Differerential Evolution methods based on local searches

Marco Locatelli

Dip. Ingegneria dell'Informazione, Univ. di Parma (Italy)

Mirko Maischberger

Dip. Sistemi e Informatica, Univ. di Firenze (Italy)

Fabio Schoen

Dip. Sistemi e Informatica, Univ. di Firenze (Italy)

Abstract

In this paper we analyze the behavior of a quite standard Differential Evolution (DE) algorithm applied to the objective function transformed by means of local searches. First some surprising results are presented which concern the application of this method to standard test functions.

Later we introduce an application to disk- and to sphere-packing problems, two well known and particularly hard global optimization problems. For these problems some more refined variations of the basic method are necessary in order to take at least partially into considerations the many symmetries those problems possess. Coupling these techniques with DE and local optimization resulted in a new method which, when tested on moderately sized packing problems, was capable of confirming known putative optima for the problem of packing disks, and of discovering quite a significant number of new putative optima for the problem of packing spheres.

Keywords: Global optimization, population-based methods, memetic algorithms, disk and sphere packing

1. Introduction and problem statement

Differential Evolution (DE) is a very simple population-based global optimization algorithm. The original DE method (see [1]), can be described as in Algorithm 1, where $\mathcal{U}(S)$ represents a uniform random number generator in the set S.

Email addresses: `locatelli@ce.unipr.it` (Marco Locatelli), `maischberger@dsi.unifi.it` (Mirko Maischberger), `fabio.schoen@unifi.it` (Fabio Schoen)

```
Data: $F \in (0,2)$, a real constant; $CR \in [0,1]$, a probability threshold
foreach $i \in 1,\dots,p$ do
    let $\bar{\imath} := \mathcal{U}(1,\dots,n)$;
    randomly choose $k_1, k_2, k_3 \in \{1,\dots,p\} \setminus \{i\}$, all different;
    let $Trial := x_{k_1} + F(x_{k_2} - x_{k_3})$;
    for $j = 1,\dots,n : j \neq \bar{\imath}$ do
        if $\mathcal{U}(0,1) < CR$ then
            let $Trial^{(j)} := x_i^{(j)}$;
        end
    end
    if $f(Trial) < f(x_i)$ then
        let $x_i := Trial$;
    end
end
```

Algorithm 1: Differential Evolution

The idea of the algorithm can be easily seen within the context of Genetic Algorithms:

- the algorithm maintains and evolves a whole population of solutions;
- a sort of *crossover* can be performed (if $CR > 0$), where a new individual is formed by substituting some of its components with a linear combination of three other members in the population;
- population overall improvement is implemented through an acceptance criterion by which a new trial solution is accepted and replaces a current individual if the associated function value improves.

The method reported in Algorithm 1 is one of the possible variants of DE; different implementations of the basic scheme can be identified thanks to a simple taxonomy which takes the form of $DE/x/y/z$. Here x, which takes the values *rand* or *best*, represents the criterion by which the solution to be perturbed is chosen (a solution chosen at random using a uniform distribution, or the current best member of the population). The second field, y, stands for the number of difference vectors used in the definition of the $Trial$ solution, where a difference vector is the difference between two distinct randomly selected elements in the population. Finally, z identifies the crossover operator – its value represents the discrete probability distribution used to generate the individuals which participate to the crossover; its value can be *bin* indicating that the choice is made following a binomial probability distribution function, as in the scheme reported in Algorithm 1, or *exp* when the choice on which components to keep from the current population is based on an exponential distribution. The strategy outlined in Algorithm 1 can be classified as DE/rand/1/bin. We refer to the existing literature for a description of its many variants (see, in particular, [2]).

The theoretical aspects of DE have received only moderate attention in the literature. In [3], under mild conditions, convergence of the whole population to a single point is proven for strictly convex objective functions, although such point is not guaranteed to be the local (global) minimum. In fact, for more general functions some phenomena may occur, like, e.g., the population collapsing to a single point or a frozen population (due to the finite number of possible moves, none of them might be improving with respect to the current members of the population), which prevent further progress of the population and thus the convergence to a global minimum (see, e.g., [4, 5]).

Despite this lack of theoretical support, the practical behavior of the method is quite interesting and the algorithm is used in a very large number of cases.

As it is common in many global optimization heuristics (see, e.g., [6]), the algorithm switches from an initial, exploratory, phase to a local refinement one where members of the population group together. While this behavior is needed in most good heuristics, it should be recalled that, when it comes to local refinement, it is usually much more efficient to base the search on standard local optimization methods, instead of performing local steps without exploiting the power of local descent methods.

In this paper we first explore a quite simple modification of the basic DE scheme, following the approach which, in combinatorial optimization, goes under the name of *memetic*: in Algorithm 1, each time the evaluation of the objective function f is required, we instead perform a local descent by means of a local optimization method $\mathscr{L}$ which, given an objective function f and a starting point x returns $\bar{x} = \mathscr{L}(f, x) \in \mathbb{R}^n$, a local optimum for f where $f(\bar{x}) \leq f(x)$. Note that memetic algorithms proved to be efficient ones as testified , e.g., by the fact that a memetic algorithm (see [7]) won the CEC competition 2010 in large scale global optimization (see `http://sci2s.ugr.es/eamhco/cec2010_functions.pdf`). The basic algorithm is transformed as represented in Algorithm 2 (MDE).

Although in this algorithm only a single line is changed with respect to Algorithm 1, the overall behavior of the whole method radically changes, as it transforms a continuous, derivative free, method like DE into a combinatorial search in the space of local minima – we remark here that in this implementation all members x_i in the population are local optima of the objective function.

The original contribution in this paper can be found in two aspects:

- first, we analyze in a quite systematic way the behavior of Algorithm 2 when applied to quite standard test functions and we comment on the somewhat surprising results (not previously observed in the literature, to the authors' knowledge), for which we give some (tentative) explanations;

- then, after introducing some specific modifications, we show how the method can be applied, with quite unexpected good results, on classical problems of packing disks in a square or spheres in a cube.

The results obtained and described in this paper are, in our opinion, interesting and further research will be surely needed in order to fully understand the capabilities of this method.

```
Data: $F \in (0,2)$, a real constant; $CR \in [0,1]$, a probability threshold
foreach $i \in 1,\dots,p$ do
    let $\bar{\imath} := \mathcal{U}(1,\dots,n)$;
    randomly choose $k_1, k_2, k_3 \in \{1,\dots,p\} \setminus \{i\}$, all different;
    let $Trial := x_{k_1} + F(x_{k_2} - x_{k_3})$;
    for $j = 1,\dots,n : j \neq \bar{\imath}$ do
        if $\mathcal{U}(0,1) < CR$ then
            let $Trial^{(j)} := x_i^{(j)}$;
        end
    end
    let $Trial := \mathscr{L}(f, Trial)$;
    if $f(Trial) < f(x_i)$ then
        let $x_i := Trial$;
    end
end
```
Algorithm 2: Memetic Differential Evolution (MDE)

The paper is structured as follows. In Section 2 we shortly describe Monotonic Basin Hopping, a simple but efficient global optimization approach based on local searches, which represents a quite natural basis for comparison for MDE. In Section 3 we describe and interpret some computational experiments with MBH and a basic version of MDE on standard global optimization functions and some, more challenging, variants of these functions. In Section 4 we describe some experiments with MDE on packing problems. For these problems variations of the basic MDE method are required in order to take at least partially into consideration the many symmetries those problems possess and, thus, the many equivalent solutions existing for this kind of problems. Finally, in Section 5 we draw some conclusions.

2. A short introduction to Monotonic Basin Hopping

In order to evaluate the behavior of the proposed algorithm, we compared its performance with a standard global optimization algorithm, Monotonic Basin Hopping, MBH, (see, e.g., [8, 6, 9]), whose general scheme is reported in Algorithm 3. This algorithm is strongly based on the application of local searches. It is an Iterated Local Search method in the space of local optima.

It consists in repeatedly performing a local optimization from a point which is randomly, uniformly, generated in a prescribed neighborhood of the current iterate. Here we choose, as a neighborhood, $D(x, \Delta)$, the hypercube centered at x with edge length equal to 2Δ, but other choices for the shape of the neighborhood, like, e.g., hyperrectangles, spheres or ellipsoids, are possible.

In all our experiments, MBH was run with a stopping criterion based on

```
while $GlobalStoppingRule$ $is$ $false$ do
    let $x := \mathcal{U}(S)$;
    let $x := \mathscr{L}(f, x)$;
    let $k := 0$;
    while $k < MaxNoImprove$ do
        let $z := \mathcal{U}(D(x, \Delta))$ ;
        let $y := \mathscr{L}(f, z)$;
        if $f(y) < f(x)$ then
            let $x := y$;
        end
    end
end
```

Algorithm 3: Basin Hopping method

$MaxNoImprove = 1\,000$ iterations without improvements. The unique parameter of MBH is thus Δ, called the radius of the neighborhood.

As it can be seen, both methods are based on local searches; DE is population-based, while this version of MBH is a sequential one. MBH is known to be a quite efficient algorithm for rapidly exploring *funnel bottoms*, where the concept of funnel, described, e.g., in [8, 9, 10], derives from computational chemistry. In global optimization problems, given a neighborhood structure and a local search, it is theoretically possible to build a graph whose nodes are local optima and whose arcs connect a local minimum with a better one if the latter may be obtained by means of a run of local search started in a neighborhood of the former. In this graph, nodes with no outgoing arcs are called *funnel bottoms*, while a *funnel* is composed of all nodes on all directed paths which lead to the same funnel bottom. As it should be clear from these definitions, MBH is *the* algorithm of choice for efficiently exploring funnel bottoms. However, its efficiency strongly depends on the neighborhood chosen: if it is too narrow, every local minimum becomes a funnel bottom, and MBH reduces to a basic Multistart method. On the opposite side, if the neighborhood is too large, only the global optimum is a funnel bottom, but finding a path which ends at the global minimum is very hard and, again, MBH behaves like a pure Multistart method. Choosing the correct neighborhood size is crucial in MBH and strongly determines its efficiency.

On the contrary, in MDE no choice is required for the neighborhood size and shape. One of the main contributions of this paper will be to show, through extensive computational experiments, that the automatic coordination of elements within the population enables the method to rapidly explore funnels without the necessity of defining a suitable neighborhood. Moreover, while MBH performs quite well when the function has a single or few funnel bottoms (with respect to a reasonably chosen neighborhood), its performance strongly deteriorates as soon as the number of funnel bottoms increases. Apparently, MDE is able to overcome or at least to limit the negative consequences of a high number of funnel bottoms: it appears to be able to perform a descent search even within the space of funnel bottoms, as we will more clearly see in the next section about the computational experiments on standard test functions.

3. DE on standard test functions

The basic algorithm was run with the following choices for parameters:

- $F = 0.5$
- $CR = 0$ - this way no crossover is performed. The reason for this choice is due to the desire of testing the new method in the most unfavorable situation. Indeed, many classical test functions are separable. Using the crossover operator for such functions might induce strong improvements due to the capability of the method of optimizing over subsets of variables. By forbidding crossover, our tests were run in such a way as to obtain no advantage from separability.
- the stopping criterion was chosen this way: the algorithm was stopped as soon as one of the following conditions is satisfied:
  1. the sum of the differences in function value between the members of the population falls below a threshold (10^{-4} in the experiments). This is an indirect way to check that the population has shrinked into a single point. Actually, as only function values are checked, more than one point cluster might exist. This rule, by far, was the most common reason for stopping in all the experiments performed.
  2. no change has been observed in the population (more precisely, in the objective values associated to population elements) during the last 100 executions of the algorithm.
  3. the best observed value did not change during the last 20 000 local searches perforfmed.

Moreover, in the algorithm we allowed for the possibility that k_1, k_2, k_3 are not distinct, which, in fact, does not significatively modify the performance. Given these choices, the unique parameter of the method is the population size p.

For the computational experiments we used some classical highly multimodal test functions together with some more challenging variants of such functions. The classical functions that we used are the following.

- Rastrigin function:

$$f_1(\mathbf{x}) = 10n + \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i)), \quad \mathbf{x} \in [-5.12, 5.12]^n,$$

  whose global minimum is $\mathbf{x}^* = \mathbf{0}$ and the global minimum value is 0.

- Ackley function:

$$f_2(\mathbf{x}) = 20 + e - 20\exp\left(-0.2\sqrt{\tfrac{1}{n}\textstyle\sum_{i=1}^n x_i^2}\right) -$$
$$- \exp\left(\tfrac{1}{n}\textstyle\sum_{i=1}^n \cos(2\pi x_i)\right), \quad \mathbf{x} \in [-32.768, 32.768]^n$$

  whose global minimum is $\mathbf{x}^* = \mathbf{0}$ and the global minimum value is 0.

- Levy function:

$$f_3(\mathbf{x}) = \sin^2\left(\pi\left(\tfrac{x_1+3}{4}\right)\right) + \textstyle\sum_{i=1}^{n-1}\left(\tfrac{x_i-1}{4}\right)^2\left(1 + 10\sin^2\left(\pi\left(\tfrac{x_i+3}{4}\right)+1\right)\right) +$$
$$+ \left(\tfrac{x_n-1}{4}\right)^2\left(1+\sin^2\left(2\pi x_n\right)\right), \quad \mathbf{x} \in [-10, 10]^n$$

  whose global minimum is $\mathbf{x}^* = (1, \ldots, 1)$ and the global minimum value is 0.

- Sinusoidal function:

$$f_4(\mathbf{x}) = -2.5\textstyle\prod_{i=1}^n \sin\left(x_i - \tfrac{\pi}{6}\right) - \textstyle\prod_{i=1}^n \sin\left(21\left(x_i - \tfrac{\pi}{6}\right)\right) + 3.5$$
$$\mathbf{x} \in [0, \pi]^n$$

  whose global minimum is $\mathbf{x}^* = \left(\frac{2\pi}{3}, \ldots, \frac{2\pi}{3}\right)$ and the global minimum value is 0.

- Schwefel function:

$$f_5(\mathbf{x}) = \sum_{i=1}^n -x_i \sin\left(\sqrt{|x_i|}\right), \qquad \mathbf{x} \in [-500, 500]^n$$

  whose global minimum is $\mathbf{x}^* = (420.9687, \ldots, 420.9687)$ and the global minimum value is $-418.9829n$.

The variants we considered are the functions

$$f_i(\mathbf{DW}(\mathbf{x} - \bar{\mathbf{x}})), \quad i = 1\ldots, 5,$$

where

- $\mathbf{D}$ is a diagonal matrix of order n with positive diagonal elements;
- $\mathbf{W}$ is an orthonormal matrix of order n;
- $\bar{\mathbf{x}}$ is a n-dimensional shift vector.

The classical functions are obtained when

$$\mathbf{D} = \mathbf{W} = \mathbf{I}, \quad \bar{\mathbf{x}} = \mathbf{0},$$

where $\mathbf{I}$ is the identity matrix of order n. Through choices of $\mathbf{D}, \mathbf{W}$ and $\bar{\mathbf{x}}$ different from the above ones, some sources of difficulty are introduced. The orthonormal matrix $\mathbf{W}$ introduces a rotation of the space, so that the objective function is no more separable (as it is the case for f_1, f_3 and f_5). The matrix $\mathbf{D}$ introduces some variability in the scaling of the variables. For functions f_1 and f_2 the global minimum lies at the origin, i.e., at the center of the search space: a shift vector $\bar{\mathbf{x}} \neq \mathbf{0}$ moves the global minimum away from the center of the search space. Note that also the search space is rotated, i.e., the feasible region is the polytope

$$X = \{\mathbf{x} \ : \ \ell_i \leq \mathbf{W}\mathbf{x} \leq u_i\}, \quad i = 1, \ldots, 5,$$

where ℓ_i, u_i denotes the lower and upper bound for the box constraints of the basic function f_i, $i = 1, \ldots, 5$. The shift vector $\bar{\mathbf{x}} \in X$ is generated by first randomly drawing a vector $\mathbf{z} \in [\ell_i, u_i]^n$, and then by setting $\bar{\mathbf{x}} = \mathbf{W}^T\mathbf{z}$.

All these functions have many local minima, which tend to increase exponentially as the dimension n increases. Functions f_1-f_4 have a single funnel bottom (with respect to a reasonably small neighborhood), while function f_5 is the most challenging function with exponentially many (2^n) funnel bottoms (once again, for reasonable sizes of the neighborhood).

In each table reporting the computational results we have the following columns:

Function : identifies the test function;

Alg.(par.) : identifies the algorithm (MBH or MDE) and the parameter value (Δ for MBH, p for MDE);

% **succ.** : denotes the percentage of successes over the set of random runs;

Av.diff.fail. : denotes the average distance between the returned value and the global optimum value in those runs where a failure occurred;

Av. # LS : average number of local searches per run;

Av. # LS per succ. : average number of local searches per successful run;

Note that each run of an algorithm has been stopped as soon as the global optimum value has been observed.

We have used the local solver MINOS [11] for all tests, with the exception of function f_1, where we also performed tests with the local solver SNOPT [12] for reasons which will be discussed in detail in Section 3.1.

For what concerns MBH, we tested different values for the parameter Δ but we usually only report the results with the best possible choice. Sometimes we report also the results for parameter values close to the "best"ones in order to give an idea about the sensitivity of MBH with respect to the choice of Δ.

We usually perform 100 runs of each method over each test function. Some exceptions, which will be clearly indicated later on, are made for the 50-dimensional versions of the test functions, on one hand because these cases require much larger computational times, on the other hand because the outcomes of a lower number of runs are already clear enough to draw conclusions about the behaviors of different methods.

3.1. Rastrigin

We first considered three versions of the Rastrigin function:

- Rastrigin$n$: where $\mathbf{D} = \mathbf{W} = \mathbf{I}$, $\bar{\mathbf{x}} = \mathbf{0}$, which is the original separable function;

- RastriginRot$n$: where $\mathbf{D} = \mathbf{I}$, $\mathbf{W}$ randomly generated orthonormal matrix, $\bar{\mathbf{x}} = \mathbf{0}$;

- RastriginShift$n$: where $\mathbf{D} = \mathbf{I}$, $\mathbf{W}$ randomly generated orthonormal matrix, $\bar{\mathbf{x}}$ randomly generated within $[-5.12, 5.12]^n$.

We considered the dimensions $n = 10$ and $n = 50$. We first performed some random tests with the local solver MINOS (100 random runs for all functions but RastriginShift50 for which we only performed 10 random runs). The results are reported in Table 1. With the same set of test functions we have also performed some experiments with the local solver SNOPT (100 random runs for $n = 10$, 10 random runs for $n = 50$). The results are reported in Table 2. Overall the performance of MDE is usually comparable with that of MBH, but the comparison of the results with the two local solvers leads to some interesting observations. We notice that the choice of the local solver has a clear impact and, in particular, we have the following.

- The best choice of the parameter Δ for MBH may strongly depend on the local solver. Indeed, for the separable functions Rastrigin$n$ the best possible value for MINOS is very low (0.003), while it is much larger (0.6) for SNOPT (using the value 0.003 with SNOPT, MBH is unable to escape from the starting local minimum). Instead, for the other versions of the Rastrigin function, namely RastriginRot and RastriginShift, the best possible choice of the parameter value is rather similar. An explanation of this fact can be given after looking into the behavior of MINOS over separable functions. In this case, MINOS perform steps along a single axis, called minor iterations, whose length depends on the size of the corresponding component of the gradient. It turns out that, due to the high oscillations of the Rastrigin function, such steps might be rather long and jump over many local minima along an axis direction even after a very small perturbation of the current local minimum (the same does not hold, e.g., for the Schwefel function, which is also separable but whose gradient values close to a local minimum are not enough to jump towards other local minima). Such behavior shows how, in some cases, separability can be, more or less explicitly, exploited in order to strongly enhance the performance of a method, thus making questionable to draw conclusions from experimentations solely based on separable test functions.

- MDE appears to perform much more poorly with respect to MBH when MINOS is used over the RastriginRot functions (the situation is clearly different when SNOPT is used). However, it is worthwhile to notice that in spite of the large number of failures (e.g., 81 and 61 respectively for MDE(10) and MDE(20) when applied to RastriginRot10), the `Av.diff.fail.` value is very low (respectively, 1.1 and 0.99) and very close to (in fact, for MDE(20) exactly equal to) the second best local minimum value.

Next we have also considered further variants of the functions above, where the diagonal entries of $\mathbf{D}$ are randomly generated in $[1, 4]$. In this case the test functions become more challenging. In Table 3 we report the results with the local solver MINOS (100 random runs for $n = 10$, 10 random runs for $n = 50$). For the separable versions Rastrigin10 and Rastrigin50 MBH appears to work better (much better for Rastrigin50, although MDE tends to stop not too far away from the global minimum value when it fails). We refer to the previous discussion for an explanation of the very good behavior of MBH with the separable version. But for the RastriginRot and RastriginShift functions the situation is radically different with MBH unable to detect the global minimum value for both versions when $n = 50$. Something similar also occurs with the separable version when the local solver SNOPT is used: MBH is unable to detect the global minimum value of Rastrigin50, while MDE(20) detects it in 7 out of 10 runs and MDE(50) always detects it (respectively, with 1063 and 2645 local searches *per success*). In fact, the behavior observed with the local solver SNOPT is the one we expected. Indeed, once we introduce a different scaling of the variables through the matrix $\mathbf{D}$, it is unlikely that a single perturbation value Δ allows to define a suitable neighborhood. It would make more sense to have a different perturbation Δ_i, $i = 1, \ldots, n$, for each variable, but finding an appropriate choice of such values is not a trivial task. Instead, it seems that MDE is able to self-adjust the step length in each direction.

3.2. Ackley

We considered three versions of the Ackley functions, denoted as follows:

- Ackley$n$: where $\mathbf{D} = \mathbf{W} = \mathbf{I}$, $\bar{\mathbf{x}} = \mathbf{0}$;
- AckleyRot$n$: where $\mathbf{D} = \mathbf{I}$, $\mathbf{W}$ randomly generated orthonormal matrix, $\bar{\mathbf{x}} = \mathbf{0}$;
- AckleyShift$n$: where $\mathbf{D} = \mathbf{I}$, $\mathbf{W}$ randomly generated orthonormal matrix, $\bar{\mathbf{x}}$ randomly generated within $[-32.768, 32.768]^n$.

We considered the dimensions $n = 10$ and $n = 50$. We performed 100 random runs with each algorithm. While having a single funnel bottom, so that MBH works well with these functions, the distances between local minima tend to decrease as we approach the global minimum. That makes the use of a single Δ value not advisable: too small a value causes a slow progress during the first iterations, whereas a large value makes the algorithm slow in detecting the global minimum once we are close to it. For this reason, in the experiments we

Table 1: Tests with the Rastrigin function using the local solver MINOS

Function	Alg.(par.)	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
Rastrigin10	MBH(0.05)	100	0	284.3	284.3
Rastrigin10	MBH(0.003)	100	0	54	54
Rastrigin10	MDE(10)	97	0.99	57.2	59
Rastrigin10	MDE(20)	100	0	116	116
Rastrigin50	MBH(0.002)	0	209.6	1837	∞
Rastrigin50	MBH(0.003)	100	0	80	80
Rastrigin50	MBH(0.005)	100	0	102.1	102.1
Rastrigin50	MDE(20)	100	0	226	226
Rastrigin50	MDE(50)	100	0	610	610
RastriginRot10	MBH(0.3)	32	1.68	2442	7631
RastriginRot10	MBH(0.4)	100	0	318.1	318.1
RastriginRot10	MBH(0.5)	100	0	149.5	149.5
RastriginRot10	MBH(0.6)	100	0	183.6	183.6
RastriginRot10	MDE(10)	19	1.1	169.4	891
RastriginRot10	MDE(20)	39	0.99	377.8	968
RastriginRot50	MBH(0.3)	80	0.99	3133.8	3917
RastriginRot50	MBH(0.4)	90	0.99	1291.5	1435
RastriginRot50	MBH(0.5)	20	1.86	3174.8	15874
RastriginRot50	MDE(20)	30	1.36	888	4440
RastriginRot50	MDE(50)	20	1.13	3730	12433
RastriginShift10	MBH(0.4)	99	0.99	377.7	381.5
RastriginShift10	MBH(0.5)	100	0	193.1	193.1
RastriginShift10	MBH(0.6)	100	0	213.3	213.3
RastriginShift10	MDE(10)	73	6.27	121.3	166
RastriginShift10	MDE(20)	100	0	227.4	227.4
RastriginShift50	MBH(0.3)	80	0.99	3750	4687
RastriginShift50	MBH(0.4)	100	0	1136	1136
RastriginShift50	MBH(0.5)	10	2.21	3187	31870
RastriginShift50	MDE(20)	80	2.16	792	990
RastriginShift50	MDE(50)	100	0	2385	2385

Table 2: Tests with the Rastrigin function using the local solver SNOPT

Function	Alg.(par.)	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
Rastrigin10	MBH(0.5)	100	0	417.16	417.16
Rastrigin10	MBH(0.6)	100	0	161.65	161.65
Rastrigin10	MBH(0.7)	100	0	272.9	272.9
Rastrigin10	MDE(10)	86	1.28	82.5	95.93
Rastrigin10	MDE(20)	100	0	148.6	148.6
Rastrigin50	MBH(0.5)	100	0	668.42	668.42
Rastrigin50	MBH(0.6)	0	6.96	3725.73	∞
Rastrigin50	MDE(10)	18	5.89	197.1	1095
Rastrigin50	MDE(20)	92	0.99	269.2	292.6
Rastrigin50	MDE(50)	100	0	757.5	757.5
RastriginRot10	MBH(0.4)	100	0	377.5	377.5
RastriginRot10	MBH(0.5)	100	0	178.6	178.6
RastriginRot10	MBH(0.6)	100	0	166.3	166.3
RastriginRot10	MDE(10)	76	1.65	96.5	127
RastriginRot10	MDE(20)	100	0	171.2	171.2
RastriginRot50	MBH(0.4)	97	0.99	1109.2	1143.5
RastriginRot50	MDE(10)	9	10.93	171.6	1906
RastriginRot50	MDE(20)	86	2.14	323.4	376
RastriginRot50	MDE(50)	98	48.5	1054.5	1076
RastriginShift10	MBH(0.4)	100	0	413.8	413.8
RastriginShift10	MBH(0.5)	100	0	186.7	186.7
RastriginShift10	MDE(10)	62	3.06	119.1	192
RastriginShift10	MDE(20)	100	0	211.6	211.6
RastriginShift50	MBH(0.4)	90	0.99	1260.2	1400
RastriginShift50	MBH(0.5)	20	2.36	4200.9	21004
RastriginShift50	MDE(20)	30	1.56	706	2353
RastriginShift50	MDE(50)	100	0	2015	2015

Table 3: Tests with the scaled version of the Rastrigin function using the local solver MINOS

Function	Alg.(par.)	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
Rastrigin10	MBH(0.03)	90	39.6	331.89	368.7
Rastrigin10	MBH(0.04)	83	26.7	380	457.8
Rastrigin10	MDE(10)	40	1.9	203.6	509
Rastrigin10	MDE(20)	94	1.16	411.4	437.6
Rastrigin50	MBH(0.01)	20	31.2	1428.7	7143
Rastrigin50	MBH(0.02)	60	82.7	808.8	1348
Rastrigin50	MBH(0.03)	50	231.4	1075.4	2150
Rastrigin50	MDE(20)	0	10.25	1460	∞
Rastrigin50	MDE(50)	10	3.76	8150	81500
RastriginRot10	MBH(0.3)	62	1.23	1513.3	2440.8
RastriginRot10	MDE(10)	17	1.86	230.9	1358.2
RastriginRot10	MDE(20)	61	1.04	656	1075.4
RastriginRot50	MBH(0.2)	0	16.5	7193.9	∞
RastriginRot50	MDE(50)	80	0.99	5720	7150
RastriginShift10	MBH(0.3)	92	1.24	674.5	711.4
RastriginShift10	MDE(10)	67	4.12	180.4	269.3
RastriginShift10	MDE(20)	97	0.99	356.4	367.4
RastriginShift50	MBH(0.2)	0	27.3	7350.9	∞
RastriginShift50	MBH(0.3)	0	23.6	4050.7	∞
RastriginShift50	MBH(0.4)	0	43.6	4308	∞
RastriginShift50	MDE(20)	70	10.22	1050	1500
RastriginShift50	MDE(50)	100	0	4355	4355

have also considered the following strategy, denoted by `Mix`, to update the value of Δ on the basis of the function values:

$$\Delta = \begin{cases} 5 & f(\mathbf{x}) \geq 15 \\ 2 & 10 \leq f(\mathbf{x}) < 15 \\ 1 & 5 \leq f(\mathbf{x}) < 10 \\ 0.5 & f(\mathbf{x}) < 5 \end{cases}$$

No such trick is needed for MDE: once again, it seems that this algorithm is quite good in self-adjusting the step length. As shown in Table 4, the `Mix` strategy is always better than the choice of a fixed Δ value. The results in this table also show that, while MBH and MDE perform quite similarly over the 10-dimensional instances, the performance of MDE is clearly superior for what concerns the 50-dimensional instances. It is also worthwhile to remark that a very small population size ($p = 10$ or 20) is suitable both for $n = 10$ and for $n = 50$.

3.3. Levy

We considered two versions of the Levy function:

- Levy$n$: where $\mathbf{D} = \mathbf{W} = \mathbf{I}$, $\bar{\mathbf{x}} = \mathbf{0}$;
- LevyRot$n$: where $\mathbf{D} = \mathbf{I}$, $\mathbf{W}$ randomly generated orthonormal matrix, $\bar{\mathbf{x}} = \mathbf{0}$.

We considered the dimensions $n = 10$ and $n = 50$ and always performed 100 random runs. It turns out that Levy10 and Levy50 are very simple functions for MBH, while MDE requires a bit more local searches. However, as soon as we rotate the search space the situation is reversed and MDE performs clearly better than MBH (see Table 5).

3.4. Sinusoidal

The sinusoidal function has a single funnel bottom. We only considered the basic version with $\mathbf{D} = \mathbf{W} = \mathbf{I}$ and $\bar{\mathbf{x}} = \mathbf{0}$. The function is not particularly challenging both for MBH and for MDE. For $n = 10$ both methods detect the global minimum quite quickly, while for $n = 30$ the results in Table 6 show that the two methods perform quite similarly (100 random runs for each algorithm). However, it is worthwhile to remark that when we performed tests with $n = 50$, all the members of the initial population had the same function value (3.5) and the population was unable to progress. This is probably due to the fact that local searches started from points randomly sampled within the feasible region often get stuck at stationary points with that function value. In fact, by running local searches from small perturbations of these stationary points, it is possible to escape from them, but we have not further investigated this issue.

Table 4: Tests with the Ackley function using the local solver MINOS

Function	Alg.(par.)	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
Ackley10	MBH(0.5)	100	0	155.3	155.3
Ackley10	MBH(1)	100	0	114.1	114.1
Ackley10	MBH(Mix)	100	0	55.2	55.2
Ackley10	MDE(10)	100	0	164.9	164.9
Ackley10	MDE(20)	100	0	301.8	301.8
Ackley50	MBH(0.5)	100	0	1995.2	1995.2
Ackley50	MBH(Mix)	100	0	506.4	506.4
Ackley50	MDE(10)	83	2.19	286.3	344.9
Ackley50	MDE(20)	100	0	529.8	529.8
AckleyRot10	MBH(1.5)	100	0	132.2	132.2
AckleyRot10	MBH(1)	100	0	136.9	136.9
AckleyRot10	MBH(0.5)	100	0	276.1	276.1
AckleyRot10	MBH(Mix)	100	0	69.7	69.7
AckleyRot10	MDE(10)	100	0	61	61
AckleyRot10	MDE(20)	100	0	121.8	121.8
AckleyRot50	MBH(Mix)	100	0	452.3	452.3
AckleyRot50	MDE(10)	100	0	82.8	82.8
AckleyRot50	MDE(20)	100	0	175.8	175.8
AckleyShift10	MBH(0.5)	100	0	563.8	563.8
AckleyShift10	MBH(1)	100	0	216.3	216.3
AckleyShift10	MBH(2)	100	0	185.2	185.2
AckleyShift10	MBH(Mix)	100	0	86.6	86.6
AckleyShift10	MDE(10)	97	4.57	70.9	73.1
AckleyShift10	MDE(20)	100	0	138.4	138.4
AckleyShift50	MBH(0.5)	100	0	1270.5	1270.5
AckleyShiftt50	MBH(Mix)	100	0	608.9	608.9
AckleyShift50	MDE(10)	100	0	87.2	87.2
AckleyShift50	MDE(20)	100	0	192	192

Table 5: Tests with the Levy function using the local solver MINOS

Function	Alg.(par.)	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
Levy10	MBH(0.5)	100	0	11.3	11.3
Levy10	MDE(10)	100	0	58.6	58.6
Levy50	MBH(0.5)	100	0	17.4	17.4
Levy50	MDE(10)	94	0.79	109.5	116.5
LevyRot10	MBH(0.8)	98	21.33	182.3	186
LevyRot10	MDE(10)	98	0.09	60	61.2
LevyRot50	MBH(1.2)	93	0.1	631.9	679.5
LevyRot50	MDE(10)	71	3.01	109.4	154.1
LevyRot50	MDE(20)	100	0	238	238

Table 6: Tests with the Sinusoidal function using the local solver MINOS

Function	Alg.(par.)	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
Sinusoidal30	MBH(1.0)	100	0	219	219
Sinusoidal30	MDE(10)	76	0.43	113.5	149.4
Sinusoidal30	MDE(20)	81	0.05	312	385

3.5. Schwefel

We considered two versions of the Schwefel function:

- Schwefel$n$: where $\mathbf{D} = \mathbf{W} = \mathbf{I}$, $\bar{\mathbf{x}} = \mathbf{0}$;
- SchwefelRot$n$: where $\mathbf{D} = \mathbf{I}$, $\mathbf{W}$ randomly generated orthonormal matrix, $\bar{\mathbf{x}} = \mathbf{0}$.

We ran tests with the dimensions $n = 10$ and $n = 50$. The number of random tests was always 100 for $n = 10$ and for MBH, while for MDE and $n = 50$ it was equal to: 20 for MDE(100); 10 for MDE(200) and MDE(400). The Schwefel function is the most challenging among the functions we considered and also the one for which the superiority of MDE over MBH is clearer. Its great difficulty is mostly related to the very high number of funnel bottoms (2^n). This determines the poor performance of MBH, with a single success for $n = 10$ and no success for $n = 50$ (see Table 7). Moreover, the results reported in Table 8 for the SchwefelRot50 function, show that even the best value observed in 100 runs of MBH is quite far from the global minimum value. The results are quite different for MDE. We immediately remark that, differently from the previous functions, it is convenient to increase considerably the size of the population as the dimension increases (while $p = 20, 40$ are suitable for $n = 10$, even $p = 100$ is not for $n = 50$). Next, we notice that MDE has a much higher percentage of successes with respect to MBH and even when it does not reach the global minimum value, its final value is much closer to the global minimum value with respect to MBH.

A possible interpretation of these results is the following. MDE has a natural capability of performing a descent search also in the space of the funnel bottoms. Then, for the previously considered functions (Rastrigin, Ackley, Levy, Sinusoidal) it is able to converge to the unique funnel bottom, which is also the global minimum, while for the Schwefel function it is able to reach (or, at least, to get very close) to the global minimum. A tentative explanation is that within the (large) population, MDE forms some small sub-populations, each related to some funnel bottom, and the combination of members within distinct sub-populations allows to generate new local minima belonging to the region of attraction of some better (in terms of function value) funnel bottom.

Finally we point out that we have also made some experiments introducing some crossover by taking some value $CR \in (0, 1)$. As already commented, we had initially discarded it because it could introduce too much an advantage for the separable functions. This was confirmed by the experiments: after setting

Table 7: Tests with the Schwefel function using the local solver MINOS

Function	Alg.(par.)	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
Schwefel10	MBH(100)	0	579.5	1378.5	∞
Schwefel10	MBH(125)	1	565	1280	128000
Schwefel10	MBH(150)	0	583	1288	∞
Schwefel10	MDE(20)	15	294.7	348.4	2322
Schwefel10	MDE(40)	31	195.6	842.4	2717
Schwefel50	MBH(100)	0	3094	1626.7	∞
Schwefel50	MBH(125)	0	2786	1833	∞
Schwefel50	MBH(150)	0	3445	3082	∞
Schwefel50	MDE(100)	0	1073.9	6440	∞
Schwefel50	MDE(200)	100	0	10280	10280
SchwefelRot10	MBH(100)	0	598.46	1154	∞
SchwefelRot10	MBH(125)	0	580.7	1229	∞
SchwefelRot10	MBH(150)	0	638.7	1238	∞
SchwefelRot10	MDE(20)	7	303.4	343.6	4908.6
SchwefelRot10	MDE(40)	10	228.1	903.2	9032
SchwefelRot50	MBH(100)	0	2863	1521.8	∞
SchwefelRot50	MBH(125)	0	2843	2811.9	∞
SchwefelRot50	MBH(150)	0	3162	3044.5	∞
SchwefelRot50	MDE(100)	0	1294.3	7895	∞
SchwefelRot50	MDE(200)	0	872.2	24380	∞
SchwefelRot50	MDE(400)	10	339.8	70280	702800

$CR = 0.1$, we had 13 successes out of 20 runs for MDE(100) over Schwefel50 (no success at all with $CR = 0$). However, as soon as a rotation of the space is introduced, the crossover operation seems to degrade the performance: the results we obtained with $CR = 0.1$ were clearly worse with respect to those with $CR = 0$, and those with $CR = 0.5$ were even worse with respect to those with $CR = 0.1$.

3.6. Tentative explanation

As a concluding remark about the above results, we notice that a quite basic DE approach powered by local searches leads towards quite interesting and somehow surprising results. Although a more thorough investigation of the reasons of such a good behavior may be performed, at the moment we believe that this is mostly related to the following:

- MDE has a natural capability of exploring neighborhoods at different levels, i.e., not only neighborhoods of local minima, whose exploration leads to funnel bottoms, but also neighborhoods of funnel bottoms, whose exploration leads to local minima within the space of funnel bottoms;
- MDE is able to self-adjust the size and direction of the perturbations of already detected local minima.

Table 8: Best and worst results over SchwefelRot50

Function	Alg.(par.)	# test	Worst	Best
SchwefelRot50	MBH(100)	100	-17040.7	-18935.7
SchwefelRot50	MBH(125)	100	-16920.7	-19172.6
SchwefelRot50	MBH(150)	100	-16465.2	-19052.6
SchwefelRot50	MDE(100)	20	-19172.6	-20238.5
SchwefelRot50	MDE(200)	10	-19646.3	-20475.4
SchwefelRot50	MDE(400)	10	-20238.5	-20949.1

It is also worthwhile to underline the simplicity of the approach: when implemented in AMPL, the implementation required very few lines of code. Of course, it is possible to think about more sophisticated versions, but, by intention, we did not investigate this issue: we wanted to put in evidence how a very simple machinery already delivers interesting results.

4. DE on packing problems

Finding a way to pack N identical disks in the unit square with no overlapping and maximal radius is a classical problem in computational geometry and optimization which has attracted a very large body of theoretical and computational research. The analogous problem in three dimensions, i.e., packing N identical spheres in the unit cube, although slightly less studied, is a particularly challenging global optimization test problem. Both problems have attracted a lot of computational research and many algorithms have been developed in order to discover or to confirm putative optimal conformations. The web site `www.packomania.com` is regularly maintained and updated with the most recent improvements found by the scientific community. Among the many papers and scientific publications on the subject, we cite [13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25].

The problem of packing non overlapping disks in a square (or spheres in a cube) is strictly related to the problem of scattering N points in such a way that their pairwise minimal distance is maximized – given any configuration for one of the two problems, an analogous configuration for the other can be readily found. It has been observed that methods which use local searches usually have a slight advantage in working with the formulation of the problem as a maximal dispersion one, so in this paper we will only introduce this formulation:

$$\begin{aligned} &\max d && && (1)\\ &\|x_i - x_j\| \geq d && 1 \leq i < j \leq N \\ &x_i \in [0,1]^n && \forall\, i \in 1,\ldots,N \end{aligned}$$

where $\|\cdot\|$ is the Euclidean norm, while $n = 2$ for scattering N points in the unit square and $n = 3$ for the analogous problem in the unit cube.

This problem is radically different from those reported in the previous section. First, the feasible region is defined by hard nonconvex constraints (the nonoverlapping ones) and is not a box or the rotation of a box. A further difficulty in this problem is caused by the presence of a large number of symmetries. Some symmetries are also present in standard test functions, but they are mostly related to separability and were not taken into consideration. On the contrary in packing problems, taking into account symmetries, in the way we will show later on, produced a significant improvement to the resulting algorithm.

Any configuration can be rigidly transformed into a totally equivalent one by using the transformations (rotations, reflections) which map the unit square or cube into itself. Moreover, and more significantly, a permutation symmetry exists: given any solution to the problem, adopting a labeling for the N points which is any permutation of $1, \ldots, N$ results in a geometrically indistinguishable solution which, however, corresponds to a different array in the solution space. Given these two important aspects of the problem, it should be clear that in order to apply a method like MDE to this problem, special care has to be taken.

In our experiments we adopted a series of modifications of the basic MDE scheme which are reported in the following.

Feasibility the problem of checking and possibly restoring feasibility is common in DE methods, even for problems defined on a box. The trial solution generated by the method might in fact be infeasible. In methods, like MDE, based on local optimization, the infeasibility might be neglected, by simply running the local solver from the possibly infeasible starting point. It will be the solver's duty to restore feasibility, if possible. This was indeed one of the strategies we tried. We also made experiments with scaling, i.e., we transformed every solution into a feasible one by scaling each variable through a common factor:

$$x_i^k \leftarrow \frac{x_i^k - \min_{i=1,N} x_i^k}{\max_{i=1,N} x_i^k - \min_{i=1,N} x_i^k} \qquad \forall\, k \in 1, n$$

Feasibility with respect to all constraints is thus obtained by simply putting $d = 0$ in (1).

Symmetry As the number of permutational symmetries is enormously larger than that due to rotations and reflections, we decided to tackle only the problem of reducing symmetries associated to the labeling of the objects. This is an important point in any population-based method which performs combinations of solutions. Not only reducing symmetries helps in reducing the dimension of the state space to be searched, but also, in these cases, significantly helps the algorithm in quickly selecting good solutions. In fact, the rationale behind combination rules in population methods is that by means of selection and mutation, new solutions are generated which inherit some of the good components of the current population, while allowing exploration of alternative solutions. However, combining solutions, e.g., in the way required by MDE to generate a trial, generates solutions which are linear combination of current elements in the population. What usually happens in practice is that when the population has eventually evolved towards a situation in which many elements represent the same good solution, the combination rule might generate quite random trial conformations. Indeed, considering as an example the disk packing problem, even if two solutions might be geometrically indistinguishable, the first two components of one of them might be radically different from the first two components of the other, only because they are related to different disks. The result is that the trial solution generated this way will be in general quite random, with very little chance of being accepted in the new population. This defect makes a standard implementation of the method excessively inefficient.

The solution we adopted was that of performing a quick matching with the aim of relabeling the objects in each solution in order to reduce the difficulties caused by symmetries.

The strategy we adopted in the experiments was a quite trivial greedy matching algorithm based on pairwise distances. In practice, we first chose a reference solution, which, typically, was chosen to be the best solution found so far, the so called record. Given any other solution we first computed the N^2 pairwise distances between each point in the current solution and each one in the reference one. Then, after sorting these pairwise distances, a greedy matching between points in the current solutions and points in the reference one is performed by giving each point in the current solution the label of the closest point in the reference one which was not already assigned to a previous point. Algorithm 4 reports a scheme of this procedure. In the algorithm, *pop*() is a procedure which extracts the top element from a list.

Trial generation The standard MDE trial generation rule was adopted for the array of the n–dimensional coordinates of centers. The variable d was not included in the trial generation, as it was always set to zero prior to local optimization

Initialization Each element in the first population was obtained by generating N uniform points in $[0, 1]^n$ and then starting a local optimization from the resulting configuration.

Local optimization As it is quite standard in these kind of problems, we adopted the SNOPT [12] constrained optimization package.

4.1. Numerical results

The results shown in this section are relative to the disk and sphere packing problems. For each instance 30 independent runs of the MDE algorithm were performed using an Intel(R) Core(TM) i7 870 cpu running at 2.93GHz. We compared the solutions found with the putative optima retrieved in June 2012 from `www.packomania.com` using up to the 12th decimal to deal with the arithmetic

```
Data: $\{\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_N\}$: a reference solution; $\{x_1, x_2, \ldots, x_N\}$: another
        solution
foreach $i, j \in 1, \ldots, N$ do
    let $dist_{ij} := \|\bar{x}_i - x_j\|$ ;
end
Sort $dist$ in non-decreasing order;
let $P = \{i, j\}$ be the set of indices ordered according to the order induced
by $dist$;
let $Assigned := 0$;
let $AssignedLabel[i] := \emptyset$ for all $i \in 1, N$;
let $UsedLabel[i] := false$ for all $i \in 1, N$;
while $Assigned < N$ do
    let $(i, j) :=$ pop($P$);
    remove $(i, j)$ from $P$;
    if $UsedLabel[i] = false$ and $AssignedLabel[j] = \emptyset$ then
        let $UsedLabel[i] := true$;
        let $AssignedLabel[j] := i$;
        let $Assigned := Assigned + 1$;
    end
    for $j = 1, 2, \ldots, N$ do
        let $X_{AssignedLabel[j]} := x_j$;
    end
    return $X$
end
```

Algorithm 4: Greedy matching algorithm

precision of the machine. As it was the case for the standard test functions (see Section 3) we choose MDE parameters to be $F = 0.5$ and $CR = 0$.

Each run used a population of 40 elements and 30 non-improving iterations over the population as the termination criterion – this choice was made following an analogous one in [15]. The new algorithm outperforms MBH, as presented in [15], for what concerns the success rate, and, differently from MBH, is able to find the putative optimum for every instance of the disk packing instances we tried.

In Table 9 we report the results for the disk packing problem. The meaning of the columns is the same as the corresponding ones in the previous tables. However, we remark that the globally optimal value is usually unknown for the tested instances of the disk packing problem. Thus, a run is considered as successful if the *putative* globally optimal value is reached.

Table 9: Tests with the Disk Packing Problem

Instance	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
30	100.00	-	1838.7	1838.7
31	100.00	-	2854.7	2854.7
32	100.00	-	2846.7	2846.7
33	100.00	-	2409.3	2409.3
34	100.00	-	2401.3	2401.3
35	100.00	-	2064.0	2064.0
36	93.33	0.001316398913	3238.7	3470.0
37	100.00	-	2500.0	2500.0
38	100.00	-	1894.7	1894.7
39	100.00	-	1902.7	1902.7
40	100.00	-	2713.3	2713.3
41	100.00	-	2501.3	2501.3
42	100.00	-	2162.7	2162.7
43	53.33	0.000003307755	3756.0	7042.5
44	93.33	0.000694836572	3049.3	3267.1
45	100.00	-	2769.3	2769.3
46	100.00	-	2638.7	2638.7
47	100.00	-	3422.7	3422.7
48	100.00	-	3205.3	3205.3
49	80.00	0.000068856366	3832.0	4790.0
50	100.00	-	3217.3	3217.3
51	100.00	-	3412.0	3412.0
52	100.00	-	2097.3	2097.3
53	100.00	-	3578.7	3578.7
54	93.33	0.000026913605	2642.7	2831.4
55	100.00	-	2713.3	2713.3
56	100.00	-	2665.3	2665.3
57	93.33	0.000347266942	3605.3	3862.9
58	50.00	0.000010782091	4948.0	9896.0
59	66.67	0.000352227933	3856.0	5784.0
60	70.00	0.000058609196	3894.7	5563.8
61	36.67	0.000006458690	3074.7	8385.5
62	100.00	-	3401.3	3401.3
63	100.00	-	2861.3	2861.3
64	100.00	-	3301.3	3301.3
65	100.00	-	3696.0	3696.0
66	100.00	-	3966.7	3966.7
67	100.00	-	4689.3	4689.3
68	90.00	0.000000024054	4566.7	5074.1
69	100.00	-	2417.3	2417.3
70	70.00	0.000056987029	3752.0	5360.0
71	50.00	0.000095745251	4305.3	8610.7
72	60.00	0.000150062198	4346.7	7244.4
73	53.33	0.000092202859	4297.3	8057.5
74	80.00	0.000177386383	3524.0	4405.0
75	10.00	0.000026611402	5477.3	54773.3
76	13.33	0.000064234622	4058.7	30440.0
77	80.00	0.000011100975	5078.7	6348.3
78	90.00	0.000000006704	4504.0	5004.4
79	96.67	0.000000000021	5505.3	5695.2
80	100.00	-	2862.7	2862.7
81	100.00	-	3452.0	3452.0
82	100.00	-	3578.7	3578.7
83	53.33	0.000036271189	3438.7	6447.5
84	13.33	0.000025783242	4965.3	37240.0
85	20.00	0.000006499816	3788.0	18940.0
86	16.67	0.000000889613	4230.7	25384.0
87	73.33	0.000008164509	3446.7	4700.0
88	36.67	0.000119039563	4338.7	11832.7
89	13.33	0.000051225276	4333.3	32500.0
90	50.00	0.000139946175	4010.7	8021.3
91	63.33	0.000162974660	5528.0	8728.4

Table 9: (continued) Tests with the Disk Packing Problem

Instance	% succ.	Av.diff.fail.	Av. # LS	Av. # LS per succ.
92	6.67	0.000050185602	5520.0	82800.0
93	90.00	0.000007057178	5472.0	6080.0
94	96.67	0.000013371453	3846.7	3979.3
95	86.67	0.000035247462	4450.7	5135.4
96	96.67	0.000140415969	4390.7	4542.1
97	70.00	0.000008500166	4113.3	5876.2
98	83.33	0.000000507880	4564.0	5476.8
99	100.00	-	2952.0	2952.0
100	50.00	0.000005933125	4290.7	8581.3

In Figure 1 the information on the success rate is visually summarized; in order to better appreciate the quality of the proposed algorithm, success ratios in this figures are displayed side by side with those obtained by means of MBH. In particular, in the figure we report, for each instance, the highest success ratio obtained by using one out of four variants of MBH in [15].

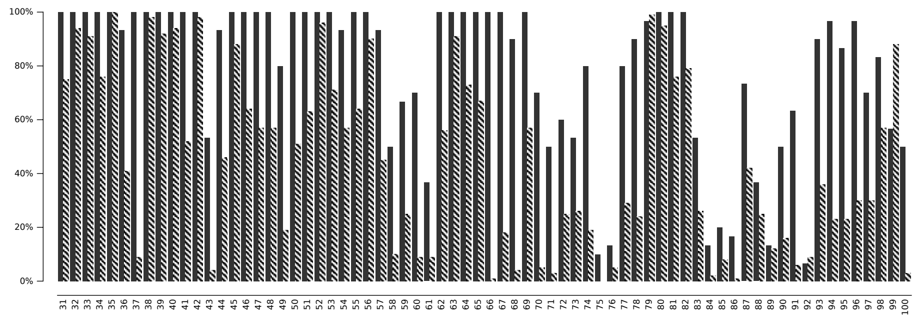

Figure 1: Success ratio for all instances of the disk packing problem; dark filled bars: success ratio for MDE, dashed bars: highest success ratio obtained by four variants of MBH as described in [15]. Although no new optima were found, the success ratio of the proposed method is clearly superior to that of MBH.

The results we obtained for disk packing were encouraging enough to motivate our trials in the much harder sphere packing problem. Here the results were quite impressive: in a relatively small number of trials we were able to improve as many as 21 putative optimal configurations in the range $N \in [30, 100]$, while re-descovering most of the other putative optima.

The results for the sphere packing problem are reported in Table 10. In this table the column % `improve` reports the percentage of runs where the previous putative optimal value has been strictly improved, while the column % `succ.` reports the percentage of runs where the final result was at least as good as the previous putative optimal value. Finally, the column `Av.diff.fail.` has been replaced by the column `Av.diff.`, where the average difference with respect to the previous putative optimal value is taken *over all the runs.* The following obvious relation exists between the two values:

$$\texttt{Av.diff.} = \texttt{Av.diff.fail.}\frac{100 - \%\text{ succ.}}{100}.$$

The negative value for the instance $N = 89$ denotes the interesting situation in which the previously best known solution is improved even on average.

Table 10: Tests with the Sphere Packing Problem (we consider a success both the matching and the improvement of an optimum)

Instance	% improve	% succ.	Av.diff.	Av. # LS	Av. # LS per succ.
30	0.00	100.00	-	1460.0	1460.0
31	0.00	100.00	-	1573.3	1573.3
32	0.00	100.00	-	1580.0	1580.0
33	0.00	100.00	-	2182.7	2182.7
34	0.00	86.67	0.000121352960	2940.0	3392.3
35	0.00	43.33	0.003963135390	2742.7	6329.2
36	0.00	0.00	0.000391685064	2738.7	∞
37	0.00	23.33	0.000024876840	2809.3	12040.0
38	0.00	90.00	0.000039471389	2426.7	2696.3
39	0.00	100.00	-	1932.0	1932.0
40	0.00	100.00	-	1837.3	1837.3
41	0.00	100.00	-	2034.7	2034.7
42	73.33	86.67	0.000012086029	3694.7	4263.1
43	0.00	73.33	0.000086824823	3157.3	4305.5
44	0.00	56.67	0.000102901117	3497.3	6171.8
45	0.00	26.67	0.000695701339	3196.0	11985.0
46	0.00	83.33	0.000071729648	2946.7	3536.0
47	0.00	6.67	0.000000050499	2284.0	34260.0
48	0.00	100.00	-	1801.3	1801.3
49	33.33	33.33	0.000070988868	3273.3	9820.0
50	0.00	50.00	0.000144332888	3453.3	6906.7
51	53.33	53.33	0.000279302726	3832.0	7185.0
52	0.00	56.67	0.000119050943	3525.3	6221.2
53	0.00	60.00	0.000480610534	3526.7	5877.8
54	0.00	6.67	0.002270004125	3750.7	56260.0
55	3.33	3.33	0.001209028348	4344.0	130320.0
56	0.00	0.00	0.000336254846	4809.3	∞
57	0.00	0.00	0.000328770774	3038.7	∞
58	0.00	0.00	0.000000822247	2585.3	∞
59	0.00	100.00	-	2654.7	2654.7
60	0.00	100.00	-	2596.0	2596.0
61	0.00	100.00	-	2402.7	2402.7
62	0.00	100.00	-	2361.3	2361.3
63	0.00	93.33	0.000801977996	2433.3	2607.1
64	60.00	93.33	0.000042652559	3922.7	4202.9
65	0.00	73.33	0.000011284981	3524.0	4805.5
66	0.00	50.00	0.000271925740	4249.3	8498.7
67	0.00	43.33	0.000669692829	3676.0	8483.1
68	23.33	40.00	0.000138734205	3938.7	9846.7
69	0.00	40.00	0.000126282512	4509.3	11273.3
70	46.67	46.67	0.000027737052	4168.0	8931.4
71	53.33	53.33	0.000020056373	4144.0	7770.0
72	0.00	76.67	0.000205817682	4325.3	5641.7
73	0.00	66.67	0.000280224238	4020.0	6030.0
74	0.00	46.67	0.000023190434	4189.3	8977.1
75	90.00	90.00	0.000000374726	4520.0	5022.2
76	0.00	90.00	0.000000000930	3381.3	3757.0
77	0.00	26.67	0.000047565713	4192.0	15720.0
78	36.67	36.67	0.000301519748	4570.7	12465.5
79	46.67	46.67	0.000381799028	3810.7	8165.7
80	0.00	0.00	0.001416162265	4594.7	∞
81	16.67	16.67	0.000804293036	4228.0	25368.0
82	0.00	10.00	0.001261980221	4198.7	41986.7
83	0.00	0.00	0.000346730138	3900.0	∞
84	0.00	76.67	0.000363324385	3720.0	4852.2
85	0.00	50.00	0.001051025719	3814.7	7629.3
86	0.00	60.00	0.000986093967	3169.3	5282.2
87	0.00	33.33	0.001707106287	3126.7	9380.0
88	0.00	100.00	-	2630.7	2630.7
89	100.00	100.00	-0.000000674792	1601.3	4804.0
90	26.67	26.67	0.000426507089	4900.0	18375.0
91	0.00	20.00	0.000215397612	4777.3	23886.7
92	33.33	33.33	0.000261778342	5788.0	17364.0
93	0.00	30.00	0.001163796008	4670.7	15568.9
94	3.33	3.33	0.000727882598	5414.7	162440.0
95	3.33	3.33	0.000273455648	5848.0	175440.0
96	13.33	13.33	0.000349887460	5386.7	40400.0
97	6.67	6.67	0.000105017113	5945.3	89180.0
98	10.00	10.00	0.000378049606	5498.7	54986.7
99	30.00	30.00	0.000616410009	5689.3	18964.4
100	0.00	70.00	0.001301426397	3538.7	5055.2

In Figure 2 the percentages of success are presented; in this figure we also report, when this is the case, the percentage of times a solution corresponding to an *improvement* over the current putative optimum was found.

Finally, in Table 11, we report the new putative globally optimal values for the 21 improved instances.

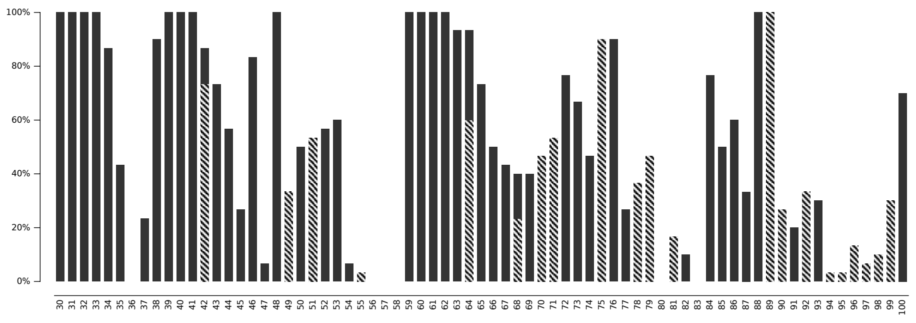

Figure 2: Success ratio for all instances of the sphere packing problem. Black filled bars represent the percentage of successes while dashed bars are used to display new putative optima discovered by the algorithm (and the associated success ratio).

Table 11: New best known values for the Sphere Packing Problem

Instance	Value	Improvement
42	0.400738455850	-0.000002577241
49	0.375238725196	-0.000000004945
51	0.369949631469	-0.000000000020
55	0.355822902422	-0.000000000026
64	0.339561588122	-0.000000353944
68	0.327794329053	-0.000000144767
70	0.326392689088	-0.000000000222
71	0.326363717355	-0.000000000246
75	0.320381089346	-0.000000010124
78	0.310774160125	-0.000002513635
79	0.308948513009	-0.000000031979
81	0.305816563817	-0.000067218542
89	0.294564596620	-0.000001008513
90	0.292932965022	-0.000000110847
92	0.290327250587	-0.000000000781
94	0.288630035326	-0.000000000829
95	0.287781438789	-0.000000000013
96	0.287496899752	-0.000000038504
97	0.287163761982	-0.000000014371
98	0.287064217790	-0.000000000782
99	0.286628569090	-0.000000001111

5. Conclusions

In this paper numerical results have been presented for a simple variant of the standard differential evolution algorithm equipped with local searches. We did not perform, on purpose, any attempt to choose the few parameters of DE in any way which could have produced better results. In particular, we maintained the fixed step size $F = 0.5$ and we did not perform any crossover by letting $CR = 0$. This latter choice is particularly critical – we feel that by introducing, at least to some extent, the possibility of crossover, the overall behavior might improve. The selection mechanism, this way, is quite a radical one in which a configuration is compared with another, totally unrelated, one which is obtained through a linear combination of three other solutions.

Despite this choice, the results we observed are quite impressive: the algorithm displays an unexpected capability both of detecting single funnels and of coordinating searches across multiple funnels; this way, in some difficult situations, MDE outperforms multiple independent repetitions of MBH, another global optimization approach based on multiple local searches.

Moreover, while in MBH several experiments might have to be performed in order to hopefully find the best neighborhood to explore (and the best neighborhood might even change across the feasible region), in MDE no parameter, except population size, has to be calibrated. It is quite evident that choosing a large enough population size is a simpler task than finding the best neighborhood (both in terms of size and of shape) to be used in MBH.

Finally, a more refined algorithm has been introduced for disk and sphere packing problems. Although the overall scheme remains the same, care was taken in order to try to match different solutions and to obtain a more efficient method to generate new solutions through the mechanism of MDE. The results, again, quite surprised us, as, with relatively low computational effort and without performing any experimentation with different parameters or different population sizes, we were able to confirm most of the known putative optima and, in the challenging case of sphere packing, we could, sometimes significantly, improve many of the currently known records.

We think that further research is needed to understand more deeply the capabilities of this method and, in particular, more research might be needed in order to be able to understand the behavior of the algorithm. Also, from the numerical point of view, a much wider set of numerical experiments should be performed in order to confirm its merits, although the successes we obtained, in particular with sphere packing, can be safely considered as a proof of the capability of this method to solve hard global optimization problems.

References

[1] R. Storn, K. Price, Differential evolution. A simple and efficient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

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

[3] M. Vasile, E. Minisci, M. Locatelli, An inflationary differential evolution algorithm for space trajectory optimization, IEEE Transactions on Evolutionary Computation 15 (2) (2011) 267 – 281.

[4] J. Lampinen, I. Zelinka, On stagnation of the differential evolution algorithm, in: P. Ošmera (Ed.), Proceedings of Mendel 2000, 6th international conference on soft computing. Brno, Czech Republic, 2000, pp. 76–83.

[5] D. Zaharie, Critical values fo the control parameters of differential evolution, in: R. Matoušek, P. Ošmera (Eds.), Proceedings of Mendel 2002, 8th International Conference on Soft Computing, 2002, pp. 62–67.

[6] M. Locateli, F. Schoen, Global Optimization, SIAM, forthcoming, 2013.

[7] D. Molina, M. Lozano, A. Sànchez, F. Herrera, Memetic algorithms based on local search chains for large scale continuous optimisation problems: MA-SSW-Chains, Soft Computing 15 (2011) 2201–2220.

[8] R. H. Leary, Global optimization on funneling landscapes, Journal of Global Optimization 18 (2000) 367–383.

[9] D. J. Wales, J. P. K. Doye, Global optimization by Basin-Hopping and the lowest energy structures of Lennard–Jones clusters containing up to 110 atoms, Journal of Physical Chemistry A 101 (28) (1997) 5111–5116.

[10] M. Locatelli, On the multilevel structure of global optimization problems, Computational Optimization and Applications 30 (1) (2005) 5–22.

[11] B. Murtagh, M. Saunders, MINOS 5.5 User's guide, Tech. Rep. SOL 83-20R, Dept of Operations Research, Stanford University (1998).

[12] P. E. Gill, W. Murray, M. A. Saunders, SNOPT: An SQP algorithm for large-scale constrained optimization, SIAM Review 47 (1) (2005) 99–131.

[13] P. G. Szabó, M. C. Markót, T. Csendes, E. Specht, L. G. Casado, I. Garcia, New Approaches to Circle Packing in a Square With Program Codes, Springer, 2007.

[14] B. Addis, M. Locatelli, F. Schoen, Efficiently packing unequal disks in a circle, Operations Research Letters 36 (1) (2008) 37–42.

[15] B. Addis, M. Locatelli, F. Schoen, Disk packing in a square: A new global optimization approach, INFORMS J. on Computing 20 (4) (2008) 516–524.

[16] D. V. Boll, J. Donovan, R. L. Graham, B. D. Lubachevsky, Improving dense packings of equal disks in a square, The Electronic Journal of Combinatorics 7 (R46) (2000) 1–9.

[17] L. G. Casado, I. Garcia, P. Szabó, T. Csendes, Equal circles packing in square II: New results for up to 100 circles using the TAMSASS-PECS algorithm, in: F. Giannessi, P. M. Pardalos, T. Rapcsak (Eds.), Optimization Theory: Recent developements from Mátraháza, Kluwer Academic Publishers, 1998, pp. 207–224.

[18] I. Castillo, F. J. Kampas, J. D. Pinter, Solving circle packing problems by global optimization: Numerical results and industrial applications, European Journal of Operational Research 191 (2008) 786–802.

[19] A. Donev, S. Torquato, F. H. Stillinger, R. Connelly, A linear programming algorithm to test for jamming in hard–sphere packings, Journal of Computational Physic 197 (2004) 139–166.

[20] A. Grosso, A. Jamali, M. Locatelli, F. Schoen, Solving the problem of packing equal and unequal circles in a circular container, Journal of Global Optimization 47 (1) (2010) 63–81.

[21] W.Q. Huang and Y. Li and H. Akeb and C.M. Li, Greedy algorithms for packing unequal circles into a rectangular container, Journal of the Operational Research Society 56 (2005) 539–548.

[22] M. Locatelli, U. Raber, Packing equal circles into a square: A deterministic global optimization approach, Discrete Applied Mathematics 122 (2002) 139–166.

[23] C. O. Lôpez, J. E. Beasley, A heuristic for the circle packing problem with a variety of containers, European Journal of Operational Research 214 (3) (2011) 512 – 525.

[24] M. C. Markôt, T. Csendes, A new verified optimization technique for the "packing circles in a unit square" problem, SIAM J. on Optimization 16 (2005) 193–219.

[25] P. G. Szabó, M. C. Markôt, T. Csendes, Global optimization in geometry - Circle packing into the square, in: C. Audet, P. Hansen, G. Savard (Eds.), Essays and Surveys in Global Optimization, Kluwer Academic Publishers, 2005, pp. 233–266.