HYBRID RULES OF PERTURBATION IN DIFFERENTIAL EVOLUTION FOR DYNAMIC OPTIMIZATION

Mikołaj Raciborski[1], Krzysztof Trojanowski[2] and Piotr Kaczyński[1]
[1]*Cardinal Stefan Wyszyński University, Faculty of Mathematics and Natural Sciences, College of Sciences, Warsaw, Poland*
[2]*Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland*

Keywords: Adaptive differential evolution, Dynamic optimization, Symmetric α-stable distribution.

Abstract: This paper studies properties of a differential evolution approach (DE) for dynamic optimization problems. An adaptive version of DE, namely the jDE algorithm has been applied to two well known benchmarks: Generalized Dynamic Benchmark Generator (GDBG) and Moving Peaks Benchmark (MPB) reimplemented in a new benchmark suite `Syringa`. The main novelty of the presented research concerns application of new type of solution, that is, solution mutated with an operator originated from another metaheuristics. The operator uses a symmetric α-stable distribution variate for modification of the solution coordinates.

1 INTRODUCTION

Optimization in dynamic environments is a continuous subject of interest for many research groups. In the case of dynamic optimization the algorithm has to cope with changes in the fitness landscape, i.e., in the evaluation function parameters or even in the evaluation function formula which appear during the process of optimum search. There exists a number of dynamic optimization benchmarks designed to estimate efficiency of optimization algorithms. Among these benchmarks we selected two with the search space defined in $\mathbf{R}^n$ to evaluate the differential evolution (DE) approach and especially the idea of hybrid population application. DE approach which originated with the Genetic Annealing algorithm (Price, 1994) has been studied from many points of view (for detailed discussion see, for example, monographs (Feokistov, 2006; Price et al., 2005)). The adaptive version of the DE algorithm (Brest et al., 2006) differs form the basic approach in that a self-adaptive control mechanism is used to change the control parameters F and CR during the run.

In the presented research we are interested in verification of the positive or negative role of a mutation operator originating from another heuristic approach when it is applied in the DE algorithm. The proposed type of mutation employs random variates controlled by the α-stable distribution which already proved their usefulness in other version of evolutionary approach and in the particle swarm optimization.

The paper is organized as follows. In Section 2 a brief description of the optimization algorithm is presented. Section 3 discuss properties of new type of mutation introduced to jDE. The description of a new benchmark suite `Syringa` is given in Section 4 whereas Section 5 includes some details of the applied measure and the range of the performed tests. Section 6 shows the results of experiments. Section 7 concludes the presented research.

2 THE jDE ALGORITHM

The differential evolution algorithm is an evolutionary method with a very specific mutation operator controlled by the scale factor F. Three different, randomly chosen solutions are needed to mutate a target solution $\mathbf{x}^i$: a base solution $\mathbf{x}^0$ and two difference solutions $\mathbf{x}^1$ and $\mathbf{x}^2$. Then, a mutant undergoes discrete recombination with the target solution which is controlled by the crossover probability factor $CR \in [0,1]$. Finally, in the selection stage trial solutions compete with their target solutions for the place in the population. This strategy of population management is called *DE/**rand/1/bin*** which means that the base solution is **rand**omly chosen, **1** difference vector is added to it and the crossover is based on a set of independent decisions for each of coordinates, that is, a number of parameters donated by the mutant closely follows a **bin**omial distribution.

Raciborski M., Trojanowski K. and Kaczyński P..
HYBRID RULES OF PERTURBATION IN DIFFERENTIAL EVOLUTION FOR DYNAMIC OPTIMIZATION.
DOI: 10.5220/0003671800320041
In *Proceedings of the International Conference on Evolutionary Computation Theory and Applications* (ECTA-2011), pages 32-41
ISBN: 978-989-8425-83-6

Algorithm 1: jDE algorithm.

```
1: Create and initialize the reference set of $(k \cdot m)$ solutions
2: repeat
3:     for $l = 1$ to $k$ do {for each subpopulation}
4:         for $i = 1$ to $m$ do {for each solution in a subpopulation}
5:             Select randomly three solutions: $\mathbf{x}^{l,0}$, $\mathbf{x}^{l,1}$, and $\mathbf{x}^{l,2}$
                   such that: $\mathbf{x}^{l,i} \neq \mathbf{x}^{l,0}$ and $\mathbf{x}^{l,1} \neq \mathbf{x}^{l,2}$
6:             for $j = 1$ to $n$ do {for each dimension in a solution}
7:                 if $(rand(0,1) > CR^{l,i})$ then
8:                     $u_j^{l,i} = x_j^{l,0} + F^{l,i} \cdot (x_j^{l,1} - x_j^{l,2})$
9:                 else
10:                    $u_j^{l,i} = x_j^{l,i}$
11:                end if
12:            end for
13:        end for
14:    end for
15:    for $i = 1$ to $(k \cdot m)$ do {for each solution}
16:        if $(f(\mathbf{u}^i) < f(\mathbf{x}^i))$ then {Let's assume this is a minimization problem}
17:            $\mathbf{x}^i = \mathbf{u}^i$
18:        end if
19:        Recalculate $F^i$ and $CR^i$
20:        Apply aging for $\mathbf{x}^i$
21:    end for
22:    Do overlapping search
23: until the stop condition is satisfied
```

The jDE algorithm (depicted in Figure 1) extends functionality of the basic approach in many ways. First, each object representing a solution in the population is extended by a couple of its personal parameters CR and F. They are adaptively modified every generation (Brest et al., 2006). The next modifications have been introduced just for better coping in the dynamic optimization environment. The population of solutions has been divided into five subpopulations of size ten. Each of them has to perform its own search process, that is, no information is shared between subpopulations. Every solution is a subject to the aging procedure protecting against stagnation in local minima and just the global-best solution is excluded from this. To avoid overlapping between subpopulations a distance between subpopulation leaders is calculated and in the case of too close localization one of subpopulations is reinitialized. However, as in previous case the subpopulation with the global-best is never the one to reinitialize. The last extension is a memory structure called archive. The archive is increased after each change in the fitness landscape by the current global-best solution. Recalling from the archive can be executed every reinitialization of a subpopulation, however, decision about the execution depends on a few conditions. Details of the above-mentioned extensions can be found in (Brest et al., 2009).

3 PROPOSED EXTENSION OF jDE

The novelty in the algorithm concerns introduction of new type of solutions into the population of size M. A small number of new solutions (just one, two, or three pieces) replace the classic ones so the population size remains the same. The difference between the classic solutions and the new ones lies in the way they are mutated. New type of mutation operator is based on the rules of movement governing quantum particles in mQSO (Trojanowski, 2009).

In the first phase of the mutation, we generate a new point uniformly distributed within a hypersphere surrounding the mutated solution. In the second phase, the point is shifted along the direction determined by the hypersphere center and the point. The distance d' from the hypersphere center to the final location of the point is calculated as follows:

$$d' \quad = \quad d \cdot S\alpha S(0,\sigma) \cdot \exp(-f'(\mathbf{x}_i)), \qquad (1)$$

where d is a distance from the original location obtained in the first phase, $S\alpha S(\cdot,\cdot)$ denotes a symmetric α-stable distribution variate, σ is evaluated as in eq. (2) and $f'(\mathbf{x}_i)$ as in eq. (3):

$$\sigma = r_{S\alpha S} \cdot (D_{\mathrm{w}}/2) \tag{2}$$

$$f'(\mathbf{x}_i) = \frac{f(\mathbf{x}_i) - f_{\min}}{(f_{\max} - f_{\min})} \tag{3}$$

where:

$$f_{\max} = \max_{j=1,\ldots,M} f(\mathbf{x}_j), \qquad f_{\min} = \min_{j=1,\ldots,M} f(\mathbf{x}_j).$$

The α-stable distribution (called also a Levý distribution) is controlled by four parameters: stability index α ($\alpha \in (0,2]$), skewness parameter β, scale parameter σ and location parameter μ. In our case we assume $\mu = 0$ and apply the symmetric version of this distribution (denoted by $S\alpha S$ for "symmetric α-stable distribution"), where β is set to 0.

The resulting behavior of the proposed operator is characterized by two parameters: the parameter $r_{S\alpha S}$ which controls the mutation strength, and the parameter α which determines the shape of the α-stable distribution. The solutions mutated in this way are labeled as $\mathbf{s}_{\text{Levý}}$ in the further text.

4 THE `Syringa` BENCHMARK SUITE

For the experimental research we developed a new testing environment `Syringa` which is able to simulate behavior of a number of existing benchmarks and to create completely new instances as well. The structure of the `Syringa` code originates from a fitness landscape model where the landscape consists of a number of simple components. A sample dynamic landscape consists of a number of components of any types and individually controlled by a number of parameters. Each of the components covers a subspace of the search space. The final landscape is the result of a union of a collection of components such that each of the solutions from the search space is covered by at least one component. In the case of a solution belonging to the intersection of a number of components the solution value equals (1) the minimum (for minimization problems) or (2) maximum (otherwise) value among the values obtained for the intersected components or (3) this can be also a sum of the fitness vales obtained from these components (not used in our case). Eventually, the `Syringa` structure is a logical consequence of the following assumptions:

1. the fitness landscape consists of a number of any different component landscapes,
2. the dynamics of each of the components can be different and individually controlled,
3. a component can cover a part or the whole of the search space, thus, in case of a solution covered by more than one component the value of this solution can be a minimum, a maximum or a sum of values returned by the covering components.

4.1 The Components

The current version of `Syringa` consists of six types of component functions (Table 1) defined for the real-valued search space. All of the formulas include the number of the search apace dimensions n which makes them able to define for the search spaces of any given complexity.

There can be defined a number of parameters which individually define the component properties and allow to introduce dynamics as well. For each of the components we can define two groups of parameters which influence the formula of the component fitness function: the parameters from the former one are embedded in the component function formula whereas the parameters from the latter one control rather the output of the formula application. For example, when we want to stretch the landscape over the search space each of the solution coordinates is multiplied by the scaling factor. For a non–uniform stretching we need to use a vector of factors containing individual values for each of the coordinates. We call this type of modification a horizontal scaling and this represents the first type of component changes. The example of the second type is a vertical scaling where just the fitness value of a solution is multiplied by a scaling factor. The first group of parameters controls changes like horizontal translation, horizontal scaling, and rotation. For simplicity they are called *horizontal changes* in the further text. The second group of changes (called respectively *vertical changes*) is represented by vertical scaling and vertical translation. All of the changes can be obtained by dynamic modification of respective parameters during the process of search.

4.2 Horizontal Change Parameters

In this case the coordinates of the solution $\mathbf{x}$ (a vector, i.e., a matrix of size n by 1) are modified before the component function equation is applied. The new coordinates are obtained with the following formula:

$$\mathbf{x}' = \mathbf{M} \cdot (\mathbf{W} \cdot (\mathbf{x} + \mathbf{X})) \tag{4}$$

where $\mathbf{X}$ is a translation vector, $\mathbf{W}$ is a diagonal matrix of scaling coefficients for the coordinates, and $\mathbf{M}$ is an orthogonal rotation matrix.

Table 1: `Syringa` components.

name	formula	domain
Peak (F_1)	$f(\mathbf{x}) = \frac{1}{1+\sum_{j=1}^{n} x_j^2}$	[-100,100]
Cone (F_2)	$f(\mathbf{x}) = 1 - \sqrt{\sum_{j=1}^{n} x_j^2}$	[-100,100]
Sphere (F_3)	$f(\mathbf{x}) = \sum_{i=1}^{n} x_i^2$	[-100,100]
Rastrigin (F_4)	$f(\mathbf{x}) = \sum_{i=1}^{n} (x_i^2 - 10\cos(2\pi x_i) + 10)$	[-5,5]
Griewank (F_5)	$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n}(x_i)^2 - \prod_{i=1}^{n}\cos(\frac{x_i}{\sqrt{i}}) + 1$	[-100,100]
Ackley (F_6)	$f(\mathbf{x}) = -20\exp(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}) - \exp(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)) + 20 + e$	[-32,32]

4.3 Vertical Change Parameters

Changes of the fitness function value are executed according to the following formula:

$$f'(\mathbf{x}) = f(\mathbf{x}) \cdot v + h \qquad (5)$$

where v is a vertical scaling coefficient and h is a vertical translation coefficient.

4.4 Parameters Control

In the case of dynamic optimization the fitness landscape components has to change the values of their parameters during the process of search. There were defined four different characteristics of variability which were applied to the component parameters: small step change ($\mathbf{T}_1$ — eq. (6)), large step change ($\mathbf{T}_2$ — eq. (7)), and two versions of random changes ($\mathbf{T}_3$ — eq. (8) and $\mathbf{T}_4$ — eq. (9)). The change Δ of a parameter value is calculated as follows:

$$\begin{aligned}
\Delta &= \alpha \cdot r \cdot (\max - \min);\ \alpha = 0.04,\ r = U(0,1), && (6)\\
\Delta &= (\alpha \cdot sign(r_1) + (\alpha_{\max} - \alpha) \cdot r_2) \cdot (\max - \min);\\
&\quad \alpha = 0.04,\ r_{1,2} = U(0,1),\ \alpha_{\max} = 0.1 && (7)\\
\Delta &= N(0,1) && (8)\\
\Delta &= U(r_{\min}, r_{\max}) && (9)
\end{aligned}$$

In the above-mentioned equations max and min represent upper and lower boundary of the search space, $N(0,1)$ is a random value obtained with gaussian distribution where $\mu = 0$ and $\sigma = 1$, $U(0,1)$ is a random value obtained with uniform distribution form the range $[0,1]$, and $[r_{\min}, r_{\max}]$ define the feasible range of Δ values.

The model of `Syringa` assumes that the component parameter control is separated from the component, that is, a dynamic component has to consist of two objects: the first one represents an *evaluator* of solutions (i.e., a component of any type mentioned in Table 1) and the second one is an *agent* which controls the behavior of the evaluator. The agent defines initial set of values for the evaluator parameters and during the process of search the values are updated by the agent according to the assumed characteristic of variability. Properties of all the types of components are unified so as to make possible assignment of any agent to any component. This architecture allows to create multiple classes of dynamic landscapes.

In the presented research we started with simulation of two existing benchmarks: Generalized Dynamic Benchmark Generator (GDBG) (Li and Yang, 2008) and the Moving Peaks Benchmark generator (Branke, 1999). In both cases optimization is carried out in a real-valued multidimensional search space, and the fitness landscape is built of multiple component functions controlled individually by their parameters. For appropriate simulation of any of the two benchmarks there are just two things to do: select a set of the right components and build agents which will control the components in the same manner like in the original benchmarks.

4.5 Simulation of Moving Peaks Benchmark (MPB)

In the case of MPB, three scenarios of the benchmark parameters control are defined (Branke, 1999). We did experiments for the first and the second scenario.

The selected fitness landscape consists of a set of peaks (F_1 — the first scenario) or cones (F_2 — the second scenario) which undergo two types of horizontal changes: the translation and the scaling and just one vertical change, that is, the translation. The horizontal scaling operator has the same scale coefficient for each of the dimensions, so in this specific case this coefficient is represented as a one-dimensional variable w instead of the vector $\mathbf{W}$.

The parameters $\mathbf{X}$, w and v are embedded into the peak function formula $f(\mathbf{x})$ in the following way:

$$f_{\text{peak}}(\mathbf{x}) = \frac{v}{1 + w \cdot \sum_{j=1}^{n} \frac{(\mathbf{x}_j - \mathbf{X}_j)^2}{n}} \tag{10}$$

The parameters $\mathbf{X}$, w and h are embedded into the cone function formula $f(\mathbf{x})$ in the following way:

$$f_{\text{cone}}(\mathbf{x}) = h - w \cdot \sqrt{\sum_{j=1}^{n} (\mathbf{x}_j - \mathbf{X}_j)^2} \tag{11}$$

All the modifications of the component parameters belong to the fourth characteristic of variability $\mathbf{T}_4$ where for every change $r_{\min}$ and $r_{\max}$ are redefined in the way to keep the value of each modified parameter in the predefined interval of feasible values. Simply, for every modified parameter of translation or scaling, which can be represented as a symbol p: $r_{\min} = p_{\min} - p$ and $r_{\max} = p_{\max} - p$. For the horizontal scaling the interval is set to $[1;12]$ and for the vertical scaling — to $[30;70]$. For the horizontal translation there is a constraint for the euclidean length of the translation: $|\mathbf{X}| \leq 3$. For both scenarios in the first version there are ten moving components whereas in the second version 50 moving components is in use.

4.6 Simulation of Generalized Dynamic Benchmark Generator (GDBG)

GDBG consists of two different benchmarks: Dynamic Rotation Peak Benchmark Generator (DRPBG) and Dynamic Composition Benchmark Generator (DCBG). There are five types of component functions: peak (F_1), sphere (F_3), Rastrigin (F_4), Griewank (F_5), and Ackley (F_6). F_1 is the base component for DRPBG whereas all the remaining types are employed in DCBG.

4.6.1 Dynamic Rotation Peak Benchmark Generator (DRPBG)

There are four types of the component parameter modification applied in DRPBG: horizontal translation, scaling and rotation and vertical scaling. As in the case of MPB the horizontal scaling operator has the same scale coefficient for each of the dimensions, so in this specific case this coefficient is also represented as a one-dimensional variable w instead of the vector $\mathbf{W}$. The component function formula is the same as in the eq. (10).

Values of the translation vector $\mathbf{X}$ in subsequent changes are evaluated with use of the rotation matrix $\mathbf{M}$. Clearly, we apply the rotation matrix to the current coordinates of the component function optimum $\mathbf{o}$, that is: $\mathbf{o}(t+1) = \mathbf{o}(t) \cdot \mathbf{M}(t)$ (where t is the number of the current change in the component) and then the final value of $\mathbf{X}(t+1)$ is calculated: $\mathbf{X}(t+1) = \mathbf{o}(t+1) - \mathbf{o}(0)$.

Subsequent values of the horizontal scaling parameter w and the vertical scaling parameter v are evaluated according to the first, the second or the third characteristic of variability, that is, $\mathbf{T}_1$, $\mathbf{T}_2$ or $\mathbf{T}_3$.

For every change a new rotation matrix $\mathbf{M}$ is generated which is common for all the components. The rotation matrix $\mathbf{M}$ is obtained as a result of multiplication of a number of rotation matrices R where each of R represents rotation in just one plane of the multi-dimensional search space.

In DRPBG we start with a selection of the rotation planes, that is, we need to generate a vector $\mathbf{r}$ of size l where l is an even number and $l \leq n/2$. The vector contains randomly selected search space dimension identifiers without repetition. Then for each of the planes defined in $\mathbf{r}$ by subsequent pairs of indices: $[1,2]$, $[3,4]$, $[5,6]$, $\ldots$ $[l-1,l]$ their rotation angles are randomly generated and finally respective matrices $R_{r[1],r[2]}, \cdots R_{r[l-1],r[l]}$ are calculated ($R_{ij}(\theta)$ represents rotation by the θ angle along the plane i–j) (c.f. (Salomon, 1996)). Eventually, the rotation matrix $\mathbf{M}$ is calculated as follows:

$$M(t) = R_{r[1],r[2]}(\theta(t)) \cdots R_{r[l-1],r[l]}(\theta(t)). \tag{12}$$

In `Syringa` the method of the rotation matrix generation slightly differs from the one described above. Instead of the vector $\mathbf{r}$ there is a vector Θ which represents a sequence of rotation angles for all the possible planes in the search space. The position in the vector Θ defines the rotation plane. Simply, $\Theta(1)$ represents the plane [1,2], $\Theta(2)$ represents the plane [2,3] and so on until the plane [n-1,n]. Then there go planes created from every second dimensions, that is, [1,3], [2,4] and so on until the plane [n-2,n]. Then there go planes created from every third dimensions, then those created from every fourth, and so on until there appears the last plane created from the first and the last dimension. When $\Theta(i)$ equals zero this means, that there is no rotation for the i-th plane, otherwise the respective rotation matrix R is generated. The final stage of generation of the matrix $\mathbf{M}$ is the same as in the description above, that is, the rotation matrix $\mathbf{M}$ is the result of multiplication of all the matrices R generated from the vector Θ.

Finally, it is important to note that in the above description the matrix $\mathbf{M}$ has been used twice for the evaluation of the component modification parameters: the first time when the translation vector $\mathbf{X}$ is calculated and the second time when the rotation is applied, that is, just before the application of the equation (10).

4.6.2 Dynamic Composition Benchmark Generator (DCBG)

DCBG performs five types of the component parameter modification: horizontal translation, scaling and rotation and vertical translation and scaling. The respective parameters are embedded into the function formula $f"(\mathbf{x})$ in the following way (Liang et al., 2005; Suganthan et al., 2005):

$$f"(\mathbf{x}) = (v \cdot (f'(\mathbf{M} \cdot (\mathbf{W} \cdot (\mathbf{x} + \mathbf{X}))) + h)) \quad (13)$$

where:

v is the weight coefficient depending of the currently evaluated $\mathbf{x}$,
$\mathbf{W}$ is called a stretch factor which equals 1 when the search range of $f(\mathbf{x})$ is the same as the entire search space and grows when the search range of $f(\mathbf{x})$ decreases,
$f'(\mathbf{x})$ represent the value of $f(\mathbf{x})$ normalized in the following way: $f'(\mathbf{x}) = C \cdot f(\mathbf{x})/|f_{\max}|$ where the constant $C = 2000$ and $f_{\max}$ is the estimated maximum value of function f which is one of the four: sphere (F_3), Rastrigin (F_4), Griewank (F_5), or Ackley (F_6).

In `Syringa` the properties of some of the parameters has had to be changed. The first difference is in the evaluation of the weight coefficient v which due to the structure of the assumed model cannot depend of the currently evaluated $\mathbf{x}$. Therefore, we assumed that $v = 1$. There is also no scaling, i.e., $\mathbf{W}$ is an identity matrix because we assumed that the component functions are always defined for the entire search space. The last issue is about the rotation matrix $\mathbf{M}$ which is calculated in the same way as for the `Syringa` version of DRPBG. Eventually, the `Syringa` version of $f"(\mathbf{x})$ has the following shape:

$$f"(\mathbf{x}) = ((f'(\mathbf{M} \cdot ((\mathbf{x} + \mathbf{X}))) + h)) \quad (14)$$

Thus, the `Syringa` version of DCBG differs from the original one because it does not contain the horizontal scaling, the rotation matrix $\mathbf{M}$ is evaluated in the different way and the stretch factor always equals one. However, a kind of the vertical scaling is still present and can be found in the step of the $f(\mathbf{x})$ normalization.

Fitness landscapes of the selected benchmark instances generated for two-dimensional search space are depicted in Figure 1. Please note that the landscape generated by DRPBG with ten peak components is in fact exactly the same as the one which would be generated by MPB sc.1 with ten peak components since in both cases the same component function is in use. Of course the dynamics of the two benchmark instances is different so there is no chance for duplication of the experiments.

5 PLAN OF EXPERIMENTS

5.1 Performance Measure

For comparisons between the results obtained for different benchmark instances and different versions of the algorithms the offline error (Branke, 1999) (briefly *oe*) was selected. The measure represents the average deviation from the optimum of the best individual value since the last change in the fitness landscape. Formally:

$$oe = \frac{1}{N_{\text{changes}}} \sum_{j=1}^{N_{\text{changes}}} \left(\frac{1}{N_{\text{e}}(j)} \sum_{i=1}^{N_{\text{e}}(j)} \left(f(\mathbf{x}^{*j}) - f(\mathbf{x}_{\text{best}}^{ji})\right) \right), \quad (15)$$

where $N_{\text{e}}(j)$ is a total number of solution evaluations performed for the j-th static state of the landscape, $f(\mathbf{x}^{*j})$ is the value of an optimal solution for the j-th landscape and $f(\mathbf{x}_{\text{best}}^{ji})$ is the current best value found for the j-th landscape. It should be clear that the measure *oe* should be minimized, that is, the better result the smaller the value of *oe*.

Our algorithm has no embedded strategy for detecting changes in the fitness landscapes. Simply, the last step in the main loop of the algorithm executes the reevaluation of the entire current solution set. Therefore, our optimization system is informed of the change as soon as it occurs, and no additional computational effort for its detection is needed.

5.2 The Tests

We performed experiments with a subset of GDBG benchmark functions as well as with four versions of MPB. For each of the components there is a defined feasible domain which, unfortunately, is not the same in every case (see the last column in Table 1). For this reason the boundaries for the search space dimensions in the test-cases are not the same but adjusted respectively to the components. For GDBG the feasible search space is within the hypercube with the same boundaries for each dimension, namely $[-7.1, 7.1]$ whereas for MPB — $[-50, 50]$. These box constraints mean that both solutions and components should not leave this area during the entire optimization process.

The number of fitness function evaluations between subsequent changes was calculated according to the rules as in the CEC'09 competition, that is, for $10^4 \cdot n$ fitness function calls between subsequent changes where n is a number of search space dimensions and in this specific case n equals five for all of the benchmark instances.

To decrease the number of algorithm configurations which would be experimentally verified we de-

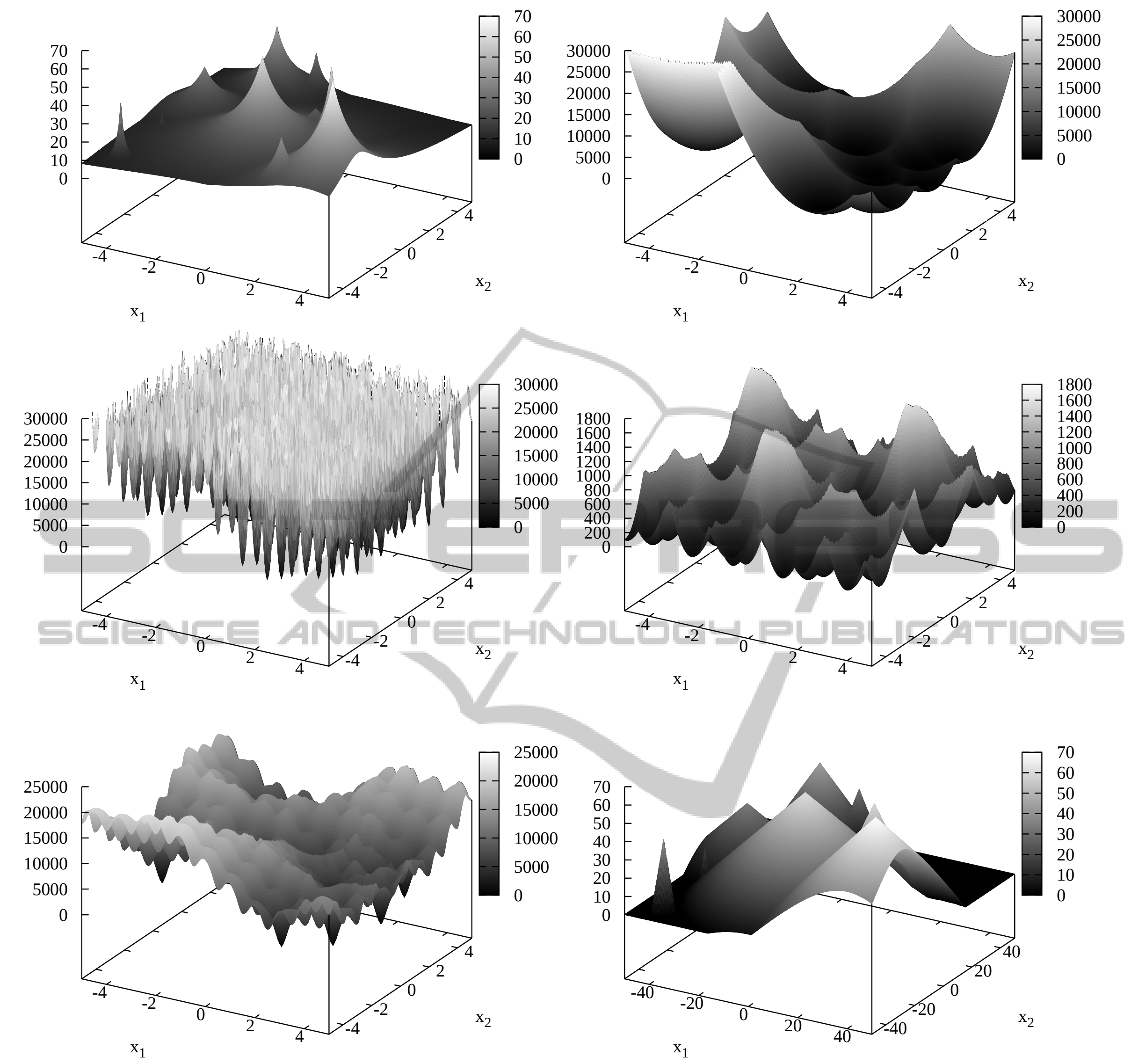

Figure 1: Fitness landscapes of the benchmark instances for 2D search space: DRPBG with ten peak components and DCBG with ten sphere components (first row), DCBG with ten Rastrigin components and DCBG with ten Griewank components (second row), DCBG with ten Ackley components and MPB sc. 2 with ten cones (the last row).

cided to fix the value of the parameter $r_{S\alpha S}$ and the only varied parameter was α. In the preliminary phase of experimental research we tested efficiency of the algorithm for different values of $r_{S\alpha S}$ and analyzed obtained values of error. Eventually, for GDBG $r_{S\alpha S} = 0.6$ whereas for MPB it is ten times smaller, that is, $r_{S\alpha S} = 0.06$. Thus, for each of the benchmark instances there were performed just 32 experiments: for α between 0.25 and 2 varying with step 0.25 and for 0, 1, 2 and 3 solutions of new type present in the population. For each of the configurations the experiments were repeated 20 times and each of them consisted of 60 changes in the fitness landscape. Graphs and tables present mean values of *oe* calculated for these series.

6 THE RESULTS

The best values of offline error obtained for each of the benchmark instances are presented in Table 2. The table contains names of the instances, the mutation parameter configurations (i.e. values of α and $ind_{S\alpha S}$) and the mean values of *oe*. The instances are sorted in ascending order of obtained *oe* values (more or less). This way we can easily show which of the instances

Table 2: The least offline error (*oe*) mean values obtained for all the benchmark instances; the instances are sorted in ascending order of *oe*; for each of the instances three values are presented for three change types of the GDBG control parameters: $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ (except for MPB where there are two cases: with ten and 50 peaks/cones).

the benchmark instance	change type	α	$ind_{S\alpha S}$	*oe*
F_5	$\mathbf{T}_1$	1.25	3.00	0.16813
(Griewank)	$\mathbf{T}_2$	1.50	3.00	0.13977
	$\mathbf{T}_3$	2.00	3.00	0.35606
MPB sc. 1 with ten peaks		1.00	1.00	0.35622
MPB sc. 1 with 50 peaks		1.50	1.00	0.66492
DCBG with F_3	$\mathbf{T}_1$	1.50	2.00	1.22583
(sphere components)	$\mathbf{T}_2$	0.75	1.00	1.81625
	$\mathbf{T}_3$	1.75	3.00	5.32476
DRPBG with ten F_1	$\mathbf{T}_1$	2.00	1.00	1.98783
	$\mathbf{T}_2$	1.00	1.00	2.51758
	$\mathbf{T}_3$	0.50	1.00	3.76098
DRPBG with 50 F_1	$\mathbf{T}_1$	1.25	1.00	3.35855
	$\mathbf{T}_2$	1.00	1.00	4.24594
	$\mathbf{T}_3$	0.75	1.00	5.87459
MPB sc. 2 with ten cones		—	0.00	4.11117
MPB sc. 2 with 50 cones		0.50	1.00	3.74856
DCBG with F_6	$\mathbf{T}_1$	2.00	1.00	8.41941
(Ackley components)	$\mathbf{T}_2$	1.50	1.00	9.77345
	$\mathbf{T}_3$	1.50	1.00	14.24450
DCBG with F_4	$\mathbf{T}_1$	—	0.00	570.72
(Rastrigin components)	$\mathbf{T}_2$	—	0.00	610.565
	$\mathbf{T}_3$	—	0.00	661.395

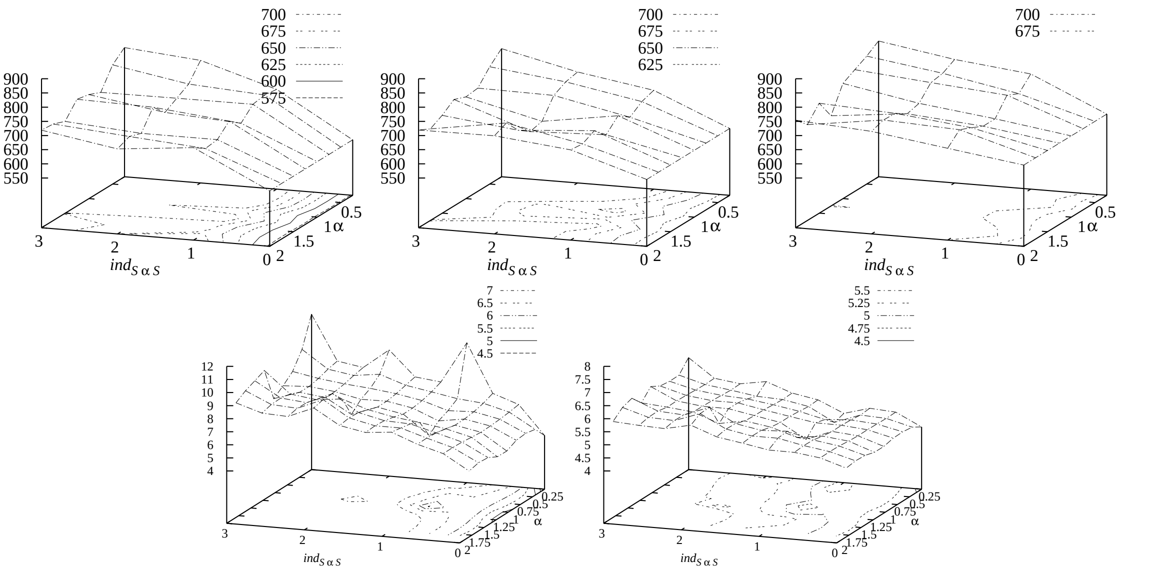

Figure 2: Characteristics of *oe* for jDE: the cases where hybrid solution deteriorates the results, i.e., DCBG with F_4 (Rastrigin components) (the first row) and MPB sc.2 with 10 and 50 F_2 (cones) (the second row); the subsequent graphs in the first row represent three change types of the GDBG control parameters: $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$.

were the most difficult and which were the easiest.

All the results are depicted in Figures 2 and 3. The graphs are divided into two groups: where due to the presence of $\mathbf{s}_{\text{Levý}}$ solutions obtained results deterio-

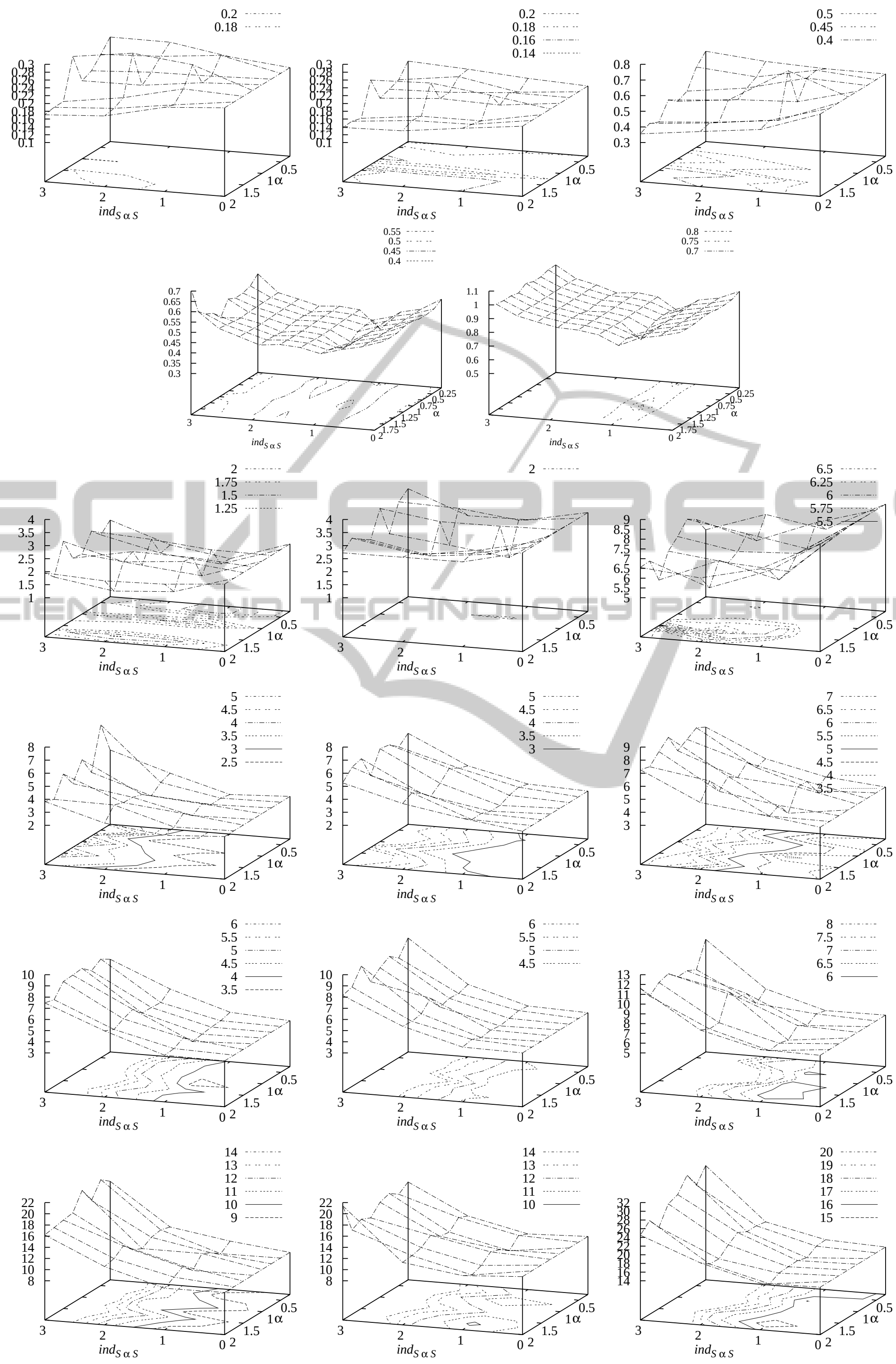

Figure 3: Characteristics of *oe* for jDE: the cases where hybrid solution improves the results: DCBG with F_5 (Griewank components) (the first row), MPB sc.1 with ten and with 50 F_1 (peaks) (the second row) DCBG with F_2 (sphere components) (the third row), DRPBG with ten and with 50 F_1 (peaks) (the fourth and the fifth row), DCBG with F_6 (Ackley components) (the last row); the columns represent three change types of the GDBG control parameters: $\mathbf{T}_1$, $\mathbf{T}_2$ and $\mathbf{T}_3$ (except for MPB where there are just two cases: with ten and 50 peaks).

rated (Figure 2), and where application of $\mathbf{s}_{\text{Lev\'y}}$ solutions in the population improved the results, that is, obtained error decreased (Figure 3).

For each of the benchmark instances there were obtained 32 values of *oe* which are presented in a graphical form as a surface generated for different values of the number of $\mathbf{s}_{\text{Lev\'y}}$ solutions (that is, $ind_{S\alpha S}$) and α. The benchmark instances in the Figures are ordered in the same way as in Table 2.

7 CONCLUSIONS

In this paper we introduce hybrid structure of population in differential evolution algorithm jDE and study its properties when applied to dynamic optimization tasks. This is a case of a kind of technology transfer where promising mechanisms form one heuristic approach appear to be useful in another. The results show that mutation operator using random variates based on α-stable distribution, that is, the operator where both fine local modification and macro-jumps can appear, improves the values of *oe*. Lack of improvement appeared in two cases, that is, for the DCBG with F4 (Rastrigin components) which is a very difficult landscape looking like a hedgehog (Figure 1, the graph in the second row on the left) and for the MPB sc.2 with ten cones. In both cases macromutation most probably introduces unnecessary noise rather than a chance for exploration of a new promising area. Difficulty of the former benchmark instance is confirmed by extremely high values of error obtained for all three types of change.

In the remaining cases the influence of $\mathbf{s}_{\text{Lev\'y}}$ solution presence is positive, however, it must be stressed that the number of these solutions should be small. In most cases the best results were obtained for just one piece. The exception is the DCBG with F5 (Grievank components) which was the easiest landscape for the algorithm, easier even than the landscape build of the spheres. In the case of the DCBG with F5 the higher number of $\mathbf{s}_{\text{Lev\'y}}$, the smaller value of *oe*.

Finally, it is worth to stress that the different complexity of the tested instances shows also that when we take them as a suite of tests and evaluate overall gain of the algorithm we need to apply different weight for the successes obtained for each of the instances. Simply, an improvement of efficiency obtained for DCBG with Grievank components have different significance than the same improvement for, e.g., DCBG with Ackley components.

REFERENCES

Branke, J. (1999). Memory enhanced evolutionary algorithm for changing optimization problems. In *Proc. of the Congr. on Evolutionary Computation*, volume 3, pages 1875–1882. IEEE Press, Piscataway, NJ.

Brest, J., Greiner, S., Boskovic, B., Mernik, M., and Zumer, V. (2006). Self-adapting control parameters in differential evolution: A comparative study on numerical benchmark problems. *IEEE Trans. Evol. Comput.*, 10(6):646–657.

Brest, J., Zamuda, A., Boskovic, B., Maucec, M. S., and Zumer, V. (2009). Dynamic optimization using self-adaptive differential evolution. In *IEEE Congr. on Evolutionary Computation*, pages 415–422. IEEE.

Feokistov, V. (2006). *Differential Evolution, In Search of Solutions*, volume 5 of *Optimization and Its Applications*. Springer.

Li, C. and Yang, S. (2008). A generalized approach to construct benchmark problems for dynamic optimization. In *Simulated Evolution and Learning, SEAL*, volume 5361 of *LNCS*, pages 391–400. Springer.

Liang, J. J., Suganthan, P. N., and Deb, K. (2005). Novel composition test functions for numerical global optimization. In *IEEE Swarm Intelligence Symposium*, pages 68–75, Pasadena, CA, USA.

Price, K. V. (1994). Genetic annealing. *Dr. Dobb's Journal*, pages 127–132.

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

Salomon, R. (1996). Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions. *BioSystems*, 39(3):263–278.

Suganthan, P. N., Hansen, N., Liang, J. J., Deb, K., Chen, Y.-P., Auger, A., and Tiwari, S. (2005). Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization. Technical report, Nanyang Technological University, Singapore.

Trojanowski, K. (2009). Properties of quantum particles in multi-swarms for dynamic optimization. *Fundamenta Informaticae*, 95(2-3):349–380.