Self-Adaptive Differential Evolution with Hybrid Rules of Perturbation for Dynamic Optimization

Krzysztof Trojanowski[a], Mikołaj Raciborski[b], and Piotr Kaczyński[b]

[a] *Institute of Computer Science, Polish Academy of Sciences, Warsaw, Poland*
[b] *Faculty of Mathematics and Natural Sciences, Cardinal Stefan Wyszyński University, Warsaw, Poland*

Abstract—In this paper an adaptive differential evolution approach for dynamic optimization problems is studied. A new benchmark suite `Syringa` is also presented. The suite allows to generate test-cases from a multiple number of dynamic optimization classes. Two dynamic benchmarks: Generalized Dynamic Benchmark Generator (GDBG) and Moving Peaks Benchmark (MPB) have been simulated in `Syringa` and in the presented research they were subject of the experimental research. Two versions of adaptive differential evolution approach, namely the jDE algorithm have been heavily tested: the pure version of jDE and jDE equipped with solutions mutated with a new operator. The operator uses a symmetric α-stable distribution variate for modification of the solution coordinates.

Keywords—adaptive differential evolution, dynamic optimization, symmetric α-stable distribution.

1. Introduction

Uncertainty in the optimized problem demands from the optimization algorithms specific features appropriate to the form of the uncertainty. Particularly, in the presented research our algorithm has to cope with changes in the evaluation function formula or the function parameters appearing during the process of optimum search. Problems with this type of uncertainty are called dynamic problems and searching for solutions of such problems is called dynamic optimization. It is worth noting, that this is just one of the forms of uncertainty appearing in the optimization process. The remaining three according to the classification given in [1] are represented by:

- a noise in the optimized function,
- when for some reason (for example, high computational costs), instead of using the optimized function, we use its approximation evaluation,
- when the main aim is to find a solution not only of the highest quality, but – more importantly – the one whose neighbors are also good, that is, when the most important issue is the robustness of the returned solution.

Among a number of existing metaheuristics we selected a differential evolution (DE) for the research. This approach is recently a subject of growing interests and has already been studied from many points of view (for detailed discussion see, for example, monographs [2] or [3]). Our attention has been paid to the self-adaptive version of the DE algorithm [4] which 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. Eventually, for our research we reimplemented the version of jDE presented in [5]. Our experiments were conducted with this version as well as a version extended by a new mutation operator inspired by a mechanism originating from the particle swarm optimization approach. We also modified the set of benchmark instances and extended the number of the algorithm quality measures which are evaluated during the experiments. This gave a wider view to the nature of the dynamic optimization process performed by jDE and allowed for its better understanding. The main aim of our research was a transfer of the idea of the new mutation operator to the differential evolution approach and experimental verification if such a transfer is justified, that is, allows to improve the algorithm effectiveness.
The paper is organized as follows. In Section 2 a brief description of the optimization algorithm is presented. Section 3 presents properties of new type of mutation introduced to jDE. The new benchmark suite called `Syringa` is presented in Section 4. Section 5 includes some details of the selected test-cases and their configurations as well as the applied performance measures. Section 6 shows the results of experiments. Section 7 concludes the presented research.

2. The Algorithm

The DE algorithm is a kind of evolutionary method with a very specific mutation operator controlled by the scale factor F. Three different, randomly chosen solutions are selected from the current population to mutate a target solution $\mathbf{x}^i$: a base solution $\mathbf{x}^0$ and two difference solutions $\mathbf{x}^1$ and $\mathbf{x}^2$. First, the three solutions are used to create a mutant solution. Then, the mutant undergoes discrete recombination with the target solution $\mathbf{x}^i$. The recombination 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

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
```

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.

Self-Adaptive Differential Evolution approach, namely the jDE algorithm 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 modified every generation [4]:

$$F_i(t+1) = \begin{cases} F_{\min} + rand[0,1) \cdot F_{\max} & \text{if } rand[0,1) < 0.1 \\ F_i(t) & \text{otherwise} \end{cases}$$

$$C_i(t+1) = \begin{cases} rand[0,1) & \text{if } rand[0,1) < 0.1 \\ C_i(t) & \text{otherwise} \end{cases}$$

where: $F_{\min} = 0.36$ and $F_{\max} = 0.9$ represent the lower and upper boundary for F, and $F_i(0) = 0.5$. For the CR factor: $CR_i \in [0,1)$ and $CR_i(0) = 0.9$.

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 (this is the configuration of the algorithm presented in [5]). Each of them has to perform its own search process, that is, no information is shared between the subpopulations. Every solution is a subject to the aging mechanism 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 if too close localization is observed then one of the subpopulations is reinitialized. However, as in the previous case the subpopulation with the global-best is never the one to reinitialize. The last extension is application of a memory structure called archive. The archive is increased after every 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. For details of the above-mentioned extension procedures the reader is referred to [5].

3. Proposed Extension of jDE

The novelty proposed in this paper is an introduction of new type of solutions into the population of size M. The difference between the regular members of the population in the DE algorithm and the new ones lies in the way they are mutated. The regular individuals undergo classic mutation, that is, the mutation typical for DE with rules of perturbation based on the base solution and two difference ones as described above. The new solutions are modified with use of the mutation operator (similar to the one presented in [6]) which is based on the rules of movement governing quantum particles in mQSO [7].

In our approach a small number of new solutions (just one, two, or three pieces) replace the classic ones so the population size remains unchanged. In the subpopulation the new solutions coexist with the classic ones and undergo just the same procedures of selection and suppression.

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

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

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

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

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

where:

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

where: D_{w} is the width of the feasible part of the search space, i.e., the distance between its lower and upper boundaries and $f(\mathbf{x}_j)$ is the value of the j-th solution.

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. 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

– the minimum (for minimization problems),

– the maximum (otherwise) value among the values obtained for the intersected components,

– the sum of the fitness vales obtained from these components.

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 be defined for a part or the whole of the search space, thus, in the case of a solution covered by more than one component the value of this solution can be the minimum, the maximum or the sum of values returned by the covering components.

4.1. The Components

Current version of Syringa consists of six types of component functions (Table 1) defined for the real-valued search space. All formulas include the number of the search apace dimensions n which makes them able to define 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 a 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.

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)	$f(\mathbf{x}) = \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) + 10\right)$	[–5,5]
Griewank (F_5)	$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n}(x_i)^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$	[–100,100]
Ackley (F_6)	$f(\mathbf{x}) = -20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - \exp\left(\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)\right) + 20 + e$	[–32,32]

4.2. Horizontal Change Parameters

In this case the coordinates of the solution $\mathbf{x}$ (a vector, that is, 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:

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

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.

4.3. Vertical Change Parameters

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

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

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. (7)), large step change ($\mathbf{T}_2$ – Eq. (8)), and two versions of random changes ($\mathbf{T}_3$ – Eq. (9) and $\mathbf{T}_4$ – Eq. (10)). The change Δ of a parameter value is calculated as follows:

$$\Delta = \alpha r(\max - \min), \tag{6}$$

$$\text{where } \alpha = 0.04,\ r = U(0,1), \tag{7}$$

$$\Delta = (\alpha \cdot sign(r_1) + (\alpha_{\max} - \alpha) r_2) \cdot (\max - \min),$$

$$\text{where } \alpha = 0.04,\ r_{1,2} = U(0,1),\ \alpha_{\max} = 0.1 \tag{8}$$

$$\Delta = N(0,1), \tag{9}$$

$$\Delta = U(r_{\min}, r_{\max}) \tag{10}$$

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 standardized normal distribution, $U(a,b)$ is a random value obtained with uniform distribution form the range $[a,b]$, 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 (that is, 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) [8] and the Moving Peaks Benchmark generator [9]. 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 components and build agents which will control the components in the same manner like in the simulated benchmark.

4.5. Reimplementation of Moving Peaks Benchmark (MPB)

In the case of MPB, three scenarios of the benchmark parameters control are defined [9]. We performed experiments for two versions of just the second scenario. In this scenario the fitness landscape has been defined for the five-dimensional search space with the same boundaries for each dimension, namely $[-50; 50]$.
The selected fitness landscape consists of a set of cones (F_2) which undergo two types of horizontal changes: the trans-

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

$$f(\mathbf{x}) = h - w \cdot \sqrt{\sum_{j=1}^{n} (\mathbf{x}_j - \mathbf{X}_j)^2}\,. \qquad (11)$$

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 feasible interval $[f_{\min}; f_{\max}]$. Simply, for every modified parameter of translation or scaling, which can be represented as a symbol p, $r_{\min}$ equals $p_{\min} - p$ and $r_{\max}$ equals $p_{\max} - p$. 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 norm of the translation vector: $|\mathbf{X}| \leq 3$. In the first version of the scenario 2 there are ten moving cones whereas in the second version 50 moving cones are in use. Besides, we extended the number of the search space dimensions and performed our tests also for $n = 10, 20, 30$ for both versions of the selected benchmark instance.

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

$$f(\mathbf{x}) = \frac{v}{1 + w \sum_{j=1}^{n} \frac{(\mathbf{x}_j - \mathbf{X}_j)^2}{n}}\,. \qquad (12)$$

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 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 multidimensional search space. A matrix $R_{ij}(\theta)$ represents rotation by the θ angle along the plane i–j and such a matrix can be easily generated as described by [10]. 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 search space dimension indices selected randomly without repetition. Then for every plane defined in $\mathbf{r}$ by subsequent pairs of indices: $[1,2]$, $[3,4]$, $[5,6]$, $\ldots$ $[l-1,l]$ a rotation angle is randomly generated and finally respective matrices $R_{r[1],r[2]}, \ldots R_{r[l-1],r[l]}$ are calculated. Eventually, the rotation matrix M is calculated as follows:

$$M(t) = R_{r[1],r[2]}(\theta(t)) R_{r[3],r[4]}(\theta(t)) \cdots R_{r[l-1],r[l]}(\theta(t)).$$

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]. The next values in Θ represent planes created from every second dimensions, that is, [1,3], [2,4] and so on until the plane [n-2,n]. Then values in Θ represent planes created from every third dimensions, then those created from every fourth, and so on until there appears the value for the last plane created from the first and the last dimension. If $\Theta(i)$ equals zero, then 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 M is the same as in the description above, that is, the rotation matrix M is the result of multiplication of all the matrices R generated from the vector Θ.

The matrix $\mathbf{M}$ is 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 Eq. (12).

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 [11], [12]:

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

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, that is, $\mathbf{W}$ 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 $f''(\mathbf{x})$ looks as follows:

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

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.

5. Plan of Experiments

There were performed two groups of experiments. In the first one we applied the pure version of jDE, whereas in the second one the version of jDE extended by $\mathbf{s}_{\text{Levý}}$ solutions was in use. The main aim of the first group was to study the effectiveness of the algorithm for testing environments with different complexity of the search space. The second group of experiments showed the influence of $\mathbf{s}_{\text{Levý}}$ solutions on the search process and the quality of the obtained results.

Experiments form the first group were performed with a subset of GDBG benchmark instances as well as with two versions of MPB scenario 2, for different numbers of the search space dimensions. The tests based on MPB were repeated twice. First, the tests were performed with the number of fitness function calls between subsequent changes defined as it was proposed for the CEC'09 competition, that is, for $10^4 \cdot n$ fitness function calls (n is a number of search space dimensions). Then the tests were repeated – for a fixed number of 5000 calls as it is recommended for experiments with MPB [9]. The tests based on GDBG were performed once just for $10^4 \cdot n$ fitness function calls between subsequent changes.

The second group includes experiments with a subset of GDBG benchmark functions as well as with four versions of MPB and for different numbers of $\mathbf{s}_{\text{Levý}}$ solutions present in the population. All the experiments in this group were performed for $10^4 \cdot n$ fitness function calls between subsequent changes. However, it must be stressed that in this group the number of search space dimensions was constant and equal five for every test-case.

To decrease the number of algorithm configurations which would be experimentally verified in the second group of experiments we decided 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 parameter configurations the experiments were repeated 20 times and each of them consisted of 60 changes in the fitness landscape.

5.1. The Measures

We used measures of the obtained results proposed for both of the benchmarks by their authors. This gave opportunity for fair comparison of the algorithm efficiency. For GDBG there were defined four measures:

$$\begin{aligned}
Avg^{\text{best}} &= \sum_{i=1}^{N_{\text{exp}}} \min_{j=1,\ldots,N_{\text{ch}}} E^{i,j}_{\text{last}}/N_{\text{exp}},\\
Avg^{\text{mean}} &= \sum_{i=1}^{N_{\text{exp}}} \sum_{j=1}^{N_{\text{ch}}} E^{i,j}_{\text{last}}/(N_{\text{exp}} \cdot N_{\text{ch}}),\\
Avg^{\text{worst}} &= \sum_{i=1}^{N_{\text{exp}}} \max_{j=1,\ldots,N_{\text{ch}}} E^{i,j}_{\text{last}}/N_{\text{exp}},\\
STD &= \left(\frac{1}{N_{\text{exp}} \cdot N_{\text{ch}} - 1} \sum_{i=1}^{N_{\text{exp}}} \sum_{j=1}^{N_{\text{ch}}} (E^{i,j}_{\text{last}} - Avg^{\text{mean}})^2\right)^{-1},
\end{aligned}$$

where: N_{exp} is a number of repeated experiments for the same control parameter settings of the algorithm, N_{ch} is a number of changes in the fitness landscape appearing during a single experiment run, and E^{j}_{last} is an absolute function error value:

$$E^{j}_{\text{last}} = |f(\mathbf{x}^{j}_{\text{best}}) - f(\mathbf{x}^{*j})|,$$

where: $\mathbf{x}^{j}_{\text{best}}$ – current best solution which has been found since the last j-th change in the fitness landscape, $\mathbf{x}^{*j}$ – real optimum solution for the fitness landscape after the j-th change.

In the case of MPB most of the publications contain the values of offline error measure (oe) obtained for performed experimental research. The offline error represents the average deviation from the optimum of the best solution value since the last change in the fitness landscape. Formally:

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

where: $N_{\mathrm{e}}(j)$ is a total number of solution evaluations performed for the j-th static state of the landscape. The measure oe should be minimized, that is, the better result the smaller the value of oe.

6. The Results

6.1. The First Group of Experiments – with the Pure Version of jDE

Majority of the results from the first group of experiments have already been reported in [13]. One of the more important observations was about a sensitivity of the Avg^{mean} measure to the complexity of the search space. Graphs in Fig. 1 depicts values of Avg^{mean} obtained for the tests which were performed for $10^4 \cdot n$ fitness function calls between subsequent changes, that is, for the number of fitness function calls depending on the number of the search space dimensions. Three types of Avg^{mean} error value trends can be observed in Fig. 1. They show that the proposed linear dependency of the evaluation number on n:

- does not compensate growth of the errors (e.g., DRPBG problems with F_4 components (Rastrigin functions)),
- does compensate growth of the errors for some limited range (e.g., DRPBG problems with F_6 components (Ackley functions) or with F_3 components (sphere functions)),
- does compensate growth of the errors excessively (e.g, DRPBG problems with F_5 components (Griewank functions)).

The above-mentioned difference in the sensitivity of Avg^{mean} and the offline error inspired the second group of experiments. We wanted to compare the results obtained with these four measures for the jDE version extended by $\mathbf{s}_{\mathrm{Lev\acute{y}}}$ solutions optimizing in five-dimensional search space.

6.2. The Second Group of Experiments – with jDE Extended by $s_{\mathrm{Lev\acute{y}}}$ Solutions

In this group of experiments we observed values of the above-mentioned measures for the experiments with different number of $\mathbf{s}_{\mathrm{Lev\acute{y}}}$ solutions present in the algorithm. We did full range of experiments for different values of α and different numbers of $\mathbf{s}_{\mathrm{Lev\acute{y}}}$ solutions in subpopulations. Then we selected the best results obtained with the offline error measure oe and with Avg^{mean}. The results are gathered in two tables: Table 2 with the results for the cases where the best values of Avg^{mean} and oe were obtained for the same configurations of jDE parameters and Table 3 where the best values of Avg^{mean} and oe were obtained for different configurations. In the latter case the best values are printed in bold characters.

Tables 2 and 3 present list of all the benchmark instances and configurations of jDE where the best values of Avg^{mean} and oe were obtained for the instances as well as the values of all the performance measures. In most cases the best values of Avg^{mean} were obtained for jDE without extension (except for DCBG with F_5 and with all three types of characteristics of the parameters variability and three another cases: DRPBG $10 \times F_1$, $\mathbf{T}_1$, DRPBG $50 \times F_1$, $\mathbf{T}_3$ and MPB sc.2 with 50 cones) whereas there were many test-cases where the best values of oe were obtained for jDE extended with $\mathbf{s}_{\mathrm{Lev\acute{y}}}$ solutions (DCBG with F_3 or F_6 and with all three types of characteristics of the parameters variability, majority of DRPBG configurations and again MPB sc.2 with 50 cones). In the latter group of jDE configurations the best values of oe can be observed for jDE with just one $\mathbf{s}_{\mathrm{Lev\acute{y}}}$ solution existing in every subpopulation, however, there is an exception which is DCBG with F_5 where the best values were obtained for the highest numbers of $\mathbf{s}_{\mathrm{Lev\acute{y}}}$ solutions.

7. Conclusions

In this paper we studied properties of jDE applied to the number of dynamic optimization tasks. A new benchmark suite called `Syringa` was implemented which is able to simulate existing benchmarks and allows to create plenty of new classes of dynamic optimization problems as well. Two existing benchmarks were selected for experiments: MPB and DCBG. Selected subsets of test-case instances originated from the two benchmarks were reimplemented within the suite. Then, a number of tests was executed for these optimization tasks. As an optimization tool the self-adaptive differential evolution approach was selected.

7.1. Conclusions from the First Group of Experiments

In the first group we used a number of test-case instances of different complexity, that is, test-cases with the number of search space dimensions growing from five to 30. To compensate growth of errors accompanying growing complexity of the optimized problems the tests were performed with the number of fitness function calls defined as it was proposed for the CEC'09 competition. Obtained different error value trends for the applied measures revealed the search process features varying for the problems of growing complexity. In spite of the fact that in every case the value of oe increased as the number of search space dimensions grew [13] obtained mean values of the best found solutions have different characteristics. For some test-cases the compensation formula was sufficient to keep the values of the best found solution on the same level whereas for the others

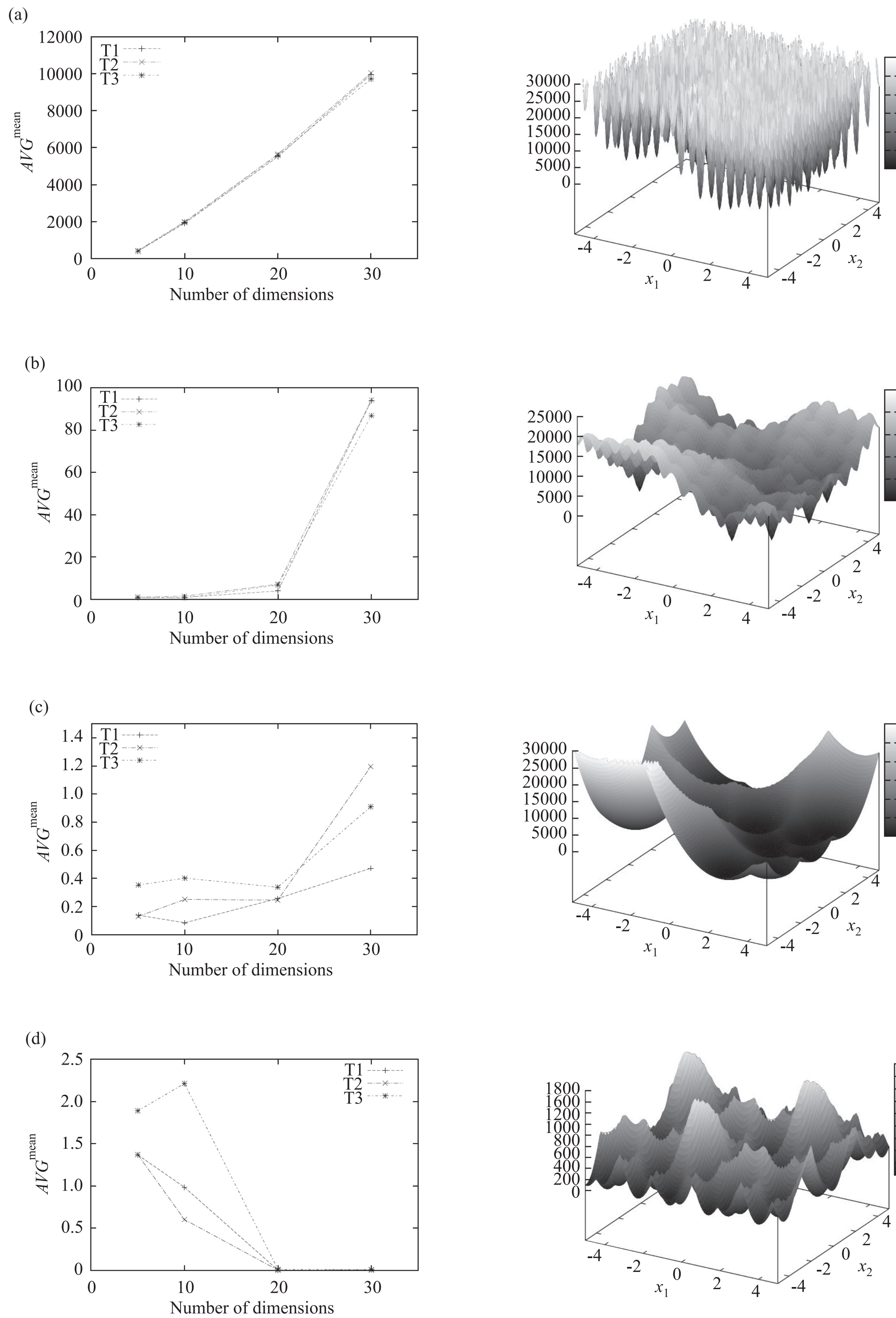


Fig. 1. Avg^{mean} values (on the left) obtained for DCBG test-cases (respective fitness landscapes in 2D are on the right) with: (a) F_4 components (Rastrigin functions), (b) F_6 components (Ackley functions), (c) F_3 components (sphere functions), and (d) F_5 components (Griewank functions).

Table 2
The results for the cases where the best values of Avg^{mean} and oe were obtained for the same configurations of jDE parameters

Testing environment	α	$\mathbf{s}_{\text{Levý}}$ num.	Avg^{best}	Avg^{mean}	Avg^{worst}	STD	oe	Std. err
MPB sc.2, 10 cones	—	0	0	**2.929**	14.241	4.387	**4.111**	2.213
MPB sc.2, 50 cones	0.5	1	$8.39 \cdot 10^{-6}$	**2.730**	11.129	2.941	**3.749**	1.013
DCBG with F_4, $\mathbf{T}_1$	—	0	2.999	**368.31**	829.59	187.81	**570.72**	48.496
DCBG with F_4, $\mathbf{T}_2$	—	0	5.725	**414.39**	824.64	191.88	**610.56**	57.612
DCBG with F_4, $\mathbf{T}_3$	—	0	0.746	**402.62**	805.08	187.34	**661.39**	54.57

Table 3
The results for the cases where best values of Avg^{mean} and oe were obtained for different configurations of jDE parameters

Testing environment	α	$\mathbf{s}_{\text{Levý}}$ num.	Avg^{best}	Avg^{mean}	Avg^{worst}	STD	oe	Std. err
DRPBG $10 \times F_1$, $\mathbf{T}_1$	2	1	0	**0.004**	0.222	0.130	2.041	0.354
DRPBG $10 \times F_1$, $\mathbf{T}_1$	0.25	1	0	0.0208	0.878	0.312	**1.988**	0.688
DRPBG $10 \times F_1$, $\mathbf{T}_2$	—	0	0	**0.0023**	0.074	0.056	2.675	0.310
DRPBG $10 \times F_1$, $\mathbf{T}_2$	1	1	0	0.0361	1.189	0.353	**2.518**	0.689
DRPBG $10 \times F_1$, $\mathbf{T}_3$	—	0	0	**0.004**	0.264	0.141	3.985	0.678
DRPBG $10 \times F_1$, $\mathbf{T}_3$	0.5	1	0	0.058	1.644	0.728	**3.761**	1.012
DRPBG $50 \times F_1$, $\mathbf{T}_1$	—	0	0	**0.352**	4.899	1.190	3.601	0.641
DRPBG $50 \times F_1$, $\mathbf{T}_1$	1	1	0	0.653	6.976	1.471	**3.501**	0.523
DRPBG $50 \times F_1$, $\mathbf{T}_2$	—	0	0	**0.385**	5.816	1.188	4.318	0.529
DRPBG $50 \times F_1$, $\mathbf{T}_2$	1.75	1	0	0.703	6.532	1.633	**4.219**	0.659
DRPBG $50 \times F_1$, $\mathbf{T}_3$	0.75	1	0	**0.619**	7.006	2.134	5.875	1.860
DRPBG $50 \times F_1$, $\mathbf{T}_3$	1.5	1	0	0.836	6.912	1.853	**5.657**	1.214
DCBG with F_3, $\mathbf{T}_1$	—	0	0	**0.138**	3.715	0.822	2.087	0.536
DCBG with F_3, $\mathbf{T}_1$	1.5	2	$2.15 \cdot 10^{-6}$	1.249	9.288	3.032	**1.226**	0.487
DCBG with F_3, $\mathbf{T}_2$	—	0	0	**0.295**	7.664	1.673	3.290	0.860
DCBG with F_3, $\mathbf{T}_2$	0.75	1	0	0.452	7.679	1.978	**1.816**	0.506
DCBG with F_3, $\mathbf{T}_3$	—	0	0	**0.240**	4.692	0.932	8.482	1.827
DCBG with F_3, $\mathbf{T}_3$	1.75	3	0.0007	1.989	10.103	2.819	**5.325**	1.923
DCBG with F_5, $\mathbf{T}_1$	2	2	0.133	**2.113**	6.118	1.483	0.180	0.033
DCBG with F_5, $\mathbf{T}_1$	1.25	3	0.464	2.253	6.163	1.362	**0.168**	0.028
DCBG with F_5, $\mathbf{T}_2$	2	2	0.128	**1.847**	5.734	1.359	0.143	0.015
DCBG with F_5, $\mathbf{T}_2$	1.75	3	0.323	2.148	5.905	1.319	**0.133**	0.019
DCBG with F_5, $\mathbf{T}_3$	1.75	3	0.411	**2.364**	6.276	1.381	0.358	0.070
DCBG with F_5, $\mathbf{T}_3$	2	3	0.288	2.491	5.891	1.340	**0.356**	0.044
DCBG with F_6, $\mathbf{T}_1$	—	0	0	**0.545**	13.169	2.698	8.539	1.847
DCBG with F_6, $\mathbf{T}_1$	2	1	0.0004	1.031	11.291	2.790	**8.419**	2.811
DCBG with F_6, $\mathbf{T}_2$	—	0	0	**0.547**	11.992	2.238	11.381	2.077
DCBG with F_6, $\mathbf{T}_2$	1.5	1	0.0003	1.309	15.526	3.651	**9.773**	2.594
DCBG with F_6, $\mathbf{T}_3$	—	0	0	**0.590**	6.859	1.643	15.914	1.086
DCBG with F_6, $\mathbf{T}_3$	1.5	1	0.0005	1.420	8.799	2.922	**14.245**	2.167

it was not (Fig. 1). This allowed to identify some of the test cases as more difficult that the others.

The observations confirm that a new formula for the number of fitness evaluations applied to compensate growth of the errors is needed (probably different ones for different landscapes). However, different trends obtained for the two measures show also that the measures evaluate different features of the search process and thus probably it will be hard of even impossible to find the rule which would be able to compensate growth of both of the errors at the same time.

7.2. Conclusions from the Second Group of Experiments

In the second group of tests we studied influence of the new type of mutation operator on the search process effectiveness. A new type of solutions which undergo the new mutation procedure was introduced into the population. The presence of new solutions changed the population behavior in the way which allowed to explore faster new promising areas of the search space. However, from the other side the enhancement of exploring properties was the reason for less precise approaching of the population to the optimum. The effect of this can be observed in the results presented in Tables 2 and 3.

7.3. Summary

Obtained results show that application of both *oe* and Avg^{mean} in the experimental research analysis returns more information about the search process and shows more pro and cons about the tested algorithm than the application of just one of them. Our experiments and especially obtained values of *oe* showed that application of $\mathbf{s}_{\text{Levý}}$ solutions allows to explore and approach faster the promising areas of the search space in many benchmark instances. However, worse values of Avg^{mean} accompanying the best values of *oe* are the argument against this if we are interested in the highest values of the best found solutions. So, in unpredictable circumstances and especially when the available number of time, that is, the available number of fitness function evaluations is hardly predictable and it is probable that the estimated number can be unexpectedly shortened the better option is the maximization of *oe*. In the opposite case, that is, when the granted number of fitness function evaluations is sufficient and guaranteed, the strategy of Avg^{mean} maximization is much more reasonable.

Acknowledgments

This research has been partially supported by the European Regional Development Fund with the grant no. POIG.01.01.02-14-013/09: *Adaptive system supporting problem solution based on analysis of textual contents of available electronic resources.*

References

[1] Y. Jin and J. Branke, "Evolutionary algorithms in uncertain environments – a survey", *IEEE Trans. Evol. Comput.*, vol. 9, no. 3, pp. 303–317, 2005.

[2] V. Feokistov, "Differential evolution", in *Search of Solutions*, vol. 5 of *Optimization and Its Applications*. Springer, 2006.

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

[4] J. Brest, S. Greiner, B. Boskovic, M. Mernik, and V. Zumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems", *IEEE Trans. Evol. Comput.*, vol. 10, no. 6, pp. 646–657, 2006.

[5] J. Brest, A. Zamuda, B. Boskovic, M. S. Maucec, and V. Zumer, "Dynamic optimization using self-adaptive differential evolution", in *IEEE Congr. Evol. Comput.*, pp. 415–422. IEEE, 2009.

[6] A. Obuchowicz and P. Pretki, "Isotropic symmetric α-stable mutations for evolutionary algorithms", in *Proc. Congr. Evol. Comput.*, vol. 1, pp. 404–410. IEEE Press, 2005.

[7] K. Trojanowski, "Properties of quantum particles in multi-swarms for dynamic optimization", *Fundamenta Informaticae*, vol. 95, no. 2–3, pp. 349–380, 2009.

[8] C. Li and S. Yang, "A generalized approach to construct benchmark problems for dynamic optimization", in *Simulated Evolution and Learning, 7th Int. Conf. SEAL 2008*, Melbourne, Australia, 2008, vol. 5361 of *LNCS*, pp. 391–400. Springer, 2008.

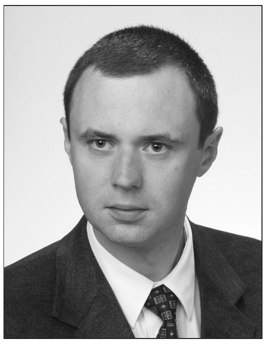
[9] J. Branke, "Memory enhanced evolutionary algorithm for changing optimization problems", in *Proc. Congr. on Evol. Comput.*, vol. 3, pages 1875–1882. IEEE Press, Piscataway, NJ, 1999.

[10] R. Salomon, "Reevaluating genetic algorithm performance under coordinate rotation of benchmark functions", *BioSystems*, vol. 39, no. 3, pp. 263–278, 1996.

[11] J. J. Liang, P. N. Suganthan, and K. Deb, "Novel composition test functions for numerical global optimization", in *Proc. IEEE Swarm Intelligence Symp.*, Pasadena, CA, USA, 2005, pp. 68–75.

[12] P. N. Suganthan *et al.*, "Problem definitions and evaluation criteria for the cec 2005 special session on real-parameter optimization", Tech. Rep., Nanyang Technological University, Singapore, 2005.

[13] M. Raciborski, K. Trojanowski, and P. Kaczyński, "Differential evolution for high scale dynamic optimization", in *Security and Intelligent Information Systems*, LNCS. Springer, 2011 (will appear in vol. 7053).



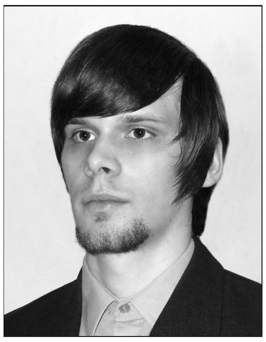
Krzysztof Trojanowski received the M.Sc. degree in Computer Science from the Warsaw University of Technology, Poland, in 1994, the Ph.D. degree from the Institute of Computer Science of the Polish Academy of Sciences (ICS PAS), Warsaw, Poland, in 2000, and the habilitation degree – also from ICS PAS in 2010. Since 1994 he has been with the Institute of Computer Science PAS. From 1995 to 2002 we was a consultant at IT companies in Poland. From 2002 to 2008 he was with University of Podlasie, Siedlce. Currently, he is with the Cardinal Stefan Wyszyński University, Warsaw, where he is an Associate Professor. From 2003 until 2008 he was a chairman of the Organizing Committee of the annual international conference Intelligent Information

Systems organized by ICS PAS. From 2006 until 2008 he was also a chairman of the International Conference on Artificial Intelligence, the oldest AI conference in Poland. His research interests include heuristic optimization techniques, such as evolutionary algorithms, immune systems, and particle swarm optimization.

E-mail: trojanow@ipipan.waw.pl
Institute of Computer Science
Polish Academy of Sciences
Ordona 21 01-237 Warsaw, Poland

Piotr Kaczyński received the M.Sc. degree in Computer Science from the Warsaw University of Technology, Poland and the M.Sc. degree in Theoretical Mathematics from Cardinal Stefan Wyszyński University in Warsaw, Poland, both in 2006. Since then he has been with the Cardinal Stefan Wyszyński University as a teaching assistant and also with various IT companies in Poland working with mathematical modeling and simulation. His research interests include both artificial intelligence applications (such as neural networks, evolutionary algorithms) and stochastic control.

E-mail: p.kaczynski@uksw.edu.pl
Faculty of Mathematics and Natural Sciences
College of Sciences
Cardinal Stefan Wyszyński University
Wóycickiego st 1/3
01-938 Warsaw, Poland



Mikołaj Raciborski received the B.Sc. degree in Natural Sciences, with a specialization of Computer Science in Applications in 2009 and the M.Sc. degree in Computer Science in 2011 both at Cardinal Stefan Wyszyński University, Faculty of Mathematics and Natural Sciences, College of Sciences. His research interests include computer science, mathematics, physics, chemistry, and electronic services.

E-mail: mikolaj.raciborski@gmail.com
Faculty of Mathematics and Natural Sciences
College of Sciences
Cardinal Stefan Wyszyński University
Wóycickiego st 1/3
01-938 Warsaw, Poland