i=1}^{n} \cos(2\pi \cdot x_i) x_i\right)$	$[-1, 1]^n$	0
Alpine	$\sum_{i=1}^{n} \lvert x_i \sin x_i + 0.1 x_i \rvert$	$[-10, 10]^n$	0
Sphere	$\lVert x \rVert^2$	$[-5.12, 5.12]^n$	0
Michalewicz	$-\sum_{i=1}^{n} \sin x_i \left(\sin\left(\frac{i \cdot x_i^2}{\pi}\right)\right)^{20}$	$[0, \pi]^n$	unknown
Rastrigin	$10n + \sum_{i=0}^{n} \left(x_i^2 - 10\cos(2\pi x_i)\right)$	$[-5.12, 5.12]^n$	0
Schwefel	$-\sum_{i=1}^{n} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$	$[-500, 500]^n$	$-418.9829n$

Although PaCoDDE may seem crude due to the fact that it makes use of a static decomposition of the search space while the modern above-mentioned algorithms are based on random or dynamic decomposition schemes, its specificity lies in that the individuals showing the best performance are traveling between subpopulations during the run of the algorithm, being optimized incrementally over each of their subcomponents, and revisiting each subpopulation multiple times. The migration rate must therefore be high enough to ensure that these individuals are not allowed to become excessively specialized within one subspace while being far from optimal when considering the whole search space.

3. Experimental Results

The test problems listed in Table 1 have been considered in this study. The rotated version of some of them have been included into the benchmark set. These rotated problems have been generated through the multiplication of the vector of variables by a randomly generated sparse orthogonal rotation matrix, created by composing n rotations of random angles (uniformly sampled in $[-\pi, \pi]$), one around each of the n axes of the search space. While the six unrotated functions are separable, the rotated versions are not. In total, ten test problems have been considered in this study with $n = 500$.

To prove the effectiveness of component decomposition within the framework of a PDE algorithm, PaCoDDE has been compared to the original PDE and to the DEwSAcc described in [17]. In both PDE and PaCoDDE the control parameters of DE, namely the scale factor F and the crossover rate CR have been set to 0.7 and 0.3 respectively, in accordance with the suggestions given in [19] and [18]. The migration rate ϕ

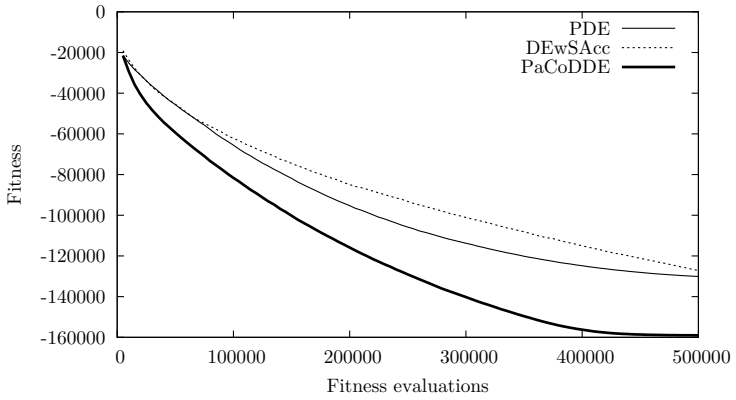


Figure 2. Example of a performance trend (Schwefel)

was set to 0.2, as suggested in [9]. The τ parameter of DEwSAcc was set to $0.2\sqrt{2}/\sqrt{500}$, as recommended in [17] for 500-dimensional problems. It has to be mentioned that DEwSAcc, in addition to search space decomposition, uses multiple mutation schemes and self-adaptive control parameters, a variation on the original DE which in itself improves performance of the algorithm compared to the original DE. PaCoDDE, on the contrary, uses a single mutation scheme and static control parameter values, as does PDE.

All the algorithms in this study have been run with populations of 200 individuals. In PDE and PaCoDDE, these individuals are organized into 5 subpopulations of 40 individuals, a setup which, according to our preliminary study, leads to the best average performance over the test functions. For DEwSAcc, a single population of 200 individuals is used. The search space was decomposed as follows. The dimensions of the problems, indexed from 1 to 500, were split into 5 intervals $C_k = \{100k - 99, \ldots, 100k\}$ for $k = 1, \ldots, 5$. Each interval C_k was then assigned to subpopulation P_k. Each algorithm has undergone 50 independent runs for each test problem. Each single run has been performed for $500,000$ fitness evaluations. Table 2 shows the average of the final results detected by each algorithm $\pm$ the standard deviations. The algorithm achieving the best result for each test problem is highlighted in bold face. An example of the performance trend is shown in Figure 2.

To prove the statistical significance of the results, the Wilcoxon Rank-sum test has been applied according to the description given in [14] for a confidence level of 0.95. Table 3a shows results of the test. A "+" indicates a case where PaCoDDE statistically outperforms, for the corresponding test problem, the algorithm mentioned in that column, a

Table 2. Average Fitness ± standard deviation at the end of the optimization process

	PDE	DEwSAcc	PaCoDDE
Ackley	$1.62e-01 \pm 1.67e-02$	$\mathbf{4.22e-02 \pm 9.85e-03}$	$5.44e-02 \pm 5.80e-03$
Alpine	$8.88e+01 \pm 1.26e+01$	$7.07e+01 \pm 1.74e+01$	$\mathbf{1.97e+01 \pm 3.05e+00}$
Sphere	$1.92e+01 \pm 3.57e+00$	$\mathbf{1.51e+00 \pm 5.68e-01}$	$2.42e+00 \pm 4.55e-01$
Michalewicz	$-3.06e+02 \pm 5.68e+00$	$-2.52e+02 \pm 1.04e+01$	$\mathbf{-4.18e+02 \pm 3.18e+00}$
Rastrigin	$1.91e+03 \pm 9.94e+01$	$1.57e+03 \pm 2.24e+02$	$\mathbf{7.84e+02 \pm 4.48e+01}$
Schwefel	$-1.30e+05 \pm 3.17e+03$	$-1.27e+05 \pm 6.91e+03$	$\mathbf{-1.59e+05 \pm 2.12e+03}$
Rt. Ackley	$2.15e-01 \pm 2.50e-02$	$\mathbf{8.36e-02 \pm 1.26e-02}$	$1.41e-01 \pm 2.10e-02$
Rt. Michalewicz	$-1.76e+02 \pm 7.76e+00$	$-1.48e+02 \pm 4.90e+00$	$\mathbf{-2.55e+02 \pm 4.93e+00}$
Rt. Rastrigin	$1.95e+03 \pm 1.51e+02$	$2.50e+03 \pm 2.32e+02$	$\mathbf{1.19e+03 \pm 5.06e+01}$
Rt. Schwefel	$-1.65e+05 \pm 4.74e+03$	$-1.25e+05 \pm 4.54e+03$	$\mathbf{-1.90e+05 \pm 2.81e+03}$

"−" indicates that PaCoDDE is outperformed by the algorithm it is compared to, and a "=" indicates that a pairwise comparison leads to success of the Wilcoxon test i.e., that the two algorithms have the same performance. The results presented in Table 3a show that PaCoDDE outperforms PDE for all ten test problems. It moreover outperforms DEwSAcc in seven cases out of the ten.

To strengthen the statistical significance of the results, the Holm procedure [5] has been applied by following the description in [4]. Considering the results in Table 2, the three algorithms under analysis have been ranked on the basis of their average performance calculated over the ten test problems, assigning to each algorithm a score R_i for $i = 0, \ldots, N_A - 1$ (where N_A is the number of algorithms under analysis, $N_A = 3$ in our case). With the calculated R_i values, the PaCoDDE has been taken as the reference algorithm. The values z_i have then been calculated as

$$z_i = (R_0 - R_i)/\sqrt{N_A(N_A+1)/(6N_{TP})}$$

where R_0 is the rank of PaCoDDE and N_{TP} is the number of test problems in consideration ($N_{TP} = 10$ in our case). The cumulative normal distribution values p_i corresponding to the z_i values have then been calculated and compared with the corresponding α/i values where α is the confidence threshold, set to 0.05 in our case. Table 3b displays z_i values, p_i values, and corresponding α/i. Moreover, it is indicated whether the null-hypothesis (when the two algorithms have indistinguishable performances) is "Rejected" i.e., the PaCoDDE statistically outperforms the algorithm under consideration, or "Accepted" if the distribution of values can be considered the same (no algorithm is outperformed). The Holm procedure thus confirms that the PaCoDDE performs significantly better than the other algorithms in this study.

To carry out a numerical comparison of the convergence speed performance for each test problem, the average fitness values J and G returned

Table 3a. Wilcoxon Rank-Sum test (Comparison with PaCoDDE)

	PDE	DEwSAcc
Ackley	+	-
Alpine	+	+
Sphere	+	-
Michalewicz	+	+
Rastrigin	+	+
Schwefel	+	+
Rt. Ackley	+	-
Rt. Michalewicz	+	+
Rt. Rastrigin	+	+
Rt. Schwefel	+	+

Table 3b. Results of the Holm procedure (Comparison with PaCoDDE)

i	2	1
Optimizer	PDE	DEwSAcc
z	-2.68e+00	-2.01e+00
p	3.65e-03	2.21e-02
α/i	2.50e-02	5.00e-02
Hypothesis	Rejected	Rejected

Table 4. Results of the Q-test

	PDE	DEwSAcc	PaCoDDE
Ackley	4.41e+03	**2.96e+03**	3.42e+03
Alpine	9.98e+03	5.35e+03	**3.61e+03**
Sphere	2.38e+03	**1.42e+03**	2.06e+03
Michalewicz	∞	∞	**4.41e+03**
Rastrigin	∞	2.48e+05	**3.92e+03**
Schwefel	∞	∞	**3.62e+03**
Rt. Ackley	4.88e+03	**3.03e+03**	4.01e+03
Rt. Michalewicz	∞	∞	**4.18e+03**
Rt. Rastrigin	∞	2.47e+05	**3.94e+03**
Rt. Schwefel	∞	∞	**3.33e+03**

by the best performing algorithm respectively at the beginning and at the end of the optimization process have been computed. The threshold value $THR = J - 0.95(J - G)$ has then been calculated and represents 95% of the decay in the fitness value of the best performing algorithm. If during a certain run an algorithm succeeds in reaching the value THR, the run is said to be successful. For each test problem, the average amount of fitness evaluations $\bar{ne}$ required for each algorithm to reach THR has been computed. Subsequently, the Q-test (Q stands for Quality) described in [3] has been applied. For each test problem and each algorithm, the Q measure is computed as: $Q = \bar{ne}/R$ where the robustness R is the percentage of successful runs. It is clear that, for each test problem, the smallest value equals the best performance in terms of convergence speed. The value "∞" means that $R = 0$, i.e., the algorithm never reached the THR. Table 4 shows the Q values for the ten problems and the best results are highlighted in bold face. These show that the PaCoDDE has the best performance in terms of convergence speed in seven cases out of the ten test problems considered. Most importantly, the PaCoDDE algorithm, throughout all considered test problems, is never characterized by an ∞ value of the Q-measure, which

demonstrates that the proposed algorithm is always competitive with the other algorithms in the benchmark and is never outperformed. In summary, the algorithmic behavior of PaCoDDE is extremely promising in terms of algorithmic robustness.

Results show that sub-component decomposition of the search space applied to PDE is an improvement over the original algorithm. This confirms the results presented in [6] about search space decomposition and sub-component optimization, and extends them to structured-population DE schemes.

4. Conclusion

This paper proposes an improved variant of the Parallel Differential Evolution for high-dimensionality problems, namely Parallel Component Decomposition Differential Evolution. The search space of the problem is decomposed into disjoint subspaces, each of which is associated to one subpopulation of a Parallel Differential Evolution. Each subpopulation runs a modified Differential Evolution algorithm, where the crossover function limits its action on the considered individual to the components belonging to the associated subspace. Individuals displaying the best performance are then given the possibility to migrate to the neighboring subpopulation, where another of their sub-component is optimized. Numerical results show that the proposed algorithmic logic leads to significant improvements in terms of algorithmic performance with respect to standard Parallel Differential Evolution. Despite the fact that our algorithm employs a single mutation scheme, static control parameter values and a static sub-component decomposition, on average it outperforms the DEwSAcc algorithm which employs multiple mutation schemes, self-adaptive control parameters and a random, dynamic decomposition scheme. The integration of such features into a Parallel Differential Evolution framework seems to be a promising research path.

References

[1] 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.

[2] J. Brest and M. S. Maučec. Population size reduction for the differential evolution algorithm. *Applied Intelligence*, 29(3):228–247, 2008.

[3] V. Feoktistov. *Differential Evolution in Search of Solutions*, pages 83–86. Springer, 2006.

[4] S. Garca, A. Fernndez, J. Luengo, and F. Herrera. A study of statistical techniques and performance measures for genetics-based machine learning: accuracy

and interpretability. *Soft Computing*, 13(10):959–977, 2008.

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

[6] M. A. Potter and K. A. De Jong. A cooperative coevolutionary approach to function optimization. In *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pages 249–257. Springer-Verlag, 1994.

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

[8] 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*, 13:398–417, 2009.

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

[10] 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*, 16:529–555, 2008.

[11] Y. Wang, B. Li, and X. Lai. Variance priority based cooperative co-evolution differential evolution for large scale global optimization. In *Evolutionary Computation, 2009. CEC '09. IEEE Congress on*, pages 1232–1239, May 2009.

[12] M. Weber, F. Neri, and V. Tirronen. Distributed differential evolution with explorative-exploitative population families. *Genetic Programming and Evolvable Machines*, 10(4):343–371, 2009.

[13] M. Weber, F. Neri, and V. Tirronen. Scale factor inheritance mechanism in distributed differential evolution. *Soft Computing*, 2009.

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

[15] Z. Yang, K. Tang, and X. Yao. Differential evolution for high-dimensional function optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 3523–3530, 2007.

[16] Z. Yang, K. Tang, and X. Yao. Large scale evolutionary optimization using cooperative coevolution. *Information Sciences*, 178(15):2985–2999, 2008.

[17] 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*, pages 3719–3726, 2008.

[18] K. Zielinski and R. Laur. Stopping criteria for differential evolution in constrained single-objective optimization. In U. K. Chakraborty, editor, *Advances in Differential Evolution*, volume 143 of *Studies in Computational Intelligence*, pages 111–138. Springer, 2008.

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

PVII

SHUFFLE OR UPDATE PARALLEL DIFFERENTIAL EVOLUTION FOR LARGE SCALE OPTIMIZATION

by

Matthieu Weber, Ferrante Neri and Ville Tirronen 2010

In Special Issue of Soft Computing on Scalability of Evolutionary Algorithms and other Metaheuristics for Large Scale Continuous Optimization Problems, to appear

Soft Computing manuscript No.
(will be inserted by the editor)

Matthieu Weber · Ferrante Neri · Ville Tirronen

Shuffle Or Update Parallel Differential Evolution for Large Scale Optimization

Received: date / Accepted: date

Abstract This paper proposes a novel algorithm for large scale optimization problems. The proposed algorithm, namely Shuffle Or Update Parallel Differential Evolution (SOUPDE) is a structured population algorithm characterized by sub-populations employing a Differential Evolution logic. The sub-populations quickly exploit some areas of the decision space, thus drastically and quickly reducing the fitness value in the highly multi-variate fitness landscape. New search logics are introduced into the sub-population functioning in order to avoid a diversity loss and thus premature convergence. Two simple mechanisms have been integrated in order to pursue this aim. The first, namely shuffling, consists of randomly rearranging the individuals over the sub-populations. The second consists of updating all the scale factors of the sub-populations.

This research is supported by the Academy of Finland, Akatemiatutkija 130600, Algorithmic Design Issues in Memetic Computing.

Matthieu Weber
Tel.: +358-14-2603056
E-mail: matthieu.weber@jyu.fi

Ferrante Neri
Tel.: +358-14-2602764
E-mail: ferrante.neri@jyu.fi

Ville Tirronen
Tel.: +358-14-2604987
E-mail: ville.tirronen@jyu.fi

University of Jyväskylä
Department of Mathematical Information Technology
P.O. Box 35 (Agora)
40014 University of Jyväskylä
Finland
Fax: +358-14-2604981

The proposed algorithm has been run on a set of various test problems for five levels of dimensionality and then compared to three popular meta-heuristics. Rigorous statistical and scalability analyses are reported in this article. Numerical results show that the proposed approach significantly outperforms the meta-heuristics considered in the benchmark and has a good performance despite the high dimensionality of the problems.

The proposed algorithm has a good balance between exploitation and exploration and succeeds to have a good performance over the various dimensionality values and test problems present in the benchmark. It succeeds at outperforming the reference algorithms considered in this study. In addition, the scalability analysis proves that with respect to a standard Differential Evolution, the proposed SOUPDE algorithm enhances its performance while the dimensionality grows.

Keywords Differential Evolution · Distributed Algorithms · Large Scale Optimization · Randomization · Scale Factor Update · Shuffling Mechanism

1 Introduction

According to the common sense, for a given problem characterized by dimensionality n, an increase in the dimensionality results into an increase of the problem difficulty. Clearly, this increase is not linear with respect to the dimensionality but follows an exponential rule. In other words, if we double the dimensionality of a problem we do not just double its difficulty. The problem difficulty would result increased by many times. Without getting into a mathematical proof, we can consider an optimization problem characterized by a flat fitness landscape and a small basin of attraction within a hypercube whose side is $\frac{1}{100}$ of the side of the entire domain. Under these conditions, we can consider this optimization problem to be solved when the optimization algorithm employed samples at least one point within the basin of attraction. If $n = 1$, the basin of attraction is $\frac{1}{100}$ of the search domain, i.e. on average, a simple random search algorithm would need to sample 100 points before detecting the basin of attraction. This means that the problem for $n = 1$ is fairly easy. If $n = 2$, a random search algorithm would need 10000 points before solving the problem. If $n = 100$, it can easily be checked that a random search would need 10^{200} points to solve the problem. Thus, for $n = 100$, the same problem, which is easy for lower dimensionality, becomes extremely hard.

Although trivial, this example can be enlightening in order to understand the difficulty of an optimization algorithms for highly dimensional problems and the so called "curse of dimensionality", i.e. the deterioration in the algorithmic performance, as the dimensionality of the search space increases, see [1]. However, the design of algorithms capable to handle highly multivariate fitness landscapes, also known as Large Scale Optimization Problems (LSOP), is crucially important for many real-world optimization problems. For example, in structural optimization an accurate description of complex spatial objects might require the formulation of a LSOP; similarly such a

situation also occurs in scheduling problems, see [2]. Another important example of a class of real-world LSOPs is the inverse problem chemical kinetics studied in [3] and [4]. Unfortunately, when an exact method cannot be applied, LSOPs can turn out to be very difficult to solve. As a matter of fact, due to high dimensionality, algorithms which perform a neighborhood search (e.g. Hooke-Jeeves Algorithm) might require an unreasonably high number of fitness evaluations at each step of the search while population based algorithms are likely to either prematurely converge to suboptimal solutions, or stagnate due to an inability to generate new promising search directions. In other words, many metaheuristics (even the most modern and sophisticated) that perform well for problems characterized by a low dimensionality can often fail to find good near optimal solutions to high-dimensional problems.

Since the employment of optimization algorithms can lead to a prohibitively high computational cost of the optimization run without the detection of a satisfactory result, it is crucially important to detect an algorithmic solution that allows good results by performing a relatively low amount of objective function evaluations. In the literature various studies have been carried out and several algorithmic solutions have been proposed. In [5], a modified Ant Colony Optimizer (ACO) has been proposed for large scale optimization problems. Some other papers propose a technique, namely cooperative coevolution, originally defined in [6] and subsequently developed in other works, see e.g. [7] and [8]. The concept of the cooperative coevolution is to decompose a LSOPs in a set of low dimension problems which can be separately solved and then recombined in order to compose the solution of the original problem. It is obvious that if the objective function (fitness function) is separable then the problem decomposition can be trivial while for nonseparable functions the problem decomposition can turn out to be a very difficult task. However, some techniques for performing the decomposition of nonseparable functions have been developed, see [9]. Recently, cooperative coevolution procedures have been successfully integrated within Differential Evolution (DE) frameworks for solving LSPs, see e.g. [10], [11], [12], [13] and [14].

It should be remarked that a standard DE can be inefficient for solving LSOPs, see [15] and [16]. However, DE framework, thanks to its simple structure and flexibility, can be easily modified and become an efficient solver of high dimensional problems. Besides the examples of DE integrating cooperative coevolution, other DE based algorithms for LSPs have been proposed. In [17] the opposition based technique is proposed for handling the high dimensionality. This technique consists of generating extra points, that are symmetric to those belonging to the original population, see details in [18]. In [19] a Memetic Algorithm (MA) (see for the definitions e.g. [20], [21], and [22]) which integrates a simplex crossover within the DE framework has been proposed in order to solve LSOPs, see also [23]. In [24], on the basis of the studies carried out in [25], [26], and [27], a DE for LSOPs has proposed. The algorithm proposed in [24] performs a probabilistic update of the control parameter of DE variation operators and a progressive size reduction of the population size. Although the theoretical justifications of the success of this algorithm are not fully clear, the proposed approach seems to be ex-

tremely promising for various problems. In [28], a memetic algorithm which hybridizes the self-adaptive DE described in [27] and a local search applied to the scale factor in order to generate candidate solutions with a high performance has been proposed. Since the local search on the scale factor (or scale factor local search) is independent on the dimensionality of the problem, the resulting memetic algorithm offered a good performance for relatively large scale problems, see [28].

Yet another possibility of enhancing the DE is to employ structured populations, i.e. to subdivide the population into subpopulation which evolve independently and somehow exchange pieces of information for performing the global search, see [29] and [30]. While the small subpopulations are efficient at quickly improving upon their fitness values, might likely dissipate the initial diversity thus resulting into a premature convergence, see [31] and [32].

This paper aims at overcoming this inconvenience while performing the optimization by means of a structured population DE by integrating two simple randomized components. These components are supposed to introduce a certain degree of randomization within the DE logic which proved to be beneficial, see [16], and, at the same time, to perform a refreshment of the search logic which allows the search to continue towards the global optimum despite the high dimensionality. The proposed algorithm, namely Shuffle Or Update Parallel Differential Evolution (SOUPDE) is composed on a structured population (parallel) DE/rand/1/exp which integrates two extra mechanism activated by means of a probabilistic criterion. The first mechanism, namely shuffling, consists of merging the sub-populations and subsequently randomly dividing the individuals again into sub-populations. The second mechanism, namely update, consists of randomly updating the values of the scale factors of each population.

The remainder of this article is organized in the following way. Section 2 describes the algorithmic components characterizing the proposed algorithm. Section 3 shows the numerical results and highlights the performance and scalability of the SOUPDE. Section 4 gives the conclusion of this work.

2 Shuffle Or Update Parallel Differential Evolution

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

At the beginning of the optimization process, S_{pop} individuals are pseudo-randomly sampled within the decision space D. These S_{pop} individuals are distributed over m sub-populations $P_k, k = 1, \ldots, m$; each sub-population is composed of $\frac{S_{pop}}{m}$ individuals. For each sub-population P_k a scale factor F^k, for $k = 1, \ldots, m$, is assigned. Each scale factor is initially generated as pseudo-random by sampling a value from a uniform distribution between 0.1 and 1.

Within each sub-population k, a DE/rand/1/exp is performed. The mutation procedure is implemented in the following way. For each individual

```
x_off ← x_i
generate j ← 1 + round(n × rand(0, 1))
x_off(j) ← x'_off(j)
p ← 0
while rand(0, 1) ≤ Cr AND p < n − 1 do
    x_off(1 + (j + p) mod n) ← x'_off(1 + (j + p) mod n)
    p ← p + 1
    generate rand(0, 1)
end while
```

Fig. 1: Exponential crossover pseudo-code

x_i of the $\frac{S_{pop}}{m}$ 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 as:

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

where usually $F^k \in [0, 2]$ is a scale factor which controls the length of the exploration vector $(x_r - x_s)$ and thus determines how far from point x_i the offspring should be generated.

When the provisional offspring has been generated by mutation, the exponential crossover is applied. 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 $rand(0, 1) \leq Cr$, 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 $rand(0, 1) > Cr$ the copy process is interrupted. Thus, all the remaining design variables of the offspring are copied from the parent. For the sake of clarity the pseudo-code of the exponential crossover is shown in Fig. 1

With a probability p_s the shuffling operation is performed. The solutions distributed over the m sub-populations are moved into a provisional list. Then the solutions are randomly re-distributed over the m sub-populations. Fig. 2 displays a graphical representation of the shuffling mechanism.

With a probability p_u the update operation is performed. The set of scale factors F^k, for $k = 1, \ldots, m$ is replaced by newly generated random numbers, sampled between 0.1 and 1.

The described operations are repeated until the budget conditions are satisfied.

2.1 Algorithmic Philosophy

This section summarizes the considerations regarding the functioning of the SOUPDE. In order to understand the working principle of the proposed algorithm it is necessary to make some considerations about the DE structure. As highlighted in [33], the success of DE is due to an implicit self-adaptation contained within the algorithmic structure. Since, for each candidate solution,

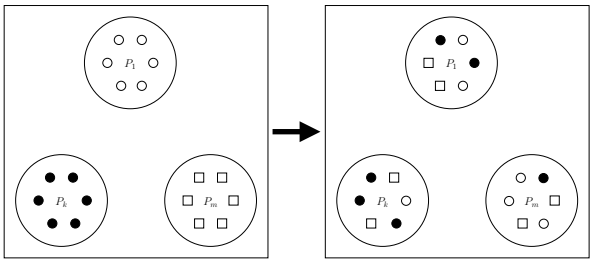


Fig. 2: Shuffling mechanism

the search rule depends on other solutions belonging to the population (e.g. x_t, x_r, and x_s), the capability of detecting new promising offspring solutions depends on the current distribution of the solutions within the decision space. During early stages of the optimization process, solutions tend to be spread out within the decision space. For a given scale factor value, this implies that the mutation appears to generate new solutions by exploring the space by means of a large step size (if x_r and x_s are distant solutions, $F^k\,(x_r - x_s)$ is a vector characterized by a large modulus). During the optimization process, the solutions of the population tend to concentrate on specific parts of the decision space. Therefore, step size in the mutation is progressively reduced and the search is performed in the neighborhood of the solutions. In other words, due to its structure, a DE scheme is highly explorative at the beginning of the evolution and subsequently becomes more exploitative during fine tuning.

However, as highlighted in [16], DE mechanism hides a limitation. If for some reasons, the algorithm does not succeed in generating offspring solutions which outperform the corresponding parent, the search is repeated again with similar step size values and will likely fail by falling into an undesired stagnation condition (see [34]). This effect can become frequent in LSOPs and often jeopardize the entire algorithmic functioning. According the analysis presented in [16], the DE incorrect functioning can be due to a limited amount of search moves and therefore recently proposed successful DE variants employ extra moves within the search logic. An efficient way to increase the amount of search moves is to introduce a certain degree of randomization as in the case of [27]. The use of multiple populations in distributed DE algorithms allows an observation of the decision space from various perspectives thus decreasing the risk of stagnation since each sub-population imposes a high exploitation pressure. The fact that focusing of the search is beneficial in the presence a highly dimensional problems has been proposed in [25] and [24] where the population size reduction allows faster improvements and in [31] where the search is distributed over explorative and exploitative subpopulations. On the other hand, the high exploitation imposed by the small size of the sub-populations leads to quick improvements in the early stages

of the optimization process and thus quickly dissipate the diversity and result into a premature convergence. This effect is amplified in LSOPs where a distributed DE easily succeed at improving the initial solutions but likely fail at detecting the global optimum or solutions characterized by a high performance.

In order to correct this drawback the proposed SOUPDE employs shuffling and update mechanisms. The idea behind this algorithmic components is simple and efficient. Since, within each sub-population, the incapability of enhancing upon the best individuals is due to an excessive exploitation of the search moves, a new set of search moves in necessary in order to continue the optimization, see [16] and [32]. The search moves in a DE scheme are due to the solutions composing the population and the scale factor values. The shuffling operation described above is equivalent to modifying the set of search moves. The new search moves promote the detection of new promising search directions and thus allow the DE search structure to be periodically "refurbished". Thus, shuffling is supposed to mitigate the risk of DE (sub-)populations losing the diversity and to enhance the global algorithmic performance. Finally, the scale factor update is supposed to give an alternative "refurbishment" strategy with respect to the shuffling and attempt to detect new areas of the search space.

The proposed SOUPDE algorithm can be seen as a multiple mini-search, where each search algorithm is a sub-population performing a DE search. This structure integrates, in a memetic computing fashion, alternative randomized search logics which lead to a restart of the mini-search while still keeping in the memory the previous achieved enhancements. Thus, the subpopulation are supposed to highly exploit their set of search moves until their action becomes ineffective. The refurbishment operation then delivers to each sob-population a new set of search moves to be subsequently exploited.

3 Numerical Results

The proposed SOUPDE algorithm has been run with $m = 3$ sub-populations composed of $\frac{S_{pop}}{m} = 20$ individuals each. The scale factors F^k are randomly generated and updated in the $[0.1, 1]$ interval using a uniform distribution, while the crossover rate Cr has been set equal to 0.9. Both the shuffling and update probabilities p_s and p_u have been set equal to 0.01. The test problems and the other parameters have been set according to the rules given for this special issue.

3.1 Parameter Settings

Preliminary tests have shown that setting the shuffling and update probabilities p_s and p_u to 0.01 have produced reasonably good results. In order to determinate if better values can be chosen, multiple tests have been run with values of p_s and p_u set to 0, 0.001, 0.01, 0.1 and 0.5. The set of values tested is limited to the above due to the enormous amount of computing time

Table 1: Holm test for parameter setting in 50 dimensions (reference: $p_s = 0.1$, $p_u = 0.5$, estimator = average)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0.001	-6.50e+00	3.96e-11	2.08e-03	Rejected
23	0	0.01	-6.33e+00	1.26e-10	2.17e-03	Rejected
22	0	0	-6.26e+00	1.93e-10	2.27e-03	Rejected
21	0.001	0	-5.16e+00	1.25e-07	2.38e-03	Rejected
20	0.01	0	-4.92e+00	4.43e-07	2.50e-03	Rejected
19	0	0.1	-4.67e+00	1.49e-06	2.63e-03	Rejected
18	0.001	0.001	-4.58e+00	2.27e-06	2.78e-03	Rejected
17	0.5	0	-4.54e+00	2.81e-06	2.94e-03	Rejected
16	0.1	0	-4.23e+00	1.16e-05	3.13e-03	Rejected
15	0.01	0.001	-4.06e+00	2.50e-05	3.33e-03	Rejected
14	0	0.5	-3.77e+00	8.19e-05	3.57e-03	Rejected
13	0.001	0.01	-3.64e+00	1.38e-04	3.85e-03	Rejected
12	0.1	0.001	-3.20e+00	6.97e-04	4.17e-03	Rejected
11	0.5	0.001	-3.11e+00	9.42e-04	4.55e-03	Rejected
10	0.01	0.01	-2.31e+00	1.03e-02	5.00e-03	Accepted
9	0.001	0.1	-2.27e+00	1.16e-02	5.56e-03	Accepted
8	0.001	0.5	-1.54e+00	6.14e-02	6.25e-03	Accepted
7	0.1	0.01	-1.08e+00	1.40e-01	7.14e-03	Accepted
6	0.5	0.01	-9.70e-01	1.66e-01	8.33e-03	Accepted
5	0.01	0.1	-8.16e-01	2.07e-01	1.00e-02	Accepted
4	0.01	0.5	-3.97e-01	3.46e-01	1.25e-02	Accepted
3	0.1	0.1	-3.53e-01	3.62e-01	1.67e-02	Accepted
2	0.5	0.5	-2.42e-01	4.04e-01	2.50e-02	Accepted
1	0.5	0.1	-1.54e-01	4.39e-01	5.00e-02	Accepted

required to test multiple combinations of parameter values. To decide which combination of parameters produces the "best" results, the Holm procedure has been applied to the results obtained from those various combinations. This procedure compares multiple algorithms based on their performance on the set of test problems under interest for this special issue. In the tables below, the performance is estimated either with the average value returned by the algorithm over multiple trials on a given test function, or with the median value. The Holm procedure then selects the algorithm presenting the best performance (based on the algorithms' average ranking over the whole set of test functions, see [35,36] for details) and tests the other algorithms for the null-hypothesis, i.e., whether the performance of each algorithm is the same as that of the best. The case where the null hypothesis is rejected, meaning that the performance of the algorithm is significantly different from the best performance, is indicated with the word "Rejected" in the tables. Otherwise, the word "Accepted" indicates that the performance is not significantly different. For the sake of completeness, the tables below moreover indicate the algorithm's rank ("i"), the z and p-values obtained, as well as the threshold (α/i) against which the p-value is compared.

The results of this analysis are presented in Tables 1 to 10. In order to help in choosing optimal values of p_s and p_u, Tables 11 and 12 have been produced. They indicated the rank obtained by a given combination of p_s and p_u in the Holm procedures described above. In a sense, they are a repetition of the same data, but presented in a way which increases the legibility of

Table 2: Holm test for parameter setting in 50 dimensions (reference: $p_s = 0.5$, $p_u = 0.5$, estimator = median)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0.01	-3.73e+00	9.76e-05	2.08e-03	Rejected
23	0	0.001	-3.50e+00	2.29e-04	2.17e-03	Rejected
22	0	0	-3.35e+00	4.04e-04	2.27e-03	Rejected
21	0	0.1	-3.22e+00	6.45e-04	2.38e-03	Rejected
20	0	0.5	-3.02e+00	1.27e-03	2.50e-03	Rejected
19	0.001	0	-2.31e+00	1.03e-02	2.63e-03	Accepted
18	0.001	0.001	-2.16e+00	1.54e-02	2.78e-03	Accepted
17	0.001	0.01	-1.70e+00	4.48e-02	2.94e-03	Accepted
16	0.001	0.1	-1.43e+00	7.60e-02	3.13e-03	Accepted
15	0.001	0.5	-1.43e+00	7.60e-02	3.33e-03	Accepted
14	0.1	0	-1.39e+00	8.25e-02	3.57e-03	Accepted
13	0.5	0.001	-1.23e+00	1.09e-01	3.85e-03	Accepted
12	0.01	0.001	-1.06e+00	1.45e-01	4.17e-03	Accepted
11	0.01	0	-1.01e+00	1.55e-01	4.55e-03	Accepted
10	0.1	0.001	-9.48e-01	1.72e-01	5.00e-03	Accepted
9	0.01	0.1	-8.16e-01	2.07e-01	5.56e-03	Accepted
8	0.01	0.5	-7.49e-01	2.27e-01	6.25e-03	Accepted
7	0.5	0	-7.27e-01	2.33e-01	7.14e-03	Accepted
6	0.5	0.01	-6.83e-01	2.47e-01	8.33e-03	Accepted
5	0.01	0.01	-5.29e-01	2.98e-01	1.00e-02	Accepted
4	0.1	0.01	-4.85e-01	3.14e-01	1.25e-02	Accepted
3	0.1	0.1	-3.97e-01	3.46e-01	1.67e-02	Accepted
2	0.5	0.1	-1.76e-01	4.30e-01	2.50e-02	Accepted
1	0.1	0.5	-1.32e-01	4.47e-01	5.00e-02	Accepted

Table 3: Holm test for parameter setting in 100 dimensions (reference = $p_s = 0.1$, $p_u = 0.5$, estimator = average)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0.001	-6.08e+00	5.88e-10	2.08e-03	Rejected
23	0	0	-5.97e+00	1.16e-09	2.17e-03	Rejected
22	0	0.01	-5.95e+00	1.33e-09	2.27e-03	Rejected
21	0	0.1	-5.00e+00	2.82e-07	2.38e-03	Rejected
20	0.001	0	-4.96e+00	3.54e-07	2.50e-03	Rejected
19	0.01	0	-4.92e+00	4.43e-07	2.63e-03	Rejected
18	0.01	0.001	-4.36e+00	6.38e-06	2.78e-03	Rejected
17	0.5	0	-4.12e+00	1.88e-05	2.94e-03	Rejected
16	0	0.5	-4.01e+00	3.02e-05	3.13e-03	Rejected
15	0.1	0	-3.99e+00	3.31e-05	3.33e-03	Rejected
14	0.001	0.001	-3.79e+00	7.50e-05	3.57e-03	Rejected
13	0.001	0.01	-2.98e+00	1.46e-03	3.85e-03	Rejected
12	0.5	0.001	-2.45e+00	7.21e-03	4.17e-03	Accepted
11	0.1	0.001	-2.29e+00	1.09e-02	4.55e-03	Accepted
10	0.01	0.01	-2.07e+00	1.91e-02	5.00e-03	Accepted
9	0.001	0.1	-1.56e+00	5.88e-02	5.56e-03	Accepted
8	0.1	0.01	-1.26e+00	1.04e-01	6.25e-03	Accepted
7	0.001	0.5	-1.17e+00	1.21e-01	7.14e-03	Accepted
6	0.5	0.01	-8.60e-01	1.95e-01	8.33e-03	Accepted
5	0.01	0.1	-3.97e-01	3.46e-01	1.00e-02	Accepted
4	0.1	0.1	-3.31e-01	3.70e-01	1.25e-02	Accepted
3	0.5	0.1	-3.31e-01	3.70e-01	1.67e-02	Accepted
2	0.01	0.5	-2.42e-01	4.04e-01	2.50e-02	Accepted
1	0.5	0.5	0.00e+00	5.00e-01	5.00e-02	Accepted

Table 4: Holm test for parameter setting in 100 dimensions (reference: $p_s = 0.5$, $p_u = 0.5$, estimator = median)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0.01	-3.70e+00	1.07e-04	2.08e-03	Rejected
23	0	0.001	-3.68e+00	1.16e-04	2.17e-03	Rejected
22	0	0.1	-3.28e+00	5.11e-04	2.27e-03	Rejected
21	0	0	-3.15e+00	8.11e-04	2.38e-03	Rejected
20	0	0.5	-2.91e+00	1.81e-03	2.50e-03	Rejected
19	0.001	0	-2.05e+00	2.02e-02	2.63e-03	Accepted
18	0.001	0.001	-1.50e+00	6.70e-02	2.78e-03	Accepted
17	0.1	0	-1.41e+00	7.92e-02	2.94e-03	Accepted
16	0.5	0	-1.41e+00	7.92e-02	3.13e-03	Accepted
15	0.01	0	-1.37e+00	8.59e-02	3.33e-03	Accepted
14	0.01	0.001	-1.34e+00	8.94e-02	3.57e-03	Accepted
13	0.001	0.01	-1.26e+00	1.04e-01	3.85e-03	Accepted
12	0.001	0.5	-1.15e+00	1.26e-01	4.17e-03	Accepted
11	0.001	0.1	-1.10e+00	1.35e-01	4.55e-03	Accepted
10	0.5	0.001	-1.04e+00	1.50e-01	5.00e-03	Accepted
9	0.1	0.001	-9.26e-01	1.77e-01	5.56e-03	Accepted
8	0.1	0.01	-4.63e-01	3.22e-01	6.25e-03	Accepted
7	0.01	0.5	-4.41e-01	3.30e-01	7.14e-03	Accepted
6	0.01	0.1	-3.97e-01	3.46e-01	8.33e-03	Accepted
5	0.01	0.01	-3.53e-01	3.62e-01	1.00e-02	Accepted
4	0.5	0.01	-3.53e-01	3.62e-01	1.25e-02	Accepted
3	0.1	0.1	-2.42e-01	4.04e-01	1.67e-02	Accepted
2	0.5	0.1	-1.54e-01	4.39e-01	2.50e-02	Accepted
1	0.1	0.5	-6.61e-02	4.74e-01	5.00e-02	Accepted

Table 5: Holm test for parameter setting in 200 dimensions (reference: $p_s = 0.1$, $p_u = 0.5$, estimator = average)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0	-5.82e+00	2.96e-09	2.08e-03	Rejected
23	0	0.001	-5.75e+00	4.39e-09	2.17e-03	Rejected
22	0	0.01	-5.40e+00	3.33e-08	2.27e-03	Rejected
21	0.001	0	-5.22e+00	8.76e-08	2.38e-03	Rejected
20	0.01	0	-4.92e+00	4.43e-07	2.50e-03	Rejected
19	0	0.1	-4.65e+00	1.65e-06	2.63e-03	Rejected
18	0	0.5	-4.52e+00	3.11e-06	2.78e-03	Rejected
17	0.5	0	-4.08e+00	2.27e-05	2.94e-03	Rejected
16	0.001	0.001	-4.03e+00	2.75e-05	3.13e-03	Rejected
15	0.1	0	-3.90e+00	4.78e-05	3.33e-03	Rejected
14	0.01	0.001	-3.57e+00	1.78e-04	3.57e-03	Rejected
13	0.001	0.01	-2.51e+00	5.99e-03	3.85e-03	Accepted
12	0.5	0.001	-2.40e+00	8.14e-03	4.17e-03	Accepted
11	0.1	0.001	-2.09e+00	1.81e-02	4.55e-03	Accepted
10	0.01	0.01	-1.50e+00	6.70e-02	5.00e-03	Accepted
9	0.1	0.01	-9.48e-01	1.72e-01	5.56e-03	Accepted
8	0.5	0.01	-8.16e-01	2.07e-01	6.25e-03	Accepted
7	0.001	0.5	-6.39e-01	2.61e-01	7.14e-03	Accepted
6	0.001	0.1	-5.73e-01	2.83e-01	8.33e-03	Accepted
5	0.1	0.1	-2.64e-01	3.96e-01	1.00e-02	Accepted
4	0.01	0.1	-2.42e-01	4.04e-01	1.25e-02	Accepted
3	0.01	0.5	-2.20e-01	4.13e-01	1.67e-02	Accepted
2	0.5	0.1	-2.20e-01	4.13e-01	2.50e-02	Accepted
1	0.5	0.5	-1.54e-01	4.39e-01	5.00e-02	Accepted

Table 6: Holm test for parameter setting in 200 dimensions (reference: $p_s = 0.5$, $p_u = 0.5$, estimator = median)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0.01	-3.50e+00	2.29e-04	2.08e-03	Rejected
23	0	0.001	-3.44e+00	2.92e-04	2.17e-03	Rejected
22	0	0.1	-3.22e+00	6.45e-04	2.27e-03	Rejected
21	0	0	-3.13e+00	8.74e-04	2.38e-03	Rejected
20	0	0.5	-3.09e+00	1.01e-03	2.50e-03	Rejected
19	0.01	0	-1.85e+00	3.20e-02	2.63e-03	Accepted
18	0.001	0	-1.74e+00	4.08e-02	2.78e-03	Accepted
17	0.1	0	-1.74e+00	4.08e-02	2.94e-03	Accepted
16	0.5	0	-1.61e+00	5.38e-02	3.13e-03	Accepted
15	0.001	0.001	-1.59e+00	5.63e-02	3.33e-03	Accepted
14	0.01	0.001	-1.17e+00	1.21e-01	3.57e-03	Accepted
13	0.5	0.001	-9.70e-01	1.66e-01	3.85e-03	Accepted
12	0.001	0.01	-8.60e-01	1.95e-01	4.17e-03	Accepted
11	0.001	0.1	-7.71e-01	2.20e-01	4.55e-03	Accepted
10	0.001	0.5	-7.49e-01	2.27e-01	5.00e-03	Accepted
9	0.1	0.001	-7.49e-01	2.27e-01	5.56e-03	Accepted
8	0.01	0.5	-4.19e-01	3.38e-01	6.25e-03	Accepted
7	0.1	0.01	-3.75e-01	3.54e-01	7.14e-03	Accepted
6	0.01	0.01	-3.53e-01	3.62e-01	8.33e-03	Accepted
5	0.5	0.01	-3.53e-01	3.62e-01	1.00e-02	Accepted
4	0.01	0.1	-2.87e-01	3.87e-01	1.25e-02	Accepted
3	0.5	0.1	-2.42e-01	4.04e-01	1.67e-02	Accepted
2	0.1	0.1	-1.76e-01	4.30e-01	2.50e-02	Accepted
1	0.1	0.5	-4.41e-02	4.82e-01	5.00e-02	Accepted

Table 7: Holm test for parameter setting in 500 dimensions (reference: $p_s = 0.5$, $p_u = 0.1$, estimator = average)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0	-5.62e+00	9.52e-09	2.08e-03	Rejected
23	0	0.001	-5.53e+00	1.58e-08	2.17e-03	Rejected
22	0.001	0	-5.38e+00	3.76e-08	2.27e-03	Rejected
21	0	0.01	-4.85e+00	6.20e-07	2.38e-03	Rejected
20	0	0.5	-4.65e+00	1.65e-06	2.50e-03	Rejected
19	0	0.1	-4.56e+00	2.53e-06	2.63e-03	Rejected
18	0.01	0	-4.41e+00	5.21e-06	2.78e-03	Rejected
17	0.1	0	-4.36e+00	6.38e-06	2.94e-03	Rejected
16	0.5	0	-4.19e+00	1.41e-05	3.13e-03	Rejected
15	0.001	0.001	-3.24e+00	5.97e-04	3.33e-03	Rejected
14	0.01	0.001	-2.62e+00	4.36e-03	3.57e-03	Accepted
13	0.1	0.001	-2.34e+00	9.74e-03	3.85e-03	Accepted
12	0.5	0.001	-2.25e+00	1.23e-02	4.17e-03	Accepted
11	0.001	0.01	-1.56e+00	5.88e-02	4.55e-03	Accepted
10	0.01	0.01	-1.34e+00	8.94e-02	5.00e-03	Accepted
9	0.001	0.5	-9.26e-01	1.77e-01	5.56e-03	Accepted
8	0.1	0.01	-8.16e-01	2.07e-01	6.25e-03	Accepted
7	0.001	0.1	-7.49e-01	2.27e-01	7.14e-03	Accepted
6	0.1	0.1	-6.39e-01	2.61e-01	8.33e-03	Accepted
5	0.5	0.01	-5.95e-01	2.76e-01	1.00e-02	Accepted
4	0.1	0.5	-5.73e-01	2.83e-01	1.25e-02	Accepted
3	0.01	0.5	-5.07e-01	3.06e-01	1.67e-02	Accepted
2	0.01	0.1	-4.63e-01	3.22e-01	2.50e-02	Accepted
1	0.5	0.5	-3.53e-01	3.62e-01	5.00e-02	Accepted

Table 8: Holm test for parameter setting in 500 dimensions (reference: $p_s = 0.5$, $p_u = 0.1$, estimator = median)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0.001	-3.28e+00	5.11e-04	2.08e-03	Rejected
23	0	0.01	-2.89e+00	1.94e-03	2.17e-03	Rejected
22	0	0.1	-2.82e+00	2.39e-03	2.27e-03	Accepted
21	0	0.5	-2.76e+00	2.93e-03	2.38e-03	Accepted
20	0	0	-2.60e+00	4.65e-03	2.50e-03	Accepted
19	0.01	0	-1.90e+00	2.90e-02	2.63e-03	Accepted
18	0.5	0	-1.79e+00	3.71e-02	2.78e-03	Accepted
17	0.001	0.001	-1.52e+00	6.41e-02	2.94e-03	Accepted
16	0.001	0	-1.45e+00	7.29e-02	3.13e-03	Accepted
15	0.1	0	-1.39e+00	8.25e-02	3.33e-03	Accepted
14	0.5	0.001	-9.26e-01	1.77e-01	3.57e-03	Accepted
13	0.1	0.001	-7.27e-01	2.33e-01	3.85e-03	Accepted
12	0.001	0.01	-5.95e-01	2.76e-01	4.17e-03	Accepted
11	0.001	0.5	-5.73e-01	2.83e-01	4.55e-03	Accepted
10	0.01	0.001	-5.29e-01	2.98e-01	5.00e-03	Accepted
9	0.01	0.5	-4.41e-01	3.30e-01	5.56e-03	Accepted
8	0.001	0.1	-4.19e-01	3.38e-01	6.25e-03	Accepted
7	0.01	0.1	-3.75e-01	3.54e-01	7.14e-03	Accepted
6	0.01	0.01	-2.64e-01	3.96e-01	8.33e-03	Accepted
5	0.1	0.5	-1.98e-01	4.21e-01	1.00e-02	Accepted
4	0.5	0.01	-1.54e-01	4.39e-01	1.25e-02	Accepted
3	0.5	0.5	-1.32e-01	4.47e-01	1.67e-02	Accepted
2	0.1	0.1	-8.82e-02	4.65e-01	2.50e-02	Accepted
1	0.1	0.01	-4.41e-02	4.82e-01	5.00e-02	Accepted

Table 9: Holm test for parameter setting in 1000 dimensions (reference: $p_s = 0.5$, $p_u = 0.1$, estimator = average)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0	-4.83e+00	6.93e-07	2.08e-03	Rejected
23	0	0.001	-4.61e+00	2.05e-06	2.17e-03	Rejected
22	0.001	0	-4.39e+00	5.77e-06	2.27e-03	Rejected
21	0.5	0	-4.32e+00	7.80e-06	2.38e-03	Rejected
20	0	0.01	-4.30e+00	8.61e-06	2.50e-03	Rejected
19	0	0.1	-4.19e+00	1.41e-05	2.63e-03	Rejected
18	0	0.5	-4.08e+00	2.27e-05	2.78e-03	Rejected
17	0.01	0	-3.86e+00	5.73e-05	2.94e-03	Rejected
16	0.1	0	-3.73e+00	9.76e-05	3.13e-03	Rejected
15	0.001	0.001	-2.73e+00	3.14e-03	3.33e-03	Rejected
14	0.01	0.001	-2.34e+00	9.74e-03	3.57e-03	Accepted
13	0.5	0.001	-2.18e+00	1.46e-02	3.85e-03	Accepted
12	0.1	0.001	-1.92e+00	2.76e-02	4.17e-03	Accepted
11	0.001	0.01	-1.28e+00	1.01e-01	4.55e-03	Accepted
10	0.001	0.1	-7.05e-01	2.40e-01	5.00e-03	Accepted
9	0.01	0.01	-7.05e-01	2.40e-01	5.56e-03	Accepted
8	0.1	0.01	-6.17e-01	2.69e-01	6.25e-03	Accepted
7	0.001	0.5	-3.97e-01	3.46e-01	7.14e-03	Accepted
6	0.01	0.1	-3.97e-01	3.46e-01	8.33e-03	Accepted
5	0.01	0.5	-3.53e-01	3.62e-01	1.00e-02	Accepted
4	0.5	0.01	-3.31e-01	3.70e-01	1.25e-02	Accepted
3	0.1	0.1	-2.64e-01	3.96e-01	1.67e-02	Accepted
2	0.1	0.5	-1.76e-01	4.30e-01	2.50e-02	Accepted
1	0.5	0.5	-1.54e-01	4.39e-01	5.00e-02	Accepted

Table 10: Holm test for parameter setting in 1000 dimensions (reference: $p_s = 0.5$, $p_u = 0.1$, estimator = median)

i	p_s	p_u	z	p	α/i	Hypothesis
24	0	0.01	-2.95e+00	1.57e-03	2.08e-03	Rejected
23	0	0.5	-2.95e+00	1.57e-03	2.17e-03	Rejected
22	0	0.1	-2.89e+00	1.94e-03	2.27e-03	Rejected
21	0	0.001	-2.84e+00	2.23e-03	2.38e-03	Rejected
20	0	0	-2.18e+00	1.46e-02	2.50e-03	Accepted
19	0.5	0	-2.12e+00	1.72e-02	2.63e-03	Accepted
18	0.001	0	-1.61e+00	5.38e-02	2.78e-03	Accepted
17	0.01	0	-1.59e+00	5.63e-02	2.94e-03	Accepted
16	0.001	0.001	-1.50e+00	6.70e-02	3.13e-03	Accepted
15	0.1	0	-1.45e+00	7.29e-02	3.33e-03	Accepted
14	0.5	0.001	-9.48e-01	1.72e-01	3.57e-03	Accepted
13	0.01	0.001	-7.93e-01	2.14e-01	3.85e-03	Accepted
12	0.001	0.1	-6.61e-01	2.54e-01	4.17e-03	Accepted
11	0.1	0.001	-6.17e-01	2.69e-01	4.55e-03	Accepted
10	0.001	0.5	-5.95e-01	2.76e-01	5.00e-03	Accepted
9	0.01	0.5	-5.07e-01	3.06e-01	5.56e-03	Accepted
8	0.001	0.01	-4.19e-01	3.38e-01	6.25e-03	Accepted
7	0.01	0.1	-3.09e-01	3.79e-01	7.14e-03	Accepted
6	0.1	0.01	-2.87e-01	3.87e-01	8.33e-03	Accepted
5	0.1	0.1	-2.87e-01	3.87e-01	1.00e-02	Accepted
4	0.01	0.01	-2.42e-01	4.04e-01	1.25e-02	Accepted
3	0.5	0.01	-2.42e-01	4.04e-01	1.67e-02	Accepted
2	0.5	0.5	-4.41e-02	4.82e-01	2.50e-02	Accepted
1	0.1	0.5	-2.20e-02	4.91e-01	5.00e-02	Accepted

a parameter's influence on the algorithm's performance. The best-ranking algorithm is indicated with an "X" in the table, and the number values are the same as in the "i" column of the corresponding table above. Parameter combinations which do no reject the null hypothesis in the tests described above are indicated in boldface.

From those tables, we can see that choosing the average or the median for estimating the performance of an algorithm on a given function leads to a significant difference in the tables: with the average, low values of both p_s and p_u lead to significantly inferior results, while when using the median, the difference in performance is not significant anymore. The best ranking combination of p_s and p_u are however always in the lower-right area of the tables, meaning values of 0.1 or 0.5 for both parameters. The rank numbers are increasing when traveling the tables from the lower-right to the upper-left area i.e., to lower values of the parameters.

More particularly, when $p_s = 0$ the Holm procedure considers the performance to be significantly worse than with higher values of the parameter. With the average as performance estimator, $p_u = 0$ leads to significantly inferior performance as well, in all dimensionalities.

As mentioned above, when considering the median as the performance estimator, the Holm procedure considers that most combinations of the parameters to produce equivalent performance, with the exception of the case when $p_s = 0$ for dimensionalities of 200 and less. The relative ranking of the combinations of parameters are however similar to those obtained when

Table 11: Ranking of SOUPDE for various values of parameters p_s and p_u. The best combination of p_s and p_u is indicated by X. All the ranks in boldface indicate that there is no significant difference in performance between these combinations of parameters according to the Holm procedure using the average as performance estimator

(a) 50 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	22	24	23	19	14
0.001	21	18	13	**9**	**8**
0.01	20	15	**10**	**5**	**4**
0.1	16	12	**7**	**3**	**X**
0.5	17	11	**6**	**1**	**2**

(b) 100 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	23	24	22	21	16
0.001	20	14	13	**9**	**7**
0.01	19	18	**10**	**5**	**2**
0.1	15	**11**	**8**	**4**	**X**
0.5	17	**12**	**6**	**3**	**1**

(c) 200 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	24	23	22	19	18
0.001	21	16	**13**	**6**	**7**
0.01	20	14	**10**	**4**	**3**
0.1	15	**11**	**9**	**5**	**X**
0.5	17	**12**	**8**	**2**	**1**

(d) 500 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	24	23	20	19	18
0.001	22	15	**11**	**10**	**7**
0.01	17	**14**	**9**	**6**	**5**
0.1	16	**12**	**8**	**3**	**2**
0.5	21	**13**	**4**	**X**	**1**

(e) 1000 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	24	23	21	19	20
0.001	22	15	**11**	**7**	**9**
0.01	18	**14**	**10**	**2**	**3**
0.1	17	**13**	**8**	**6**	**4**
0.5	16	**12**	**5**	**X**	**1**

Table 12: Ranking of SOUPDE for various values of parameters p_s and p_u. The best combination of p_s and p_u is indicated by X. All the ranks in boldface indicate that there is no significant difference in performance between these combinations of parameters according to the Holm procedure using the median as performance estimator

(a) 50 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	22	23	24	21	20
0.001	**19**	**18**	**17**	**15**	**16**
0.01	**11**	**12**	**5**	**9**	**8**
0.1	**14**	**10**	**4**	**3**	**1**
0.5	**7**	**13**	**6**	**2**	**X**

(b) 100 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	21	23	24	22	20
0.001	**19**	**18**	**13**	**11**	**12**
0.01	**15**	**14**	**4**	**6**	**7**
0.1	**16**	**9**	**8**	**3**	**1**
0.5	**17**	**10**	**5**	**2**	**X**

(c) 200 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	21	23	24	22	20
0.001	**17**	**15**	**12**	**11**	**10**
0.01	**19**	**14**	**5**	**4**	**8**
0.1	**18**	**9**	**7**	**2**	**1**
0.5	**16**	**13**	**6**	**3**	**X**

(d) 500 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	**20**	24	23	**22**	**21**
0.001	**16**	**17**	**12**	**8**	**11**
0.01	**19**	**10**	**6**	**7**	**9**
0.1	**15**	**13**	**1**	**2**	**5**
0.5	**18**	**14**	**4**	**X**	**3**

(e) 1000 dimensions

p_s \ p_u	0	0.001	0.01	0.1	0.5
0	**20**	21	23	22	24
0.001	**18**	**16**	**8**	**12**	**10**
0.01	**17**	**13**	**3**	**7**	**9**
0.1	**15**	**11**	**6**	**5**	**1**
0.5	**19**	**14**	**4**	**X**	**2**

considering the average, with the same progression form the lower-right area to the upper-left.

Based on the results presented above, both the shuffling and update probabilities p_s and p_u have been set equal to 0.5 for the experiments described in the remainder of this paper.

3.2 Performance Analysis

Tables 13, 18, 23, 28 and 33 present the error obtained on the fitness values by SOUPDE at the end of the allocated budget of fitness evaluations, for dimensions 50, 100, 200, 500 and 1000, respectively. In each table, the top subtable presents the average, median, minimum and maximum fitness values obtained by performing 25 independent runs of the algorithm on each test function. Conforming to the requirements for this study, all values below 10^{-14} have been rounded to 0. The small histogram shown between the minimum and maximum values graphically shows how those 25 values are distributed between the minimum and maximum values. This representation allows to see for example in Table 13 that SOUPDE, when applied to function F14 reaches 0 in all the cases except two. This can be interpreted as the fact that in most runs, SOUPDE performs well on function F14 in 50 dimensions, except in rare conditions where it fails to find the function's minimum. The bottom subtables present the average and median error values reached by SOUPDE compared to those of the three reference algorithms, DE, CHC and G-CMA-ES. The best average value(s) for each function is highlighted in bold, while the best median value(s) are highlighted in italics.

From the study of those bar graphs, it has been concluded that for most functions, the average error value is not a good estimator of the performance of the algorithm, and that the median is generally better suited. We have therefore decided to perform the Wilcoxon Signed-Rank test and the Holm procedure using both the average and median error values as a measure of the algorithm's performance and present both series of results.

The Wilcoxon Signed-Rank test [37] allows to compare two algorithms based on their individual performances on the set of functions F1–F19*. SOUPDE is thus compared to DE, CHC and G-CMA-ES (except for the 1000-dimensional test functions where G-CMA-ES is not compared against). Tables 14, 19, 24, 29 and 34 show the results of this test when using the algorithms' average error value as a measure of their performance, while Tables 16, 21, 26, 31 and 36 present the results of the same test when the median is used. A two-tailed Signed-Rank test with a threshold set to 0.05 can be performed to determinate if SOUPDE has similar performance than the algorithm it is compared to (this is the null hypothesis for this test). The p-value obtained from this test is indicated in the table. However, in order to find out if SOUPDE potentially outperforms or is outperformed by the other algorithm, we used up to two directional Signed-Rank tests with a threshold of 0.025. Such a test has first been performed in order to determinate if SOUPDE outperforms the algorithm it is compared to. If this is the case, it is thus indicated in the table with a "+" symbol. Otherwise, a second directional Signed-Rank test with the same threshold is performed in order

to determinate if the other algorithm outperforms SOUPDE. If this is the case, it is indicated in the table with a "-" symbol. Otherwise, the case when no algorithm is outperforming the other one is indicated with an "=" symbol in the table.

The Wilcoxon Signed-Rank test thus indicates that SOUPDE outperforms CHC and G-CMA-ES in every case. When considering the algorithms' average performance, SOUPDE exhibits performances similar to the ones of DE in 50, and 200 dimensions, but outperforms the latter in 50, 100, 500 and 1000 dimensions. When considering the algorithms' median performance, SOUPDE outperforms DE in all the dimensionalities under consideration.

As explained in [36] and [35], the Wilcoxon test does not allow to extract conclusion when more than one pairwise comparison is made, because the errors of each pairwise comparison accumulate. The Holm procedure [38] is one method that allows to alleviate this problem by adapting the threshold against which the p-value is compared for each algorithm. Tables 15, 20, 25, 30 and 35 show the results of this test when using the algorithms' average error value as a measure of their performance, while Tables 17, 22, 27, 32 and 37 present the results of the same test when the median is used. For this procedure, one formulates the null-hypothesis as "the two algorithm exhibit similar performance". The algorithms are then ranked according to their performance on each test function, and their average rank is then computed. The algorithm with the smallest rank is chosen as a reference against which the others are compared. The tables show the z value obtained from the Friedman test, and from which a p-value (p in the table) is derived. This p-value is then compared against a threshold (α/i) which varies depending on the algorithm's average rank. If the p-value is lower than the threshold, the null hypothesis is rejected and the reference algorithm therefore exhibits better performance than the one it is compared to. Otherwise, the null hypothesis is accepted and the two algorithms have similar performance.

The Holm procedure confirms the findings of the Wilcoxon test regarding CHC and G-CMA-ES for all the dimensionalities under consideration. When considering the algorithm's average performance, SOUPDE is ranked first for all dimensionalities while DE comes second. On the contrary to the Wilcoxon test mentioned above, the Holm procedure does not consider that SOUPDE is performing significantly better than DE. The difference between the results of those tests might be due to the fact that the Holm procedure seems more "strict" than the Wilcoxon test. Since both DE and SOUPDE reach the global minimum of the functions in several cases, the difference in average rank of those algorithms is not large enough to reject the null hypothesis.

As a general consideration, the proposed SOUPDE algorithm appears to efficiently handle most of the problems in the benchmark. On the other hand, numerical results show that SOUPDE does not succeed at efficiently minimizing test problems F2, F3, F8, F13, and F17*. For these test problems the search logic of DE appear to be not so efficient since also the DE version in this article fails at detecting the minimum for the above-mentioned functions. Regarding test problems F2 and F8 the directional search logic of G-CMA-ES seems more promising than the randomized DE logic. This fact, according to our conjecture, is due to the presence of a strong optimal basin of attraction

within the landscape related to these functions. The exploitative directional logic of G-CMA-ES is, in our opinion, then suitable for problems of this kind. Regarding function F3, G-CMA-ES outperforms DE based algorithms but still fails at detecting the global minimum. Regarding test problems, F13 and F17*, the landscapes are very complex and, to our knowledge, currently, there are no algorithms which are capable to solve them. However, our proposed SOUPDE succeeds at outperforming all the other algorithms considered in this study for these two problems.

Table 13: Error in 50D

(a) Average, Median, Min and Max Error of SOUPDE

	SOUPDE			
	Average	Median	Minimum	Maximum
F1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F2	$1.18e+00$	$6.73e-01$	$9.70e-02$	$4.21e+00$
F3	$3.10e+01$	$2.37e+01$	$2.12e+01$	$7.19e+01$
F4	$3.98e-02$	$0.00e+00$	$0.00e+00$	$9.95e-01$
F5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F6	$1.47e-14$	$1.47e-14$	$1.47e-14$	$1.47e-14$
F7	$2.28e-14$	$2.28e-14$	$2.28e-14$	$2.28e-14$
F8	$9.69e-02$	$8.42e-02$	$4.11e-02$	$1.90e-01$
F9	$3.75e-06$	$3.75e-06$	$3.75e-06$	$3.75e-06$
F10	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F11	$3.09e-06$	$3.09e-06$	$3.09e-06$	$3.09e-06$
F12	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F13	$2.06e+01$	$2.01e+01$	$1.88e+01$	$2.50e+01$
F14	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F15	$1.38e-14$	$1.38e-14$	$1.38e-14$	$1.38e-14$
F16*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F17*	$2.53e-01$	$7.38e-02$	$1.18e-02$	$3.99e+00$
F18*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F19*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$

(b) Average and Median Error of SOUPDE compared to the reference algorithms

	SOUPDE		DE		CHC		G-CMA-ES	
	Average	Median	Average	Median	Average	Median	Average	Median
F1	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$1.67e-11$	$1.67e-11$	$\mathbf{0.00e+00}$	*0.00e+00*
F2	$1.18e+00$	$6.73e-01$	$3.60e-01$	$3.29e-01$	$6.19e+01$	$6.19e+01$	$\mathbf{2.75e-11}$	*2.64e-11*
F3	$3.10e+01$	$2.37e+01$	$2.89e+01$	$2.90e+01$	$1.25e+06$	$1.25e+06$	$\mathbf{7.97e-01}$	*0.00e+00*
F4	$\mathbf{3.98e-02}$	*0.00e+00*	$3.98e-02$	$1.51e-13$	$7.43e+01$	$7.43e+01$	$1.05e+02$	$1.08e+02$
F5	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$1.67e-03$	$1.67e-03$	$2.96e-04$	*0.00e+00*
F6	$\mathbf{1.47e-14}$	*1.47e-14*	$1.43e-13$	$1.42e-13$	$6.15e-07$	$6.15e-07$	$2.09e+01$	$2.11e+01$
F7	$2.28e-14$	$2.28e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$2.66e-09$	$2.66e-09$	$1.01e-10$	$7.67e-11$
F8	$9.69e-02$	$8.42e-02$	$3.44e+00$	$3.54e+00$	$2.24e+02$	$2.24e+02$	$\mathbf{0.00e+00}$	*0.00e+00*
F9	$\mathbf{3.75e-06}$	*3.75e-06*	$2.73e+02$	$2.73e+02$	$3.10e+02$	$3.10e+02$	$1.66e+01$	$1.61e+01$
F10	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$7.30e+00$	$7.30e+00$	$6.81e+00$	$6.71e+00$
F11	$\mathbf{3.09e-06}$	*3.09e-06*	$6.23e-05$	$5.60e-05$	$2.16e+00$	$2.16e+00$	$3.01e+01$	$2.83e+01$
F12	$\mathbf{0.00e+00}$	*0.00e+00*	$5.35e-13$	$5.27e-13$	$9.57e-01$	$9.57e-01$	$1.88e+02$	$1.87e+02$
F13	$\mathbf{2.06e+01}$	*2.01e+01*	$2.45e+01$	$2.44e+01$	$2.08e+06$	$2.08e+06$	$1.97e+02$	$1.97e+02$
F14	$\mathbf{0.00e+00}$	*0.00e+00*	$4.16e-08$	$2.58e-08$	$6.17e+01$	$6.17e+01$	$1.09e+02$	$1.05e+02$
F15	$1.38e-14$	$1.38e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$3.98e-01$	$3.98e-01$	$9.79e-04$	$8.12e-04$
F16*	$\mathbf{0.00e+00}$	*0.00e+00*	$1.56e-09$	$1.51e-09$	$2.95e-09$	$2.95e-09$	$4.27e+02$	$4.22e+02$
F17*	$\mathbf{2.53e-01}$	*7.38e-02*	$7.98e-01$	$6.83e-01$	$2.26e+04$	$2.26e+04$	$6.89e+02$	$6.71e+02$
F18*	$\mathbf{0.00e+00}$	*0.00e+00*	$1.22e-04$	$1.20e-04$	$1.58e+01$	$1.58e+01$	$1.31e+02$	$1.27e+02$
F19*	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$3.59e+02$	$3.59e+02$	$4.76e+00$	$4.03e+00$

Table 14: Wilcoxon Signed-Rank test in 50D (reference = SOUPDE, estimator = average)

Optimizer	p-value	Hypothesis
DE	1.950e+00	=
CHC	3.815e-06	+
G-CMA-ES	4.644e-03	+

Table 15: Holm procedure in 50D (reference = SOUPDE, estimator = average)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.77e+00	8.99e-07	1.67e-02	Rejected
2	G-CMA-ES	-3.27e+00	5.43e-04	2.50e-02	Rejected
1	DE	-1.01e+00	1.57e-01	5.00e-02	Accepted

Table 16: Wilcoxon Signed-Rank test in 50D (reference = SOUPDE, estimator = median)

Optimizer	p-value	Hypothesis
DE	8.266e-03	+
CHC	3.815e-06	+
G-CMA-ES	4.507e-03	+

Table 17: Holm procedure in 50D (reference = SOUPDE, estimator = median)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.77e+00	8.99e-07	1.67e-02	Rejected
2	G-CMA-ES	-3.27e+00	5.43e-04	2.50e-02	Rejected
1	DE	-1.26e+00	1.04e-01	5.00e-02	Accepted

Table 18: Error in 100D

(a) Average, Median, Min and Max Error of SOUPDE

	SOUPDE			
	Average	Median	Minimum	Maximum
F1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F2	$7.47e+00$	$7.01e+00$	$4.10e+00$	$1.28e+01$
F3	$7.92e+01$	$7.30e+01$	$6.96e+01$	$1.18e+02$
F4	$3.98e-02$	$0.00e+00$	$0.00e+00$	$9.95e-01$
F5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F6	$3.03e-14$	$2.89e-14$	$2.53e-14$	$3.24e-14$
F7	$3.88e-14$	$3.88e-14$	$3.88e-14$	$3.88e-14$
F8	$6.55e+01$	$6.52e+01$	$3.63e+01$	$1.06e+02$
F9	$7.82e-06$	$7.82e-06$	$7.82e-06$	$7.82e-06$
F10	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F11	$6.75e-06$	$6.75e-06$	$6.75e-06$	$6.75e-06$
F12	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F13	$5.85e+01$	$5.83e+01$	$5.70e+01$	$6.37e+01$
F14	$0.00e+00$	$0.00e+00$	$0.00e+00$	$1.14e-13$
F15	$2.79e-14$	$2.79e-14$	$2.79e-14$	$2.79e-14$
F16*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F17*	$8.55e+00$	$8.74e+00$	$1.30e-03$	$1.13e+01$
F18*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F19*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$

(b) Average and Median Error of SOUPDE compared to the reference algorithms

	SOUPDE		DE		CHC		G-CMA-ES	
	Average	Median	Average	Median	Average	Median	Average	Median
F1	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$3.56e-11$	$3.56e-11$	$\mathbf{0.00e+00}$	*0.00e+00*
F2	$7.47e+00$	$7.01e+00$	$4.45e+00$	$4.34e+00$	$8.58e+01$	$8.58e+01$	$\mathbf{1.51e-10}$	*1.62e-10*
F3	$7.92e+01$	$7.30e+01$	$8.01e+01$	$7.81e+01$	$4.19e+06$	$4.19e+06$	$\mathbf{3.88e+00}$	*2.27e+00*
F4	$\mathbf{3.98e-02}$	*0.00e+00*	$7.96e-02$	$4.23e-13$	$2.19e+02$	$2.19e+02$	$2.50e+02$	$2.50e+02$
F5	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$3.83e-03$	$3.83e-03$	$1.58e-03$	*0.00e+00*
F6	$\mathbf{3.03e-14}$	*2.89e-14*	$3.10e-13$	$3.13e-13$	$4.10e-07$	$4.10e-07$	$2.12e+01$	$2.13e+01$
F7	$3.88e-14$	$3.88e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$1.40e-02$	$1.40e-02$	$4.22e-04$	$6.98e-07$
F8	$6.55e+01$	$6.52e+01$	$3.69e+02$	$3.47e+02$	$1.69e+03$	$1.69e+03$	$\mathbf{0.00e+00}$	*0.00e+00*
F9	$\mathbf{7.82e-06}$	*7.82e-06*	$5.06e+02$	$5.06e+02$	$5.86e+02$	$5.86e+02$	$1.02e+02$	$1.06e+02$
F10	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$3.30e+01$	$3.30e+01$	$1.66e+01$	$1.68e+01$
F11	$\mathbf{6.75e-06}$	*6.75e-06*	$1.28e-04$	$1.29e-04$	$7.32e+01$	$7.32e+01$	$1.64e+02$	$1.51e+02$
F12	$\mathbf{0.00e+00}$	*0.00e+00*	$5.99e-11$	$6.18e-11$	$1.03e+01$	$1.03e+01$	$4.17e+02$	$4.20e+02$
F13	$\mathbf{5.85e+01}$	*5.83e+01*	$6.17e+01$	$6.17e+01$	$2.70e+06$	$2.70e+06$	$4.21e+02$	$4.12e+02$
F14	$\mathbf{0.00e+00}$	*0.00e+00*	$4.79e-02$	$1.30e-07$	$1.66e+02$	$1.66e+02$	$2.55e+02$	$2.52e+02$
F15	$2.79e-14$	$2.79e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$8.13e+00$	$8.13e+00$	$6.30e-01$	$4.13e-01$
F16*	$\mathbf{0.00e+00}$	*0.00e+00*	$3.58e-09$	$3.53e-09$	$2.23e+01$	$2.23e+01$	$8.59e+02$	$8.48e+02$
F17*	$\mathbf{8.55e+00}$	*8.74e+00*	$1.23e+01$	$1.28e+01$	$1.47e+05$	$1.47e+05$	$1.51e+03$	$1.52e+03$
F18*	$\mathbf{0.00e+00}$	*0.00e+00*	$2.98e-04$	$2.86e-04$	$7.00e+01$	$7.00e+01$	$3.07e+02$	$3.13e+02$
F19*	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$5.45e+02$	$5.45e+02$	$2.02e+01$	$1.47e+01$

Table 19: Wilcoxon Signed-Rank test in 100D (reference = SOUPDE, estimator = average)

Optimizer	p-value	Hypothesis
DE	9.760e-03	+
CHC	3.815e-06	+
G-CMA-ES	5.316e-03	+

Table 20: Holm procedure in 100D (reference = SOUPDE, estimator = average)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.90e+00	4.78e-07	1.67e-02	Rejected
2	G-CMA-ES	-3.39e+00	3.46e-04	2.50e-02	Rejected
1	DE	-1.26e+00	1.04e-01	5.00e-02	Accepted

Table 21: Wilcoxon Signed-Rank test in 100D (reference = SOUPDE, estimator = median)

Optimizer	p-value	Hypothesis
DE	8.266e-03	+
CHC	3.815e-06	+
G-CMA-ES	6.040e-03	+

Table 22: Holm procedure in 100D (reference = SOUPDE, estimator = median)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.77e+00	8.99e-07	1.67e-02	Rejected
2	G-CMA-ES	-3.27e+00	5.43e-04	2.50e-02	Rejected
1	DE	-1.26e+00	1.04e-01	5.00e-02	Accepted

Table 23: Error in 200D

(a) Average, Median, Min and Max Error of SOUPDE

	SOUPDE			
	Average	Median	Minimum	Maximum
F1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F2	$2.38e+01$	$2.33e+01$	$2.17e+01$	$2.75e+01$
F3	$1.80e+02$	$1.71e+02$	$1.68e+02$	$2.18e+02$
F4	$1.19e-01$	$2.27e-13$	$0.00e+00$	$9.95e-01$
F5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F6	$6.40e-14$	$6.44e-14$	$5.73e-14$	$6.79e-14$
F7	$7.46e-14$	$7.46e-14$	$7.46e-14$	$7.46e-14$
F8	$2.46e+03$	$2.46e+03$	$1.75e+03$	$3.23e+03$
F9	$1.51e-05$	$1.51e-05$	$1.51e-05$	$1.51e-05$
F10	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F11	$1.43e-05$	$1.43e-05$	$1.43e-05$	$1.43e-05$
F12	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F13	$1.35e+02$	$1.32e+02$	$1.31e+02$	$1.75e+02$
F14	$3.98e-02$	$2.27e-13$	$2.27e-13$	$9.95e-01$
F15	$5.79e-14$	$5.79e-14$	$5.79e-14$	$5.79e-14$
F16*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F17*	$3.31e+01$	$3.30e+01$	$3.14e+01$	$3.52e+01$
F18*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F19*	$1.91e-14$	$1.91e-14$	$1.91e-14$	$1.91e-14$

(b) Average and Median Error of SOUPDE compared to the reference algorithms

	SOUPDE		DE		CHC		G-CMA-ES	
	Average	Median	Average	Median	Average	Median	Average	Median
F1	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$8.34e-01$	$8.34e-01$	$\mathbf{0.00e+00}$	*0.00e+00*
F2	$2.38e+01$	$2.33e+01$	$1.92e+01$	$1.93e+01$	$1.03e+02$	$1.03e+02$	$\mathbf{1.16e-09}$	*9.91e-10*
F3	$1.80e+02$	$1.71e+02$	$1.78e+02$	$1.77e+02$	$2.01e+07$	$2.01e+07$	$\mathbf{8.91e+01}$	*8.95e+01*
F4	$\mathbf{1.19e-01}$	*2.27e-13*	$1.27e-01$	$3.58e-12$	$5.40e+02$	$5.40e+02$	$6.48e+02$	$6.68e+02$
F5	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$8.76e-03$	$8.76e-03$	$\mathbf{0.00e+00}$	*0.00e+00*
F6	$\mathbf{6.40e-14}$	*6.44e-14*	$6.54e-13$	$6.54e-13$	$1.23e+00$	$1.23e+00$	$2.14e+01$	$2.14e+01$
F7	$7.46e-14$	$7.46e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$2.59e-01$	$2.59e-01$	$1.17e-01$	$2.61e-02$
F8	$2.46e+03$	$2.46e+03$	$5.53e+03$	$5.33e+03$	$9.38e+03$	$9.38e+03$	$\mathbf{0.00e+00}$	*0.00e+00*
F9	$\mathbf{1.51e-05}$	*1.51e-05*	$1.01e+03$	$1.01e+03$	$1.19e+03$	$1.19e+03$	$3.75e+02$	$3.81e+02$
F10	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$7.13e+01$	$7.13e+01$	$4.43e+01$	$4.41e+01$
F11	$\mathbf{1.43e-05}$	*1.43e-05*	$2.62e-04$	$2.59e-04$	$3.85e+02$	$3.85e+02$	$8.03e+02$	$7.93e+02$
F12	$\mathbf{0.00e+00}$	*0.00e+00*	$9.76e-10$	$9.36e-10$	$7.44e+01$	$7.44e+01$	$9.06e+02$	$9.08e+02$
F13	$\mathbf{1.35e+02}$	*1.32e+02*	$1.36e+02$	$1.36e+02$	$5.75e+06$	$5.75e+06$	$9.43e+02$	$9.34e+02$
F14	$\mathbf{3.98e-02}$	*2.27e-13*	$1.38e-01$	$2.71e-07$	$4.29e+02$	$4.29e+02$	$6.09e+02$	$6.24e+02$
F15	$5.79e-14$	$5.79e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$2.14e+01$	$2.14e+01$	$1.75e+00$	$2.10e+00$
F16*	$\mathbf{0.00e+00}$	*0.00e+00*	$7.46e-09$	$7.26e-09$	$1.60e+02$	$1.60e+02$	$1.92e+03$	$1.90e+03$
F17*	$\mathbf{3.31e+01}$	*3.30e+01*	$3.70e+01$	$3.70e+01$	$1.75e+05$	$1.75e+05$	$3.36e+03$	$3.33e+03$
F18*	$\mathbf{0.00e+00}$	*0.00e+00*	$4.73e-04$	$4.70e-04$	$2.12e+02$	$2.12e+02$	$6.89e+02$	$6.88e+02$
F19*	$1.91e-14$	$1.91e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$2.06e+03$	$2.06e+03$	$7.52e+02$	$5.74e+02$

Table 24: Wilcoxon Signed-Rank test in 200D (reference = SOUPDE, estimator = average)

Optimizer	p-value	Hypothesis
DE	1.941e+00	=
CHC	3.815e-06	+
G-CMA-ES	1.794e-02	+

Table 25: Holm procedure in 200D (reference = SOUPDE, estimator = average)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.65e+00	1.67e-06	1.67e-02	Rejected
2	G-CMA-ES	-3.14e+00	8.41e-04	2.50e-02	Rejected
1	DE	-8.80e-01	1.90e-01	5.00e-02	Accepted

Table 26: Wilcoxon Signed-Rank test in 200D (reference = SOUPDE, estimator = median)

Optimizer	p-value	Hypothesis
DE	1.048e-02	+
CHC	3.815e-06	+
G-CMA-ES	1.794e-02	+

Table 27: Holm procedure in 200D (reference = SOUPDE, estimator = median)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.77e+00	8.99e-07	1.67e-02	Rejected
2	G-CMA-ES	-3.27e+00	5.43e-04	2.50e-02	Rejected
1	DE	-1.13e+00	1.29e-01	5.00e-02	Accepted

Table 28: Error in 500D

(a) Average, Median, Min and Max Error of SOUPDE

	SOUPDE			
	Average	Median	Minimum	Maximum
F1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F2	$6.50e+01$	$6.50e+01$	$6.28e+01$	$6.78e+01$
F3	$4.71e+02$	$4.67e+02$	$4.65e+02$	$5.12e+02$
F4	$7.96e-02$	$4.55e-12$	$2.73e-12$	$9.95e-01$
F5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F6	$1.67e-13$	$1.64e-13$	$1.57e-13$	$1.82e-13$
F7	$1.78e-13$	$1.78e-13$	$1.78e-13$	$1.78e-13$
F8	$4.36e+04$	$4.38e+04$	$3.64e+04$	$4.72e+04$
F9	$3.59e-05$	$3.59e-05$	$3.59e-05$	$3.59e-05$
F10	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F11	$4.66e-04$	$3.72e-05$	$3.72e-05$	$1.08e-02$
F12	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F13	$3.58e+02$	$3.55e+02$	$3.53e+02$	$3.95e+02$
F14	$1.31e-12$	$1.36e-12$	$9.09e-13$	$1.82e-12$
F15	$1.39e-13$	$1.39e-13$	$1.39e-13$	$1.39e-13$
F16*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F17*	$1.09e+02$	$1.08e+02$	$1.06e+02$	$1.53e+02$
F18*	$2.82e-13$	$2.27e-13$	$2.27e-13$	$4.55e-13$
F19*	$4.95e-14$	$4.95e-14$	$4.95e-14$	$4.95e-14$

(b) Average and Median Error of SOUPDE compared to the reference algorithms

	SOUPDE		DE		CHC		G-CMA-ES	
	Average	Median	Average	Median	Average	Median	Average	Median
F1	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$2.84e-12$	$2.84e-12$	$\mathbf{0.00e+00}$	*0.00e+00*
F2	$6.50e+01$	$6.50e+01$	$5.35e+01$	$5.33e+01$	$1.29e+02$	$1.29e+02$	$\mathbf{3.48e-04}$	*3.31e−04*
F3	$4.71e+02$	$4.67e+02$	$4.76e+02$	$4.74e+02$	$1.14e+06$	$1.14e+06$	$\mathbf{3.58e+02}$	*3.55e+02*
F4	$\mathbf{7.96e-02}$	*4.55e−12*	$3.20e-01$	$9.22e-03$	$1.91e+03$	$1.91e+03$	$2.10e+03$	$2.07e+03$
F5	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$6.98e-03$	$6.98e-03$	$2.96e-04$	*0.00e+00*
F6	$\mathbf{1.67e-13}$	*1.64e−13*	$1.65e-12$	$1.65e-12$	$5.16e+00$	$5.16e+00$	$2.15e+01$	$2.15e+01$
F7	$1.78e-13$	$1.78e-13$	$\mathbf{0.00e+00}$	*0.00e+00*	$1.27e-01$	$1.27e-01$	$7.21e+153$	$2.14e+143$
F8	$4.36e+04$	$4.38e+04$	$6.09e+04$	$6.11e+04$	$7.22e+04$	$7.22e+04$	$\mathbf{2.36e-06}$	*2.31e−06*
F9	$\mathbf{3.59e-05}$	*3.59e−05*	$2.52e+03$	$2.52e+03$	$3.00e+03$	$3.00e+03$	$1.74e+03$	$1.76e+03$
F10	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$1.86e+02$	$1.86e+02$	$1.27e+02$	$1.27e+02$
F11	$\mathbf{4.66e-04}$	*3.72e−05*	$6.76e-04$	$6.71e-04$	$1.81e+03$	$1.81e+03$	$4.16e+03$	$4.18e+03$
F12	$\mathbf{0.00e+00}$	*0.00e+00*	$7.07e-09$	$6.98e-09$	$4.48e+02$	$4.48e+02$	$2.58e+03$	$2.59e+03$
F13	$\mathbf{3.58e+02}$	*3.55e+02*	$3.59e+02$	$3.58e+02$	$3.22e+07$	$3.22e+07$	$2.87e+03$	$2.87e+03$
F14	$\mathbf{1.31e-12}$	*1.36e−12*	$1.35e-01$	$9.01e-07$	$1.46e+03$	$1.46e+03$	$1.95e+03$	$1.95e+03$
F15	$1.39e-13$	$1.39e-13$	$\mathbf{0.00e+00}$	*0.00e+00*	$6.01e+01$	$6.01e+01$	$2.82e+262$	$5.57e+258$
F16*	$\mathbf{0.00e+00}$	*0.00e+00*	$2.04e-08$	$2.05e-08$	$9.55e+02$	$9.55e+02$	$5.45e+03$	$5.43e+03$
F17*	$\mathbf{1.09e+02}$	*1.08e+02*	$1.11e+02$	$1.11e+02$	$8.40e+05$	$8.40e+05$	$9.59e+03$	$9.50e+03$
F18*	$\mathbf{2.82e-13}$	*2.27e−13*	$1.22e-03$	$1.22e-03$	$7.32e+02$	$7.32e+02$	$2.05e+03$	$2.06e+03$
F19*	$4.95e-14$	$4.95e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$1.76e+03$	$1.76e+03$	$2.44e+06$	$2.50e+06$

Table 29: Wilcoxon Signed-Rank test in 500D (reference = SOUPDE, estimator = average)

Optimizer	p-value	Hypothesis
DE	1.404e-02	+
CHC	3.815e-06	+
G-CMA-ES	6.076e-03	+

Table 30: Holm procedure in 500D (reference = SOUPDE, estimator = average)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.52e+00	3.04e-06	1.67e-02	Rejected
2	G-CMA-ES	-3.77e+00	8.17e-05	2.50e-02	Rejected
1	DE	-1.13e+00	1.29e-01	5.00e-02	Accepted

Table 31: Wilcoxon Signed-Rank test in 500D (reference = SOUPDE, estimator = median)

Optimizer	p-value	Hypothesis
DE	1.404e-02	+
CHC	3.815e-06	+
G-CMA-ES	6.970e-03	+

Table 32: Holm procedure in 500D (reference = SOUPDE, estimator = median)

i	Optimizer	z	p	α/i	Hypothesis
3	CHC	-4.40e+00	5.46e-06	1.67e-02	Rejected
2	G-CMA-ES	-3.64e+00	1.34e-04	2.50e-02	Rejected
1	DE	-1.13e+00	1.29e-01	5.00e-02	Accepted

3.3 Scalability Analysis

Tables 38 and 39 present the average and median errors, respectively, obtained by SOUPDE on the test function F1–F19* in 50, 100, 200, 500 and 1000 dimensions. For functions F1, F5, F10, F12 and F16*, the average error remains at 0. On the other function, where SOUPDE did not manage to get "close enough" to the minimum, the error increases with the dimensionality. One must however mention the notable exception of the average error in F14 in 500 dimensions, where the value is much lower than in 200 and 1000 dimensions.

The same comments apply to the table of median errors: the value remains at 0 on functions F1, F5, F10, F12 and F16*, and increases with the dimensionality on the other functions.

3.4 Running Time

The algorithms have been run on a cluster computer of forty nodes, each equipped with four Intel Xeon processors running at 3.0 GHz. The nodes are running Linux. The algorithms were developed in Python 2.4 and the numpy library was used for implementing the test functions.

Table 40 shows the average running time of SOUPDE on each test function, expressed in seconds. Table 41 presents the average runtime of individual test functions, and Table 42 shows the overhead running time related to the algorithm itself, calculated as the difference between SOUPDE's average execution time and the average execution time of the test functions alone.

Table 41 shows that the execution time of the test functions increases non-linearly with the dimensionality, which may be partly attributed to the increased needs of memory allocation which are not necessarily behaving linearly. From the same table one can read that the average overhead of the algorithm is also increases non-linearly with the dimensionality. Given that the nodes of the cluster are not real-time computers, it is difficult to draw conclusions from the variations of the overhead within one dimensionality for different test problems: delays in the computing process may be caused by other causes (e.g., input-output) than the algorithm itself. However, we can observe from Table 42 that the SOUPDE overhead per single fitness call is approximately constant, e.g. for each function the total overhead in 100 dimensions is approximately twice the overhead in 50 dimensions. This feature is important in terms of scalability because, unlike other algorithms (e.g. G-CMA-ES) which make use of algorithmic structure whose computational overhead is heavily biased by the dimensionality of the problem, SOUPDE by itself has essentially the same cost regardless the dimensionality under consideration.

4 Conclusion

This paper proposes a distributed algorithm based on Differential Evolution in order to solve complex large scale optimization problem. The proposed

Table 33: Error in 1000 D

(a) Average, Median, Min and Max Error of SOUPDE

	SOUPDE			
	Average	Median	Minimum	Maximum
F1	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F2	$9.25e+01$	$9.25e+01$	$8.97e+01$	$9.43e+01$
F3	$9.62e+02$	$9.62e+02$	$9.59e+02$	$9.65e+02$
F4	$3.18e-01$	$2.00e-11$	$1.64e-11$	$9.95e-01$
F5	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F6	$3.41e-13$	$3.42e-13$	$3.27e-13$	$3.52e-13$
F7	$3.57e-13$	$3.57e-13$	$3.57e-13$	$3.57e-13$
F8	$2.13e+05$	$2.12e+05$	$2.00e+05$	$2.30e+05$
F9	$7.39e-05$	$7.39e-05$	$7.39e-05$	$7.40e-05$
F10	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F11	$7.44e-05$	$7.44e-05$	$7.44e-05$	$7.44e-05$
F12	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F13	$7.29e+02$	$7.26e+02$	$7.24e+02$	$7.67e+02$
F14	$2.79e-01$	$6.37e-12$	$5.46e-12$	$9.95e-01$
F15	$2.69e-13$	$2.69e-13$	$2.69e-13$	$2.69e-13$
F16*	$0.00e+00$	$0.00e+00$	$0.00e+00$	$0.00e+00$
F17*	$2.31e+02$	$2.31e+02$	$2.29e+02$	$2.32e+02$
F18*	$4.29e-04$	$1.36e-12$	$9.09e-13$	$1.07e-02$
F19*	$9.50e-14$	$9.50e-14$	$9.50e-14$	$9.50e-14$

(b) Average and Median Error of SOUPDE compared to the reference algorithms

	SOUPDE		DE		CHC	
	Average	Median	Average	Median	Average	Median
F1	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$1.36e-11$	$1.36e-11$
F2	$9.25e+01$	$9.25e+01$	$\mathbf{8.46e+01}$	*8.44e+01*	$1.44e+02$	$1.44e+02$
F3	$\mathbf{9.62e+02}$	*9.62e+02*	$9.69e+02$	$9.69e+02$	$8.75e+03$	$8.75e+03$
F4	$\mathbf{3.18e-01}$	*2.00e-11*	$1.44e+00$	$1.32e+00$	$4.76e+03$	$4.76e+03$
F5	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$7.02e-03$	$7.02e-03$
F6	$\mathbf{3.41e-13}$	*3.42e-13*	$3.29e-12$	$3.30e-12$	$1.38e+01$	$1.38e+01$
F7	$3.57e-13$	$3.57e-13$	$\mathbf{0.00e+00}$	*0.00e+00*	$3.52e-01$	$3.52e-01$
F8	$\mathbf{2.13e+05}$	*2.12e+05*	$2.46e+05$	$2.46e+05$	$3.11e+05$	$3.11e+05$
F9	$\mathbf{7.39e-05}$	*7.39e-05*	$5.13e+03$	$5.13e+03$	$6.11e+03$	$6.11e+03$
F10	$\mathbf{0.00e+00}$	*0.00e+00*	$\mathbf{0.00e+00}$	*0.00e+00*	$3.83e+02$	$3.83e+02$
F11	$\mathbf{7.44e-05}$	*7.44e-05*	$1.35e-03$	$1.35e-03$	$4.82e+03$	$4.82e+03$
F12	$\mathbf{0.00e+00}$	*0.00e+00*	$1.68e-08$	$1.70e-08$	$1.05e+03$	$1.05e+03$
F13	$\mathbf{7.29e+02}$	*7.26e+02*	$7.30e+02$	$7.29e+02$	$6.66e+07$	$6.66e+07$
F14	$\mathbf{2.79e-01}$	*6.37e-12*	$6.90e-01$	$9.95e-01$	$3.62e+03$	$3.62e+03$
F15	$2.69e-13$	$2.69e-13$	$\mathbf{0.00e+00}$	*0.00e+00*	$8.37e+01$	$8.37e+01$
F16*	$\mathbf{0.00e+00}$	*0.00e+00*	$4.18e-08$	$4.19e-08$	$2.32e+03$	$2.32e+03$
F17*	$\mathbf{2.31e+02}$	*2.31e+02*	$2.36e+02$	$2.35e+02$	$2.04e+07$	$2.04e+07$
F18*	$\mathbf{4.29e-04}$	*1.36e-12*	$2.37e-03$	$2.37e-03$	$1.72e+03$	$1.72e+03$
F19*	$9.50e-14$	$9.50e-14$	$\mathbf{0.00e+00}$	*0.00e+00*	$4.20e+03$	$4.20e+03$

Table 34: Wilcoxon Signed-Rank test in 1000D (reference = SOUPDE, estimator = average)

Optimizer	p-value	Hypothesis
DE	1.404e-02	+
CHC	3.815e-06	+

Table 35: Holm procedure in 1000D (reference = SOUPDE, estimator = average)

i	Optimizer	z	p	α/i	Hypothesis
2	CHC	-5.03e+00	2.47e-07	2.50e-02	Rejected
1	DE	-1.30e+00	9.72e-02	5.00e-02	Accepted

Table 36: Wilcoxon Signed-Rank test in 1000D (reference = SOUPDE, estimator = median)

Optimizer	p-value	Hypothesis
DE	1.404e-02	+
CHC	1.386e-04	+

Table 37: Holm procedure in 1000D (reference = SOUPDE, estimator = median)

i	Optimizer	z	p	α/i	Hypothesis
2	CHC	-5.03e+00	2.47e-07	2.50e-02	Rejected
1	DE	-1.30e+00	9.72e-02	5.00e-02	Accepted

Table 38: Scalability analysis on average error (SOUPDE)

	50	100	200	500	1000
F1	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F2	1.18e+00	7.47e+00	2.38e+01	6.50e+01	9.25e+01
F3	3.10e+01	7.92e+01	1.80e+02	4.71e+02	9.62e+02
F4	3.98e-02	3.98e-02	1.19e-01	7.96e-02	3.18e-01
F5	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F6	1.47e-14	3.03e-14	6.40e-14	1.67e-13	3.41e-13
F7	2.28e-14	3.88e-14	7.46e-14	1.78e-13	3.57e-13
F8	9.69e-02	6.55e+01	2.46e+03	4.36e+04	2.13e+05
F9	3.75e-06	7.82e-06	1.51e-05	3.59e-05	7.39e-05
F10	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F11	3.09e-06	6.75e-06	1.43e-05	4.66e-04	7.44e-05
F12	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F13	2.06e+01	5.85e+01	1.35e+02	3.58e+02	7.29e+02
F14	0.00e+00	9.09e-15	3.98e-02	1.31e-12	2.79e-01
F15	1.38e-14	2.79e-14	5.79e-14	1.39e-13	2.69e-13
F16*	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F17*	2.53e-01	8.55e+00	3.31e+01	1.09e+02	2.31e+02
F18*	0.00e+00	0.00e+00	0.00e+00	2.82e-13	4.29e-04
F19*	0.00e+00	0.00e+00	1.91e-14	4.95e-14	9.50e-14

Table 39: Scalability analysis on median error (SOUPDE)

	50	100	200	500	1000
F1	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F2	6.73e-01	7.01e+00	2.33e+01	6.50e+01	9.25e+01
F3	2.37e+01	7.30e+01	1.71e+02	4.67e+02	9.62e+02
F4	0.00e+00	0.00e+00	2.27e-13	4.55e-12	2.00e-11
F5	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F6	1.47e-14	2.89e-14	6.44e-14	1.64e-13	3.42e-13
F7	2.28e-14	3.88e-14	7.46e-14	1.78e-13	3.57e-13
F8	8.42e-02	6.52e+01	2.46e+03	4.38e+04	2.12e+05
F9	3.75e-06	7.82e-06	1.51e-05	3.59e-05	7.39e-05
F10	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F11	3.09e-06	6.75e-06	1.43e-05	3.72e-05	7.44e-05
F12	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F13	2.01e+01	5.83e+01	1.32e+02	3.55e+02	7.26e+02
F14	0.00e+00	0.00e+00	2.27e-13	1.36e-12	6.37e-12
F15	1.38e-14	2.79e-14	5.79e-14	1.39e-13	2.69e-13
F16*	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.00e+00
F17*	7.35e-02	8.74e+00	3.30e+01	1.08e+02	2.31e+02
F18*	0.00e+00	0.00e+00	0.00e+00	2.27e-13	1.36e-12
F19*	0.00e+00	0.00e+00	1.91e-14	4.95e-14	9.50e-14

Table 40: Average execution time of SOUPDE (in seconds)

	50	100	200	500	1000
F1	115	248	546	1768	4922
F2	117	252	558	1810	5032
F3	153	328	713	2211	5829
F4	129	284	637	2066	5812
F5	132	289	648	2113	5889
F6	134	298	658	2096	5695
F7	119	262	583	1879	5216
F8	104	231	514	1671	4686
F9	156	356	826	2916	8877
F10	163	355	783	2462	6739
F11	159	360	831	2933	8888
F12	265	585	1339	4570	13253
F13	304	675	1510	5026	14451
F14	284	629	1437	4886	14343
F15	288	635	1430	4727	13577
F16*	264	592	1367	4727	14042
F17*	308	684	1561	5378	15691
F18*	281	615	1443	5039	15386
F19*	271	609	1362	4402	10905

algorithm consists of a Differential Evolution employing a structured population where each sub-population evolves independently, for a portion of the optimization process, by quickly exploiting the set of available search moves and thus enhancing upon the original fitness values. Two randomized mechanisms periodically activated allow interaction among sub-populations and generation of unexplored search logics. The resulting algorithm behaves in a similar way to a multi-start multiple search and proved to be efficient for highly dimensional problems.

Table 41: Average execution time for the test functions (in seconds)

	50	100	200	500	1000
F1	25	51	111	329	783
F2	28	57	122	363	883
F3	65	137	285	797	1807
F4	44	97	217	755	2231
F5	47	104	240	903	2639
F6	52	114	251	823	2364
F7	32	70	149	441	1080
F8	21	46	98	287	696
F9	75	172	435	1846	6168
F10	79	178	403	1381	4014
F11	74	175	435	1853	6122
F12	173	390	906	3233	9652
F13	213	472	1072	3694	10537
F14	192	432	1012	3638	10817
F15	196	435	988	3414	9713
F16*	172	400	943	3459	10574
F17*	213	486	1136	4144	12375
F18*	191	444	1055	3986	12165
F19*	194	439	990	3415	8948

Table 42: Average overhead of SOUPDE (in seconds)

	50	100	200	500	1000
F1	90	196	435	1439	4139
F2	89	194	436	1447	4148
F3	88	191	428	1413	4021
F4	84	187	420	1311	3580
F5	85	185	408	1210	3249
F6	82	184	407	1273	3330
F7	86	191	433	1437	4135
F8	82	185	415	1384	3990
F9	81	184	390	1070	2709
F10	83	177	380	1080	2724
F11	84	185	395	1079	2765
F12	91	195	432	1336	3601
F13	90	202	438	1331	3913
F14	91	197	424	1248	3526
F15	92	200	441	1313	3864
F16*	92	191	424	1268	3468
F17*	94	198	424	1233	3315
F18*	90	170	387	1052	3220
F19*	76	169	371	987	1957

Two important features of the proposed algorithm must be highlighted. The first is the capability of detecting the theoretical global optimum in most of the runs and for most of the problems in the benchmark. The second feature is the robustness to the curse of dimensionality with respect to the other algorithms considered in the study. More specifically, the standard DE in the benchmark has a performance comparable to the proposed algorithm for low dimensional cases. While the dimensionality grows the proposed algorithm

appears to be more capable to handle the fitness landscapes and for the set of most multi-variate problems displays a significantly better performance.

References

1. F. van den Bergh and A. P. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 3, pp. 225–239, 2004.
2. E. Marchiori and A. Steenbeek, "An evolutionary algorithm for large scale set covering problems with application to airline crew scheduling," in *Scheduling, in Real World Applications of Evolutionary Computing*, Lecture Notes in Computer Science, pp. 367–381, Springer, 2000.
3. 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*, pp. 2366–2373, 2007.
4. 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*, pp. 3906–3913, 2008.
5. P. Korošec and J. Šilc, "The differential ant-stigmergy algorithm for large scale real-parameter optimization," in *ANTS '08: Proceedings of the 6th international conference on Ant Colony Optimization and Swarm Intelligence*, Lecture Notes in Computer Science, pp. 413–414, Springer, 2008.
6. M. A. Potter and K. A. De Jong, "A cooperative coevolutionary approach to function optimization," in *Proceedings of the Third Conference on Parallel Problem Solving from Nature*, pp. 249–257, Springer-Verlag, 1994.
7. Y. Liu and Q. Zhao, "Scaling up fast evolutionary programming with cooperative coevolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1101–1108, 2001.
8. D. Sofge, K. De Jong, and A. Schultz, "A blended population approach to cooperative coevolution fordecomposition of complex problems," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 413–418, 2002.
9. M. A. Potter and K. De Jong, "Cooperative coevolution: An architecture for evolving coadapted subcomponents," *Evolutionary Computation*, vol. 8, no. 1, pp. 1–29, 2000.
10. Y.-J. Shi, H.-F. Teng, and Z.-Q. Li, "Cooperative co-evolutionary differential evolution for function optimization," in *Advances in Natural Computation*, vol. 3611 of *Lecture Notes in Computer Science*, pp. 1080–1088, Springer, 2005.
11. Z. Yang, K. Tang, and X. Yao, "Differential evolution for high-dimensional function optimization," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 3523–3530, 2007.
12. A. Zamuda, J. Brest, B. Bošković, and V. Žumer, "Large scale global optimization using differential evolution with self-adaptation and cooperative coevolution," in *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 3719–3726, 2008.
13. O. Olorunda and A. Engelbrecht, "Differential evolution in high-dimensional search spaces," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 1934–1941, 2007.
14. Z. Yang, K. Tang, and X. Yao, "Large scale evolutionary optimization using cooperative coevolution," *Information Sciences*, vol. 178, no. 15, pp. 2985–2999, 2008.
15. U. K. Chakraborty, ed., *Advances in Differential Evolution*, vol. 143 of *Studies in Computational Intelligence*. Springer, 2008.
16. F. Neri and V. Tirronen, "Recent advances in differential evolution: A review and experimental analysis," *Artificial Intelligence Review*, vol. 33, no. 1, pp. 61–106, 2010.

17. S. Rahnamayan and G. G. Wang, "Solving large scale optimization problems by opposition-based differential evolution (ode)," *WSEAS Transactions on Computers*, vol. 7, no. 10, pp. 1792–1804, 2008.
18. S. Rahnamayan, H. R. Tizhoosh, and M. M. Salama, "Opposition-based differential evolution," *IEEE Transactions on Evolutionary Computation*, vol. 12, no. 1, pp. 64–79, 2008.
19. N. Noman and H. Iba, "Enhancing differential evolution performance with local search for high dimensional function optimization," in *Proceedings of the 2005 conference on Genetic and evolutionary computation*, pp. 967–974, ACM, 2005.
20. P. Moscato and M. Norman, "A competitive and cooperative approach to complex combinatorial search," Tech. Rep. 790, 1989.
21. W. E. Hart, N. Krasnogor, and J. E. Smith, "Memetic evolutionary algorithms," in *Recent Advances in Memetic Algorithms* (W. E. Hart, N. Krasnogor, and J. E. Smith, eds.), (Berlin, Germany), pp. 3–27, Springer, 2004.
22. 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.
23. 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.
24. J. Brest, A. Zamuda, B. Bošković, M. S. Maucec, and V. Žumer, "High-dimensional real-parameter optimization using self-adaptive differential evolution algorithm with population size reduction," in *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 2032–2039, 2008.
25. J. Brest and M. S. Maučec, "Population size reduction for the differential evolution algorithm," *Applied Intelligence*, vol. 29, no. 3, pp. 228–247, 2008.
26. J. Brest, B. Bošković, S. Greiner, V. Žumer, and M. S. Maučec, "Performance comparison of self-adaptive and adaptive differential evolution algorithms," *Soft Computing*, vol. 11, no. 7, pp. 617–629, 2007.
27. 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.
28. F. Neri and V. Tirronen, "Scale factor local search in differential evolution," *Memetic Computing Journal*, vol. 1, no. 2, pp. 153–171, 2009.
29. D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution," in *Proceedings of the IEEE Congress on Evolutionary Computation*, pp. 2023–2029, 2004.
30. J. Apolloni, G. Leguizamón, J. García-Nieto, and E. Alba, "Island based distributed differential evolution: An experimental study on hybrid testbeds," in *Proceedings of the IEEE International Conference on Hybrid Intelligent Systems*, pp. 696–701, 2008.
31. M. Weber, F. Neri, and V. Tirronen, "Distributed differential evolution with explorative-exploitative population families," *Genetic Programming and Evolvable Machines*, vol. 10, no. 4, pp. 343–371, 2010.
32. M. Weber, V. Tirronen, and F. Neri, "Scale factor inheritance mechanism in distributed differential evolution," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*. to appear.
33. V. Feoktistov, *Differential Evolution in Search of Solutions*. Springer, 2006.
34. 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.), pp. 76–83, 2000.
35. S. García, A. Fernández, 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. S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the cec'2005 special session on real parameter optimization," *Journal of Heuristics*, vol. 15, no. 6, pp. 617–644, 2008.

37. F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.
38. S. Holm, "A simple sequentially rejective multiple test procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979.