The grouping differential evolution algorithm for multi-dimensional optimization problems*

by

Adam P. Piotrowski and Jarosław J. Napiórkowski

Institute of Geophysics, Polish Academy of Sciences,
Księcia Janusza 64, 01-452, Warsaw, Poland

Abstract: A variant of the Differential Evolution method is presented. The classical Differential Evolution approach is very successful for simple problems, but does not perform well enough for troublesome multi-dimensional non-convex continuous functions. To overcome some of the drawbacks, the Grouped Multi-Strategy Differential Evolution algorithm is proposed here. The main idea behind the new approach is to exploit the knowledge about the local minima already found in different parts of the search space in order to facilitate further search for the global one. In the proposed method, the population is split into four groups: three of them rarely communicate with the others, but one is allowed to gain all available knowledge from the whole population throughout the search process. The individuals simultaneously use three different crossover/mutation strategies, which makes the algorithm more flexible. The proposed approach was compared with two Differential Evolution based algorithms on a set of 10- to 100-dimensional test functions of varying difficulty. The proposed method achieved very encouraging results; its advantage was especially significant when more difficult 50- and 100-dimensional problems were considered. When dividing population into separate groups, the total number of individuals becomes a crucial restriction. Hence, the impact of the number of individuals on the performance of the algorithms was studied. It was shown that increasing the number of individuals above the number initially proposed for classic Differential Evolution method is in most cases not advantageous and sometimes may even result in deterioration of results.

Keywords: differential evolution, multidimensional problems, multimodal problems, metaheuristics.

1.   Introduction

The global optimization problem in this paper may be described in the following form – for a continuous real valued function $f(\boldsymbol{x})$, find the global optimum vector $\boldsymbol{x}^*$, such that:

$$f(\boldsymbol{x}^*) = \min f(\boldsymbol{x}) \tag{1}$$

where $\boldsymbol{x}$ is an M–dimensional continuous variable vector with domain $\Omega \subset R^M$.

Among a number of global optimization methods proposed during last twenty years, the Differential Evolution (DE) algorithm described in detail in Storn and Price (1995; 1997) is said to be one of the most promising (Clerc, 2006), although probably not the most popular one. DE is a purely heuristic population-based algorithm that has been successfully applied to a wide range of test functions and real-world problems (Ilonen, Kamarainen and Lampinen, 2003; Ali, Khompatraporn and Zabinsky, 2005; Price, Storn and Lampinen, 2005; Rowiński and Piotrowski, 2008). The successful idea of DE consists in evolving iteratively a population of individuals in the search space to the vicinity of the global optimal solution $\boldsymbol{x}^*$, according to a predefined heuristic that is based on adding the weighted differences of two population vectors to a third vector. Although the basic DE is successful in many simple low-dimensional cases, rather large computational time is usually required for multimodal problems with higher dimensionality, and the average success rate significantly decreases for such cases. Hence, increasing the success rate in finding the global optimum $\boldsymbol{x}^*$ and reducing computational time were obvious motivations for the recently proposed improvements (e.g. Liu and Lampinen, 2005; Mishra, 2006; Wang and Zhang, 2007; Ali, 2007; Brest et al., 2007, Das et al., 2009, Qin, Huang and Suganthan, 2009).

In this paper, we propose a new modification of the DE algorithm, namely Grouped Multi-Strategy Differential Evolution (GMS), and compare it with the basic DE and Multi-Strategy Differential Evolution algorithm (MS) suggested by Mishra (2006) for a set of selected test functions.

In the present paper we understand *strategy* as a rule referring to crossover and mutation only, whereas *algorithm* is understood as the whole heuristic, defining the rules of optimization process. In the DE algorithm, only one strategy is used, while in the MS algorithm, three different strategies related to crossover and mutation are applied with predefined probability. In the case of the proposed GMS algorithm, the population is divided into four different groups, three of them in principle work mutually independently using the same strategies as in the MS. The methods of inter-group exchange of information, that we propose, significantly improve the performance of the GMS approach in comparison with both basic DE and MS algorithms.

For DE, MS and GMS algorithms, the effect of both dimensionality (M equal to 10, 30, 50 and 100) and the population size (varying from $10 \cdot M$ to $100 \cdot M$) on the success rate of finding of $\boldsymbol{x}^*$ and the calculation time is studied in details.

The optimization techniques are tested for a set of very popular and relatively simple functions, as well as for some rarely used and much more troublesome ones.

2. Differential evolution algorithm (DE) – strategy 1

The DE algorithm proposed by Storn and Price (1995) is a stochastic population-based method. To find the optimal solution $\boldsymbol{x^*}$ in the M-dimensional space, the DE initializes a population P with K individuals (vectors). The assumed parameter bounds define the domain, from which individuals are chosen randomly by uniform distribution over search space. The algorithm maintains a population with K individuals in each generation. Storn and Price (1995) suggested $K = 10M$ as a default value.

   In each iteration, for every individual $\boldsymbol{x}_i$ $(i = 1, \ldots, K)$, three other distinct vectors $\boldsymbol{x}_a$, $\boldsymbol{x}_b$, and $\boldsymbol{x}_c$ are randomly chosen from the population P. Then, a new vector $\boldsymbol{u}_i$ is generated by adding the weighted difference between two random vectors to a third vector – this operation is called mutation. The mutated vector $\boldsymbol{u}_i$ and the target (initial) vector $\boldsymbol{x}_i$ form two parents. In order to form the trial vector (offspring) $\boldsymbol{y}_i$, the crossover between $\boldsymbol{u}_i$ and $\boldsymbol{x}_i$ is performed. The pseudocode of the described mutation and crossover used in the DE algorithm, denoted as Strategy 1, is given in Fig. 1.

```
...
do  a= ceil(rand(0,l)*K); while (a==i);
do  b= ceil(rand(0,l)*K); while (b==a || b==i);
do  c= ceil(rand(0,1)*K); while (c==a  || c==b || c==i);
jrand= ceil(M*rand(0,l)) ;
uᵢ =  xₐ + F*(x_b  - x_c) // mutation
for (j=1; j = M; j++)  // generate a trial vector yʲ by crossover
      {if  (rand(0,1) ≤  Cr or j = jrand)  yᵢʲ = uᵢʲ;
       else  yᵢʲ = xᵢʲ }
```

Figure 1. Strategy 1

   If $f\left(\boldsymbol{y}_i\right) \leq f\left(\boldsymbol{x}_i\right)$, then $\boldsymbol{y}_i$ replaces $\boldsymbol{x}_i$ – this operation is called selection. In Strategy 1, $rand(0,1)$ is a random number from a uniform distribution over the interval $(0,1)$; $ceil(z)$ returns the smallest integer value that is not less than z; F is the scale factor (suggested by Storn and Price, 1997, to be 0.5, and by Kaelo and Ali, 2006, to vary between 0.4 and 1); $jrand$ is a random integer from $[1,M]$. The value of the algorithm parameter Cr depends on the problem (Price, Storn and Lampinen, 2005), but Ali (2007) suggests that the value of $Cr = 0.5$ is usually a reasonable choice. The above evolution process is repeated until termination conditions are met.

The entire algorithm terminates when one of the stopping criteria defined by the user is met. The stopping criterion used in this paper is based on the improvement of the objective function during the predefined number of iterations PNI:

$$FBEST_{PNI} - FBEST < \varepsilon \qquad\qquad (2)$$

where $FBEST$ denotes the best objective function value at present iteration, $FBEST_{PNI}$ is the best objective function value obtained PNI iterations earlier, and ε defines the threshold value of the required minimum improvement.

3. Multi–strategy differential evolution algorithm (MS)

Since 1995 different methods of crossover and/or mutation have been proposed for Differential Evolution-based algorithms. A detailed discussion of superior variations of the basic methodology may be found in Price, Storn and Lampinen (2005). The new strategies differ in their ways of creating offspring and may be successfully used for minimizing different types of continuous functions.

An interesting suggestion leading to an improvement of the basic DE can be found in Mishra (2006). In the proposed Multi-Strategy Differential Evolution algorithm, for any vector $\boldsymbol{x}_i$ from the population P, one out of three selected strategies is chosen according to predefined probability. The following strategies are recommended: Strategy 1 used in basic DE, the Exponential Crossover and the modified Either-Or method (Mishra, 2006). MS technique significantly improves the success rate of finding the global optimal solution.

Other strategies discussed in Price, Storn and Lampinen (2005) are not included in MS, because they use a perturbation of the best individual (strategy called Best) or the best individual in determination of the direction of perturbation (called Target-To-Best). That may result in premature convergence to local optima, especially in multi-dimensional problems.

3.1. Exponential crossover – strategy 2

Mutation in the Exponential Crossover strategy is performed in the same way as in Strategy 1, by determining the vector $\boldsymbol{u}_i$. To create an offspring, the crossover is modified in the following way. An index value $jrand$ is randomly selected from $[1, M]$, and the corresponding parameter u_i^j is copied from mutant $\boldsymbol{u}_i$ to the parent $\boldsymbol{x}_i$, so that the trial vector (offspring) $\boldsymbol{y}_i$ will be different from the parent vector $\boldsymbol{x}_i$. Then, the source of subsequent elements of the trial vector are determined by comparing Cr to a uniformly distributed random number $rand(0, l)$ that is generated anew for each index j. As long as $rand(0, l) \leq Cr$, parameters continue to be taken from the mutant vector $\boldsymbol{u}_i$, but since for the first time $rand(0, l) > Cr$, the current and all remaining parameters are taken from the target vector $\boldsymbol{x}_i$. The method with $Cr = 0.5$ tends to retain most of the elements from the parent vector $\boldsymbol{x}_i$. Hence, when used alone, exponential

crossover rarely leads to successful finding of global optimum. However, when applied together with other strategies, it turns out to be an important part of the algorithm. It facilitates optimization of different functions, not only, but first of all, the separable ones, and can improve performance of the algorithm near the optimum for many problems.

The pseudocode of the Exponential Crossover Strategy is provided in Fig. 2.

```
...
do  a= ceil(rand(0,l)*K); while (a==i);
do  b= ceil(rand(0,l)*K); while (b==a || b==i);
do  c= ceil(rand(0,1)*K); while (c==a  || c==b ||  c==i);
y_i = x_i  // child inherits parent parameters
u_i =  x_a + F*(x_b  - x_c); // mutation
jrand= ceil(M*rand(0,l));
j = jrand
do { y_i^j = u_i^j ;  // child inherits a mutant parameter
     j = (j + 1) % M // increment j, modulo M
}
while (rand(0,1) ≤ Cr && j!= jrand)
```

Figure 2. Strategy 2

3.2. Modified either-or strategy

Contrary to the Exponential Crossover, the Modified Either-Or strategy does not prefer copying elements of the parent vector $\boldsymbol{x}_i$ to the offspring $\boldsymbol{y}_i$. To create the offspring $\boldsymbol{y}_i$, for any target vector, two additional vectors are created: the first one by mutation, $\boldsymbol{u}_i$, and the second one by recombination, $\boldsymbol{w}_i$, with the use of a random number from the normal distribution. Then, for each j-th component, a random number $rand(0,l)$ is generated and compared to Cr. If $rand(0,l) \leq Cr$, the offspring parameter is copied from the mutant vector $\boldsymbol{u}_i$; otherwise, the parameter is copied from the recombined vector $\boldsymbol{w}_i$. The pseudocode of the Modified Either-Or Strategy is provided in Fig. 3. Note that $\boldsymbol{x}_a$ is not used to create $\boldsymbol{w}_i$. In Fig. 3 $normal\_distribution(0,1)$ is a random value generated from the standardized normal distribution with the mean equal to 0 and the variance equal to 1. This method ensures that each component of the offspring $\boldsymbol{y}_i$ almost surely differs from the corresponding element of the target vector $\boldsymbol{x}_i$.

The details of the MS algorithm are presented in Fig. 4.

```
...
do  a= ceil(rand(0,l)*K);  while (a==i);
do  b= ceil(rand(0,l)*K); while (b==a || b==i);
do  c= ceil(rand(0,1)*K); while (c==a  ||  c==b || c ==i);
// mutation
uᵢ  =  xₐ  + F*(x_b   - x_c );
// recombination
wᵢ  =  xᵢ  + (x_b   + x_c  - 2*xᵢ )*normal_distribution(0,1);
// generate a trial vector yʲ by crossover
for (j=1; j = M; j++)
       {if  (rand(0,1) ≤  Cr) yᵢʲ = uᵢʲ;
        else                  yᵢʲ = wᵢʲ;}
```

Figure 3. Strategy 3

```
Step 0. Preset mutation parameter F, crossover parameter Cr,
        the number of individuals in population set K,
        parameters of stopping criterion PNI and ε .
        Randomly generate population P of K individuals xᵢ
        from the solution space Ω ;
        Iter = 0  // iteration number;

Step 1. while (the stopping criterion is not satisfied) {
            Iter = Iter +1  // add 1 to iteration number
            for (i=1; i = K; i++)  {
            // select one out of three  strategies
            S= ceil(3*rand(0,l));
            Generate offspring  yᵢ  by means of  Strategy S
            if  (f(yᵢ )  ≤  f(xᵢ )) then  xᵢ  = yᵢ  // selection
            } // poorer vectors in Ω are replaced by
               // better vectors
        } // end while
```

Figure 4. The MS Algorithm.

4. Grouped multi-strategy differential evolution (GMS)

The main goal of the Grouped Multi-Strategy Differential Evolution (GMS)
proposed in the present paper is to make the optimization algorithm less vul-
nerable to being trapped in a local minimum. The basic idea is to exploit the
information on location of local minima in the search space in order to find a
still better minimum, hopefully the global one – x^*. In order to find several
different local optima, the population P is divided into four separate groups

(sets) $G(k)$, $k \in \{1, 2, 3, 4\}$, i.e.

$$P = \bigcup_{k=1}^{4} G(k) \tag{3}$$

For all $k, l \leq 4, k \neq l \ \ G(k) \cap G(l) = \oslash$.

All groups have approximately the same number of elements, large enough to allow for a reasonable search by means of the DE technique, but four times smaller than P. Each individual in P belongs to the one group only.

If the problem belongs to the class of the so-called "nasty" ones, four groups are expected to be trapped in different local optima.

The GMS algorithm attempts to replace each individual in each group by means of the three strategies described above, like in MS. The first three groups, $G(1)$, $G(2)$ and $G(3)$, work separately during most of iterations, while the fourth group, $G(4)$, has access to all the available information stored by individuals in population P. If the predefined conditions are met, one of the first three groups may be allowed to gain information from other groups. Additionally, in particular situations the best individual in the group may be "frozen" in the search-space. The stopping criteria are adopted in order to avoid premature cessation of the algorithm prior to exchanging all gained information among the groups.

The strategy proposed in the paper has some similarities with the so-called Island Models or species based approaches (see, e.g., Holland, 2000; Liu, Yao and Higruchi, 2000; Gustafson and Burke, 2006). However, contrary to most island models, we do not allow individuals to migrate, and do not select privileged species to be located in separate group.

4.1. Exchanging information between groups

In the GMS algorithm, a logical variable $LG(k)$ is assigned to each group. If $LG(k) = 0$, then for each parent $\boldsymbol{x}_i \in G(k)$ the vectors $\boldsymbol{x}_a$, $\boldsymbol{x}_b$ and $\boldsymbol{x}_c$ are randomly chosen from the group $G(k)$ only. Hence, the groups with $LG(k) = 0$ are looking for the optimal solution independently, without exchanging the information with other groups. On the other hand, if $LG(k))$ is equal to 1, the vectors $\boldsymbol{x}_a$, $\boldsymbol{x}_b$ and $\boldsymbol{x}_c$ for each parent $\boldsymbol{x}_i \in G(k)$ may be chosen randomly from the whole population P. Hence, an individual from the group with the assigned variable $LG(k) = 1$ is able to gain information from any individuals in population P and has a chance to use the information about location of local minima found by all groups. By definition $LG(4) = 1$.

Initially $LG(1)$, $LG(2)$ and $LG(3)$ are set to 0. Then, after every *PNI* iterations, the factor of improvement (GF) is computed:

$$GF = \left(\sum_{k=1}^{4} \left(GFBEST(k)_{PNI} - GFBEST(k) \right) \right) / 100 \tag{4}$$

where $GFBEST\,(k)$ and $GFBEST\,(k)_{PNI}$ are the best values of the objective function for the group $G(\mathrm{k})$ at current iteration and PNI iterations earlier, respectively. If the objective function for the best individual in a group ($GFBEST\,(k)$, $k = 1, 2, 3$) fulfils the relation:

$$GFBEST\,(k)_{PNI} - GFBEST\,(k) < GF \qquad\qquad (5)$$

it may be assumed that group $G\,(k)$ is trapped in a local minimum. To allow it to escape, the value of $LG\,(k)$ for this group $G\,(k)$ is set to 1 for the next PNI iterations. After PNI iterations, the $LG\,(k)$ is set back to 0 and the individuals from the group $G\,(k)$ again create the mutated vectors without communication with the other groups. Only one logical value $LG\,(k)$ that was assigned to first three groups may change its value to 1 at the same time.

This method allows the individuals from the group that does not improve its performance to continue the search efficiently after some time of being trapped in local minimum. Please note that in the meantime, the individuals from $G\,(4)$, i.e. the group with $LG\,(k) = 1$, may easily benefit from knowledge about the positions of such local minima.

4.2. "Freezing" of individuals in the search space

For some complex multidimensional problems the number of local minima found during the search may be substantial. The information about their location in the search space may be used during further search. Therefore, another logical variable $Lx\,(i)$ is assigned to each target vector $\boldsymbol{x}_i$ in population. The individuals with $Lx\,(i) = 0$ create offspring $\boldsymbol{y}_i$ and can be replaced in population P by them. The individuals with $Lx\,(i) = 1$ are "frozen" – i.e. they cannot create offspring, but may be used in the operation of mutation as one of the vectors $\boldsymbol{x}_a$, $\boldsymbol{x}_b$ or $\boldsymbol{x}_c$ by the parent vector from the same group, or from the group with the assigned value $LG\,(k) = 1$. Initially, for any $\boldsymbol{x}_i$ in the population, $Lx(i) = 0$. When the logical variable $LG\,(k)$ assigned to one of the first three groups is changed to 1, then the $Lx\,(i)$ for the best individual $GB(\mathrm{k})$ in the group $G\,(k)$ – the one with the lowest $GFBEST\,(k)$ – is set to 1. "Freezing" of individuals results in decreasing the number of active vectors that can create an offspring. Therefore, it is assumed that no more than a predefined number of individuals NFI may be "frozen", e.g. NFI is approximately set to $K/10$. If the next individual is to be "frozen" once the number of "frozen" individuals equals $K/10$, then the individual *already* "frozen" for the highest number of iterations (for example the one with index if) is "released", and its corresponding variable $Lx\,(if)$ is set to 0.

4.3. Termination criteria

Usually, the minimum of objective function is not known in advance. In this case, optimization can be terminated after $Imax$ generations. Finding a value

Imax that is large enough to secure enough time to find the optimum, but not too high, involves additional tests. Optimization can also be terminated when the difference between the population's worst and best values of the objective function falls below a predetermined limit.

The stopping criterion used in the present paper is based on the improvement of the objective function during a predefined number of iterations *PNI* (Inequality 2). For the GMS algorithm, the logical value called *StopFlag* is introduced and initially set to 0. It defines whether *PNI* iterations took place before inequality (2) was fulfilled or not.

Let *GFBEST$_W$* be the worst and *FBEST* the best objective function values among the best individuals of each group:

$$FBEST = \min_k GFBEST\,(k)$$
$$GFBEST_W = \max_k GFBEST\,(k)\,. \tag{6}$$

If inequality (2) is fulfilled and the difference between the worst *GFBEST$_W$* and best *GFBEST* objective function values is greater than the predetermined limit

$$GFBEST_W - FBEST > \varepsilon \tag{7}$$

then one may suppose that the groups are still scattered in the search space and the global optimum is probably not yet found. So the *StopFlag* is changed to 1 and the search is continued for the next *PNI* iterations, but with all logical variables *LG(k)* set to 1. If, after the next *PNI* iterations inequality (2) is fulfilled, it means that the algorithm was unable to improve the objective function using the information stored in all groups and it is terminated. On the other hand, if after *PNI* iterations inequality (2) is not fulfilled, it means that the algorithm escaped from the local minima and continues the search with *LG(k)* for the first three groups as well as *StopFlag* set back to 0. Note that in the basic DE and in most of other population-based algorithms, fulfillment of condition (2) would terminate optimization.

The GMS algorithm is presented in Figs. 4 and 5 .

Some discussion on the number of groups may be expected. To allow for a reasonable search, the number of individuals in each group cannot be too small, and so the number of groups must be very limited – we suggest dividing populations into four groups for greatest efficiency. Storn and Price (1995) suggested the use of $10M$ individuals in the DE algorithm. But an increase in the number of individuals, which is studied further in this paper, would further slow the search process and increase the number of function evaluations. In the preliminary study we have found that while for many simple testing problems the reduction of the number of individuals in P to $3M$ may only slightly deteriorate the results when the MS approach is applied, a further decrease in the number of individuals has significant negative impact on the results. When the number

```
Step 0. Preset mutation parameter F, crossover parameter Cr,
        the number of individuals in population set K,
        parameters of stopping criterion PNI and initial
        value of FBEST_PNI (very high);
        Randomly generate population P of K individuals x_i
        from the solution space Ω ;
        Divide population P into groups G(1), G(2), G(3), G(4);
        LG(4) = 1 and LG(k) = 0 for k = 1, 2, 3;
        Lx(i) = 0 for any x_i  ∈ P
        Iter = 0          // iteration number;
        StopFlag = 0
        Determine the value of the objective
        function f(x_i ) for each x_i  ∈ P

Step 1. if (the stopping criterion is satisfied) Stop algorithm
        Iter = Iter +1           // add 1 to iteration number
        Find    BEST; FBEST;     // the best vector in P with
                                    corresponding function value
        GB(k), GFBEST(k)         // the best vector in each
                                    group k with corresponding
                                    function value
        for each vector x_i (i=1, 2,..., K)
          Determine the group number k for individual x_i
          if (Lx(i) == 0) {       //if the individual is not frozen
                S= ceil(3*rand(0,l));   // Select one out of
                                            three strategies
                if (LG(k) == 0)
                  Generate offspring  y_i  based on
                  x_a, x_b, x_c  ∈ G(k) using Strategy S;
                else
                  Generate offspring  y_i based on
                  x_a, x_b, x_c ∈ P using Strategy S;
                end if
                if  (f(y_i) ≤  f(x_i)       x_i = y_i    // selection
          end if
        end for
```

Figure 5. The GMS Algorithm – steps 0 and 1

```
Step 2. After every PNI iterations, i.e.  when Iter (mod) PNI = 0

        LG(k)  = 0 for k=1, 2, 3
        Compute the group factor of improvement (GF) (Eq.4)
        for each group G(k),  k = 1, 2, 3
          if (condition described by Eq.(5) is fulfilled)
            LG(k) = 1  // will be valid for next PNI iterations
            Lx(i) = 1  // freezing one xᵢ - the GB(k)
                             in group G(k)
            if (number of frozen individuals
                exceeds the predetermined value)
              release the individual x_if with Lx(if)=1 frozen
              for the highest number of iterations
              by setting Lx(if) = 0
              GOTO Step 3 // only one LG(k),
                            (k=1,2,3) may change to 1
            end if
          end if
        end for

Step 3. After every PNI iterations, i.e.  when Iter (mod) PNI = 0,
        check the stopping criteria:
        if (FBEST_PNI - FBEST) > ε
          StopFlag = 0
          FBEST_PNI = FBEST
          Turn to Step 1
        else
          if (StopFlag = =1)
             Stop algorithm
          else
            StopFlag = 1
            FBEST_PNI = FBEST
            Find GFBEST_W = max GFBEST(k) // the worst from the best
                                              objective function
                                              values of all groups
            if (GFBEST_W - FBEST) >   ε
              LG(k) = 1 for k=1,2,3 // valid for next PNI iteration
              Turn to Step 1
            else
              Stop algorithm
            end if
          end if
        end if
        Turn to Step 1
```

Figure 6. The GMS Algorithm – steps 2 and 3

of individuals in P is $10M$, the number of individuals in one group would be $2.5M$, i.e. on the lower level of acceptance for the relatively simple problems. We suggest that at least three independently working groups are needed. As a result, we partitioned the population into four groups, three of which search independently. The idea of dividing the population into non-uniform groups or using more groups for $P > 10M$ was not verified and may be the subject of future research.

5. Numerical results

In the present section, the performance of the proposed GMS method is compared with the one of the MS and DE algorithms for a set of multi-dimensional test functions. This set includes traditional numerical benchmark functions that can be defined for any dimensionality M (see Appendix). The last six test problems may be considered to be more difficult and therefore particularly interesting. For example, problems in successful optimizing of even two-dimensional Normalized Rana function were reported in Tao and Wang (2007). Unfortunately, they are rarely used for comparison of the optimization algorithms (Whitley et al., 2004).

In the present paper we adopted the following parameter values: $PNI = 500$, $F = 0.5$, $CR = 0.5$, $\varepsilon = 10^{-4}$. The maximum number of function calls was set to a very high number (at least $3 \cdot 10^7$), as the algorithms were designed to stop according to the termination criteria described in Section 4.3, that were almost always fulfilled much quicker. Each strategy (1, 2 or 3) may be selected with the same probability ($1/3$).

Because the partition of the population into four groups decreases the number of individuals that may communicate with each other during optimization, the impact of the number of individuals K in population P was studied in detail.

The 10-, 30-, 50- and 100-dimensional versions of each function were considered. The number of individuals used by each algorithm was 100, 300, 500 and 1000 for 10-dimensional, then 300, 500 and 1000 for 30-dimensional, 500 and 1000 for 50-dimensional, and 1000 for 100-dimensional problems. The individuals in the parameter space were randomly initiated from the appropriate range, depending on the test function (see Appendix).

Optimization of each function by three algorithms (DE, MS and GMS) was repeated 100 times. Tables 1-3 contain the number of unsuccessful runs (integer values) or, when global optimum was never found, both the 100-run average and the best solutions found during 100 runs (*real values in italic*). Table 4 presents the average number of function calls for 10- and 100-dimensional problems (*100-dimensional in italic*). The number of function calls is averaged over 100 trials, both successful and unsuccessful.

In the case of 10-dimensional problems, the performance of the tested algorithms differs only for the most troublesome problems, namely Salomon (SA), Whitley (WH), Rana (RN), Eggholder (EG) and Dixon-Price (DP) functions.

All other functions are solved almost perfectly; the basic DE had difficulty finding the single optimum with the suggested number of individuals in population, $10M$, for the Rosenbrock (RO) problem.

In the case of the Salomon (SA) function, all algorithms failed, as expected. As a result, additional tests for lower-dimensional SA were performed in order to find the highest integer value of M, for which the global optimum can be found by the particular algorithm (when $K = 10M$). It was verified that during 100 runs the global minimum of Salomon function was found at least once by GMS – for 5-dimensional, by MS – for 4-dimensional and by DE – only for 3-dimensional cases.

Also, none of algorithms was able to determine the global optimum for the Rana function. Similarly to Salomon's problem, the tests for various lower-dimensional versions were performed. The results obtained are the following: the best GMS algorithm was able to the find global optimum of the Rana function for 6-dimensional, DE for 5-dimensional and MS only for 4-dimensional versions.

The basic DE clearly performs the worst for the 10-dimensional versions of WH, RN, EG and DP functions. It was the only method unable to find the global optimum for WH and DP functions. It is interesting that DE is also the only algorithm whose performance decreases with the increase of K for the EG function.

The tests with different K showed that, in general, for 10-dimensional problems, increase of K only increases the computational time, without improving the performance of the particular algorithm (see Tables 1 and 4).

The difference between the performance of the three algorithms under consideration is more evident for 30- and 50-dimensional versions (see Tables 2 and 3). The DE was outperformed both by MS and GMS more frequently than in the 10-dimensional case (see Rastrigin, RS, Neumaier 3, NU, as well as SA and EG problems). The difference between MS and GMS also becomes significant, especially for RO, WH, RN and EG functions. For the 50-dimensional SA problem, the GMS is able to find the "sphere" closer to the global optimum than MS. For K equal $10M$, some problems with determining the optimum of the Rosenbrock function by all the methods occur, but GMS performs much better than the other algorithms. For 30- and 50- dimensional problems, the increase of K gives no clear improvement for any algorithm. Although GMS with 1000 individuals outperforms the version with 500 individuals for 50-dimensional Rana function, the use of too many individuals leads to the decrease in performance for EG and WH problems.

For 100-dimensional problems GMS performs much better than the other algorithms. The difference becomes noticeable even for some simple test functions like Griewank (GR) and Levy-Montalvo 2 (LM2). For more intricate ones, like SA, WH, RN, EG and RO, the difference in the performance of the discussed algorithms becomes very significant.

Table 1. Results obtained for 10-dimensional functions after 100 runs for each algorithm. The integer values denote the number of cases not matching the global optimum in the case when algorithm was successful at least once. The *real italic values* denote the best (upper) and the average (lower) solution found during 100 runs when algorithm was not successful. In the first row the number of individuals is shown. For the test functions not included in the table algorithms were always successful.

Fct	$f(x_i^*)$	GMS	MS	DE	GMS	MS	DE	GMS	MS	DE	GMS	MS	DE
		100	100	100	300	300	300	500	500	500	1000	1000	1000
RO	0	0	0	50	0	0	0	0	0	0	0	0	0
SA	0	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*	*0.1* / *0.1*
WH	0	3	12	100	1	1	100	0	0	100	0	0	100
RN	-512.7	*-496.8* / *-479.8*	*-496.9* / *-465.7*	*-439.6* / *-401.7*	*-502.4* / *-484.1*	*-498.4* / *-470.7*	*-444.4* / *-409.2*	*-499.4* / *-494*	*-495.8* / *-471.8*	*-454.1* / *-411.8*	*-502.7* / *-488.2*	*-498.3* / *-476.1*	*-452.1* / *-419.4*
EG	(-8291.2)	33	45	76	7	42	85	7	33	93	10	39	96
DP	0	72	97	100	88	95	100	86	89	100	78	82	100

Table 2. Best results obtained for 30-dimensional functions in 100 runs for each algorithm. The integer values denote the number of cases not matching the global optimum in the case when algorithm was successful at least once. The *real italic values* denote the best (upper) and the average (lower) solution found during 100 runs when algorithm was not successful. In the first row the number of individuals is shown. For the test functions not included in the table algorithms were always successful.

Fct	$f(x_i^*)$	GMS	MS	DE	GMS	MS	DE	GMS	MS	DE
		300	300	300	500	500	500	1000	1000	1000
RS	0	0	0	100	0	0	100	0	0	100
RO	0	1	3	0	0	1	0	0	0	0
NU	-4930	0	0	100	0	0	100	0	0	100
SA	0	*0.1*	*0.1*	*0.2*	*0.1*	*0.1*	*0.2*	*0.1*	*0.1*	*0.2*
		0.13	*0.1*	*0.2*	*0.1*	*0.12*	*0.2*	*0.1*	*0.15*	*0.2*
WH	0	6	56	100	6	48	100	12	54	100
RN	-512.7	*-477.2*	*-456.7*	*-262.2*	*-472.3*	*-460.7*	*-263.1*	*-479.5*	*-466.1*	*-269.4*
		-442.3	*-432*	*-228.9*	*-447.9*	*-426.5*	*-231.5*	*-445.8*	*-424.8*	*-238.8*
EG	-	*-26568.1*	*-26549.6*	*-12176.9*	*-26575.6*	*-26477*	*-12235.7*	*-26567.2*	*-26514*	*-13022.5*
		-25736	*-24275*	*-10691*	*-25776*	*-24451*	*-10821*	*-26061*	*-24692*	*-11227*
DP	0	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*
		0.67	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*

Table 3. Best results obtained for 50- and 100-dimensional functions in 100 runs of each algorithm. The integer values denote the number of cases not matching the global optimum in the case when algorithm was successful at least once. The *real italic values* denote the best (upper) and the average (lower) solution found during 100 runs when algorithm was not successful. In the first row the number of individuals is shown. For the test functions not included in the table algorithms were always successful. In case of NU function – optima for 50- and 100-dimensional functions differ, both are presented in column $f(x_i^*)$.

Fct	$f(x_i^*)$	50-dimensional functions						100-dimensional functions		
		GMS	MS	DE	GMS	MS	DE	GMS	MS	DE
		500	500	500	1000	1000	1000	1000	1000	1000
GR	0	0	1	0	0	0	0	0	3	0
AC	0	0	0	0	0	0	0	0	0	0
RS	0	0	0	100	0	0	100	0	0	100
RO	0	1	32	3	0	24	13	3	31	100
LM1	0	0	0	0	0	0	0	0	0	0
LM2	0	0	0	0	0	0	0	0	0	0
NU	-22050 / -171600	0	0	100	0	0	100	0	0	100
SA	0	*0.1 / 0.2*	*0.2 / 0.22*	*0.3 / 0.31*	*0.1 / 0.19*	*0.2 / 0.2*	*0.3 / 0.31*	*0.2 / 0.3*	*0.3 / 0.34*	*0.6 / 0.81*
WH	0	1	45	100	8	56	100	0	54	100
RN	-512.7	*-460.4 / -426.8*	*-438 / -409.5*	*-192.9 / -176.1*	*-480.1 / -431.2*	*-442.9 / -413.6*	*-206.6 / -180.5*	*-438.8 / -409.7*	*-418.8 / -384.9*	*-142.2 / -125.4*
EG	-	*-44843 / -42241*	*-44415 / -38494*	*-15703 / -13.997*	*-44761 / -42517*	*-44478 / -38997*	*-15715 / -13699*	*-89938 / -78038*	*-78142 / -71214*	*-22020 / -18896*
DP	0	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*	*0.67*
SW	-418.9	0	0	0	0	0	0	0	0	100

Table 4. Average number of function calls during 100 runs for 10- and 100-dimensional functions (in thousands). Numbers for 100-dimensional functions are in *italic* (for 1000 individuals only).

Fct	GMS 100	MS 100	DE 100	GMS 300	MS 300	DE 300	GMS 500	MS 500	DE 500	GMS 1000	MS 1000	DE 1000
GR	62	73	82	197	217	247	335	364	413	686 *1284*	724 *1228*	840 *7500*
AC	31	31	29	91	92	88	153	153	147	301 *1686*	303 *1838*	293 *9000*
RS	56	60	85	174	179	263	293	299	438	589 *6371*	595 *6168*	879 *6840*
RO	135	136	1317	365	396	1772	618	665	2466	1232 *16007*	1309 *15025*	4881 *113715*
LM1	13	13	13	40	40	40	66	66	66	130 *644*	130 *633*	131 *6390*
LM2	13	13	13	40	40	39	66	66	64	129 *893*	129 *910*	127 *4997*
NU	102	100	152	300	300	450	500	500	750	1000 *65216*	1000 *18045*	1500 *2915*
SA	100	100	103	300	300	300	500	500	500	1000 *4644*	1000 *3855*	1000 *12175*
WH	105	180	585	465	485	1977	618	797	3232	1560 *4677*	1587 *16147*	6430 *19885*
RN	587	374	214	1810	1090	634	2948	1772	1025	6443 *20896*	505 *12575*	2060 *1540*
EG	626	619	669	2176	1914	1855	3732	3510	2892	7570 *31960*	6810 *18480*	5185 *1555*
DP	85	98	100	283	293	300	491	930	500	453 *2454*	927 *2000*	1045 *11250*
SW	103	100	100	300	300	300	500	500	500	1000 *4000*	1000 *4000*	1000 *3025*

The GMS algorithm does not clearly speed up the optimization process in comparison with the basic DE method (see Table 4). However, it searches the space of admissible solutions much better and, at the slight cost of computational time, achieves much better success rate. When both MS and GMS are able to find global optima, they usually need similar computational time.

It is interesting to trace the exchange of information between four groups, i.e. the different ways of finding the best solution by the proposed GMS algorithm. Because it depends on a particular objective function, our discussion is limited to probably the most interesting case of the 100-dimensional Rana function. In Figs. 7-9 the best objective function values found by each group $G(k)$ after every 5000 iterations are presented for three selected runs of GMS algorithm, which correspond to heavy, medium and low exchange of information. The objective function values of each group are denoted by different patterns: the triangles refer to $G(1)$, squares to $G(2)$, crosslets to $G(3)$ and diamonds to $G(4)$. The change of logical value $LG(k)$ for the first three groups from zero to one, signifying that group k gains information from other groups, is marked by vertical lines. The striped line corresponds to $G(1)$, the solid line to $G(2)$, while the dotted line represents $G(3)$. The striped, solid and dotted vertical lines mean that stopping criterion was reached, but since *StopFlag* was 0 and inequality (6) was fulfilled, the $LG(k)$ of all groups were set to 1 and the algorithm proceeded for the next *PNI* iterations.

In the case depicted in Fig. 7, no improvement was observed by using the proposed GMS algorithm. Although the groups ($G(1)$-$G(3)$) did not share the results, the group $G(2)$ (not the expected $G(4)$) reached the best solution. The final solution found (-382.8) is very poor in comparison with the best one found in 100 runs (-438.8). It looks like the MS algorithm with $K = 2.5M$ would provide the same results much faster.

In Fig. 8, much better performance of the GMS algorithm is shown – each group from time to time makes the total use of information about the local minima found by the other groups. In this case, the alternate periods of exchanging and not exchanging information between groups led to improvement of the performance of the recommended technique. Finally, Fig. 9 depicts an even more complicated and interesting example, in which the individual belonging to the privileged group $G(4)$ found the best optimum. In this case, the solution found was the best among all 100 runs of GMS algorithm. It should be emphasized that many local minima found during the search process successfully provided additional information for $G(4)$.
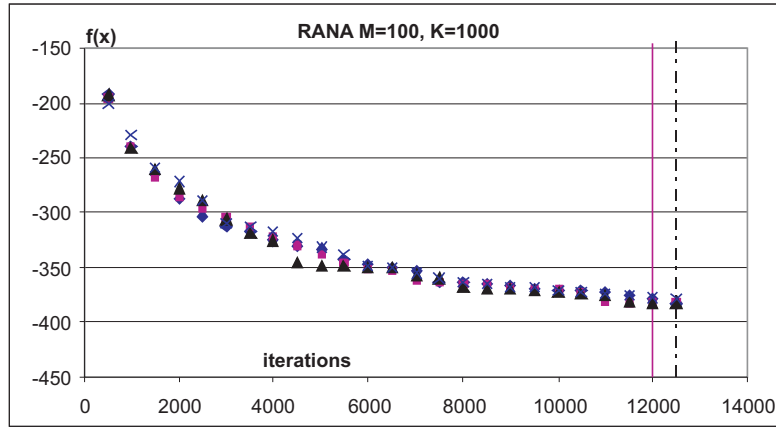
Figure 7. The objective function values found by each group. One example of 100 runs solving the 100-dimensional Rana function. A poor case. G(1) – triangles; G(2) – squares, G(3) – crosslets; G(4) – diamonds. The change of logical value LG(k) for the first three groups from zero to one is marked by vertical lines: LG(1) - the vertical line with strips; LG(2) – the vertical solid line; LG(3) – the vertical line with dots; all LG(k) – vertical line with strips and dots.
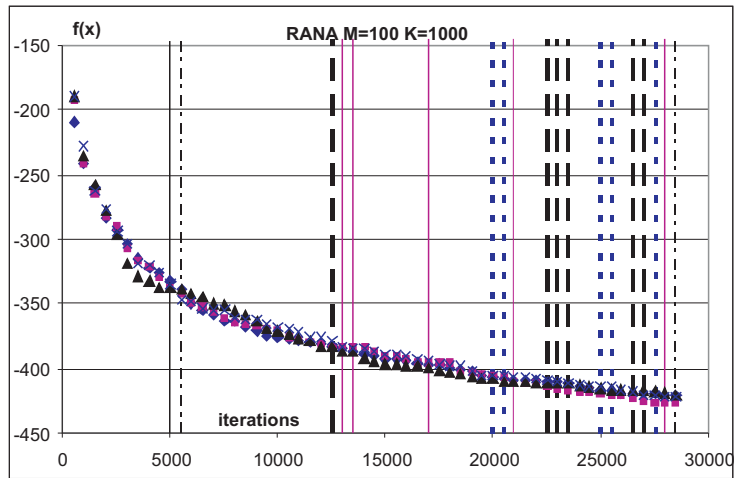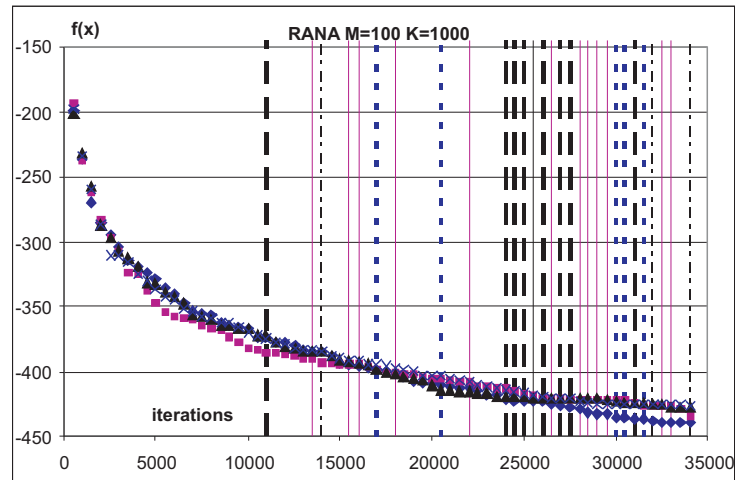

Figure 8. The objective function values found by each group. One example of 100 runs solving the 100-dimensional Rana function. An average case. Notations as in Fig. 6.

Figure 9. The objective function values found by each group. The best of 100 runs solving the 100-dimensional RANA function. Notation as for Fig. 6.

6. Conclusions

In the present paper the Grouped Multi-Strategy Differential Evolution algorithm was proposed for multi-dimensional optimization of continuous real functions. The method was compared with the original Differential Evolution algorithm as well as an approach proposed by Mishra (2006), based on a set of 10- to 100-dimensional test functions of different complexity. The numerical results revealed that the Grouped Multi-Strategy Differential Evolution algorithm clearly provides the best results in terms of minimizing the objective functions, while its superiority to competitors increases with the dimensionality of the problem. For simpler problems the method is much less fallible, whereas for very difficult multi-dimensional test functions, it is almost always able to provide significantly better results than the two other tested approaches.

The impact of the number of individuals in population greater than the heuristic choice recommended by Storn and Price (1995), namely 10-times the dimensionality of the problem, on the performance of the algorithms, was studied. As it can be seen in Table 4, the increase in the number of individuals leads to slowing down the search process and to increase of the number of function evaluations. In most cases this is not advantageous and sometimes may even result in the deterioration of the results.

Acknowledgment

Appendix

1. Griewank function (GR), Price, Storn and Lampinen (2005)

$$f\left(\boldsymbol{x}\right) = \frac{1}{4000}\sum_{j=1}^{M} x_j^2 - \prod_{j=1}^{M} cos\left(\frac{x_j}{\sqrt{j}}\right) + 1$$

$f\left(\boldsymbol{x^*}\right) = 0, \quad x_j^*{=}0;$ in this paper $-1000 \leq x_j \leq 1000$.

2. Ackley function (AC), Price, Storn and Lampinen (2005)

$$f\left(\boldsymbol{x}\right) = -20\exp\left(-0.2\sqrt{\frac{1}{M}\sum_{j=1}^{M} x_j^2}\right) - \exp\left(\frac{1}{M}\sum_{j=1}^{M} cos\left(2\pi x_j\right)\right) + 20 + e$$

$f\left(\boldsymbol{x^*}\right) = 0, \quad x_j^*{=}0;$ in this paper $-32 \leq x_j \leq 32$.

3. Rastrigin function (RS), Price, Storn and Lampinen (2005)

$$f\left(\boldsymbol{x}\right) = \sum_{j=1}^{M} \left(x_j^2 - 10cos\left(2\pi x_j\right) + 10\right)$$

$f\left(\boldsymbol{x^*}\right) = 0, \quad x_j^*{=}0;$ in this paper $-1000 \leq x_j \leq 1000$.

4. Generalized Rosenbrock function (RO), Price, Storn and Lampinen (2005)

$$f\left(\boldsymbol{x}\right) = \sum_{j=1}^{M-1} \left(100\left(x_{j+1} - x_j^2\right)^2 + \left(x_j - 1\right)^2\right)$$

$f\left(\boldsymbol{x^*}\right) = 0, \quad x_j^*{=}1;$ in this paper $-1000 \leq x_j \leq 1000$.

5. Levy and Montalvo 1st function (LM1), Ali, Khompatraporn and Zabinsky (2005)

$$f\left(\boldsymbol{x}\right) = \left(\frac{\pi}{M}\right)\left(10sin^2\left(\pi y_1\right) + \sum_{j=1}^{M-1} \left(y_j - 1\right)^2 \left(1 + 10sin^2\left(\pi y_{j+1}\right)\right) + \left(y_M - 1\right)^2\right)$$

$y_j = 1 + 0.25(x_j + 1)$,
$f\left(\boldsymbol{x^*}\right) = 0, \ x_j^* = -1; \ -10 \leq x_j \leq 10$.

6. Levy and Montalvo 2nd function (LM2), Ali, Khompatraporn and Zabinsky (2005)

$$f\left(\boldsymbol{x}\right) = 0.1\Big(sin^2\left(3\pi x_1\right) + \sum_{j=1}^{M-1} \left(x_j - 1\right)^2 \left(1 + sin^2\left(3\pi x_{j+1}\right)\right)$$
$$+ \left(x_M - 1\right)^2 \left(1 + sin^2\left(2\pi x_M\right)\right)\Big)$$

$f\left(\boldsymbol{x^*}\right) = 0, \ x_j^* = 1; \ -5 \leq x_j \leq 5$.

7. Neumaier 3 function (NU), Ali, Khompatraporn and Zabinsky (2005)

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} (x_j - 1)^2 - \sum_{j=2}^{M} x_j x_{j-1}$$

$$f(\boldsymbol{x^*}) = -M(M+4)(M-1)/6, \ x_j^* = j(M+1-j); \ -M^2 \le x_j \le M^2.$$

8. Salomon function (SA), Ali, Khompatraporn and Zabinsky (2005)

$$f(\boldsymbol{x}) = 1 - cos\left(2\pi\sqrt{\sum_{j=1}^{M} x_j^2}\right) + 0.1\sqrt{\sum_{j=1}^{M} x_j^2}$$

$$f(\boldsymbol{x^*}) = 0, \ x_j^* = 0, \ -100 \le x_j \le 100.$$

9. Whitley function (WH), Whitley et al. (1996), Price, Storn and Lampinen (2005)

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} \sum_{l=1}^{M} \left(\frac{\left(100\left(x_j - x_l^2\right)^2 + (1-x_l)^2\right)^2}{4000} \right) - cos\left(100\left(x_j - x_l^2\right)^2 \right.$$
$$\left. + (1-x_l)^2\right) + 1$$

$$f(\boldsymbol{x^*}) = 0, \ x_j^* = 1; \ -100 \le x_j \le 100.$$

10. Normalized Rana function (RN), Whitley et al. (1996), Price, Storn and Lampinen (2005), www.it.lut.fi/ip/evo/functions

$$f(\boldsymbol{x}) = \sum_{j=1}^{M} x_j sin\left(\sqrt{|x_l + 1 - x_j|}\right) cos\left(\sqrt{|x_l + 1 + x_j|}\right) +$$
$$(x_l + 1) cos\left(\sqrt{|x_l + 1 - x_j|}\right) sin\left(\sqrt{|x_l + 1 + x_j|}\right)$$

$$l = (j+1)Mod(M) \ f(\boldsymbol{x^*}) = -512.7531624, \ x_j^* = -514.04168,$$
$$-520 \le x_j \le 520.$$

11. Eggholder function (EG), Whitley et al. (1996), Adorio (2005)

$$f(\boldsymbol{x}) = \sum_{j=1}^{M-1} -(x_{j+1} + 47) sin\sqrt{|x_{j+1} + x_j 0.5 + 47|}$$
$$+ sin\sqrt{|x_j - (x_{j+1} + 47)|}(-x_j)$$

$$f(\boldsymbol{x^*}) \text{ and } x_j^* \text{ depend on } M, \ -512 \le x_j \le 512$$

12. Dixon-Price function (DP), Heddar and Fukushima (2006)

$$f(\boldsymbol{x}) = (x_1 - 1)^2 + \sum_{j=2}^{M} j \left(2x_j^2 - x_{j-1}\right)^2$$

$$f(\boldsymbol{x^*}) = 0, \ x_j^* = 2^{-\left(\frac{2^j-2}{2^j}\right)}, \ -10 \le x_j \le 10.$$

13. Schwefel function (SW), Price, Storn and Lampinen (2005)

$$f(\boldsymbol{x}) = -\frac{1}{M} \sum_{j=1}^{M} x_j sin\left(\sqrt{|x_j|}\right)$$

$$f(\boldsymbol{x^*}) = -418.983, \ x_j^* = 420.9687, \ -500 \le x_j \le 500.$$

References

ADRIO, E.P. (2005) Multivariate Test Functions Library in C for unconstrained global optimization. http://geocities.com/eadorio/mvf.pdf.

ALI, M.M., KHOMPATRAPORN, C. and ZABINSKY, Z.B. (2005) A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization* **31**, 635–672.

ALI, M.M. (2007) Differential Evolution with preferential crossover. *European Journal of Operational Research* **181**, 1137–1147.

BREST, J., BOSKOVIC, B., GREINER, S., ZUMER, V. and MAUCEC, M.S. (2007) Performance comparison of Self-Adaptive and Adaptive Differential Evolution algorithms. *Soft Computing* **11**, 7, 617–629.

CLERC, M. (2006) *Particle Swarm Optimization.* ISTE Ltd, London.

DAS, S., ABRAHAM, A., CHAKRABORTY, U.K. and KONAR, A. (2009) Differential Evolution Using a Neighborhood-based Mutation Operator. *IEEE Transactions on Evolutionary Computations* **13**, 3, 526–553.

GUSTAFSON, S. and BURKE, E.K. (2006) The Speciating Island Model: An alternative parallel evolutionary algorithm. *Journal of Parallel and Distributed Computing* **66** (8), 1025–1036.

HEDDAR, A.R. and FUKUSHIMA, M. (2006) Tabu Search directed by direct search methods for nonlinear global optimization. *European Journal of Operational Research* **170**, 329–349.

HOLLAND, J.H. (2000) Building blocks, cohort genetic algorithms and hyperplane - defined functions. *Evolutionary Computation* **8** (4), 373–391.

ILONEN, J., KAMARAINEN, J.K. and LAMPINEN, J. (2003) Differential Evolution training algorithm for feed-foreward neural networks. *Neural Processing Letters* **17**, 93–105.

KAELO, P. and ALI, M.M. (2006) A numerical study of some modified Differtential Evolution algorithms. *European Journal of Operational Research* **169**, 1176–1184.

LIU, J. and LAMPINEN, J. (2005) A Fuzzy Adaptive Differential Evolution algorithm. *Soft Computing* **9**, 448–462.

LIU, Y., YAO, X. and HIGRUCHI, T. (2000) Evolutionary ensembles with negative correlation learning. *IEEE Transactions on Evolutionary Computation* **4** (4), 380–387.

MISHRA, S.K. (2006) Global optimization by Differential Evolution and Particle Swarm methods evaluation on some benchmark functions. Social Science Research Network, *Working Papers Series*, http://ssrn.com/ abstract=933827.

PRICE, K.V., STORN, R.M. and LAMPINEN, J.A. (2005) *Differential Evolution. A Practical Approach to Global Optimization.* Springer-Verlag, Berlin-Heidelberg.

QIN, A.K., HUANG, V.L. and SUGANTHAN, P.N. (2009) Differential Evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions Evolutionary Computation* **13** (2), 398–417.

ROWIŃSKI, P.M. and PIOTROWSKI, A. (2008) Estimation of parameters of transient storage model by means of multi-layer perceptron neural networks. *Hydrological Sciences Journal* **53** (1), 165-178.

STORN, R. and PRICE, K.V. (1995) Differential Evolution – a simple and efficient adaptive scheme for global optimization over continuous spaces. *Technical Report* TR-95-012, International Computer Sciences Institute, Berkeley, CA, USA .

STORN, R. and PRICE, K.V. (1997) Differential Evolution – a simple and efficient heuristic for global optimization over continuous spaces. *Journal of Global Optimization* **11** (1), 341–359.

WANG, Y.J. and ZHANG, J.S. (2007) Global optimization by an improved Differential Evolutionary algorithm. *Applied Mathematics and Computation* **188**, 669–680.

WHITLEY, D., RANA, S., DZUBERA, J. and MATHIAS, K.E. (1996) Evaluating evolutionary algorithms. *Artificial Intelligence* **85**, 245–276.

WHITLEY, D., LUNACEK, M. and KNIGHT, J. (2004) Ruffled by Ridges: How Evolutionary Algorithms Can Fail. In: **LNCS 3103**, Springer, Berlin-Heidelberg.

TAO, J. and WANG, N. (2007) DNA computing based RNA genetic algorithm with applications in parameter estimation of chemical engineering processes. *Computers and Chemical Engineering* **31**, 1602–1618.