Baltic J. Modern Computing, Vol. 2 (2014), No. 4, pp. 199–214

Empirical Study of Particle Swarm Optimization Mutation Operators

Vytautas Jančauskas

Institute of Mathematics and Informatics
Vilnius University
Akademijos g. 4
LT-08663 Vilnius, Lithuania

vytautas.jancauskas@mif.vu.lt

Abstract. Particle Swarm Optimization (PSO) is a global optimization algorithm for real valued problems. One of the known positive traits of the algorithm is fast convergence. While this is considered a good thing because it means the solutions are found faster it can lead to stagnation at a local minimum. There are several strategies to circumvent this. One of them is the use of mutation operators to increase diversity in the swarm. Mutation operators, sometimes called turbulence or craziness operators, apply changes to solutions outside the scope of the PSO update rules. Several different such operators are proposed in the literature mostly based on existing approaches in evolutionary algorithms. However, it is impossible to say which mutation operator to use in which case and why. There is also some controversy whether mutation is necessary at all. We use an algorithm that generates test functions with given properties - number of local minima, dimensionality, global minimizer attraction region placement and attempt to explore the relationship between function properties and mutation operator choice. An empirical study of the operators proposed in literature is given with extensive experimental results. Recommendations for choosing appropriate mutation operators are proposed.

1 Introduction

In this work we examine the use of mutation operators in Particle Swarm Optimization (PSO) algorithms. PSO is an efficient global optimization algorithm in terms of convergence speed, however it can get stuck in local minima easily. To remedy this several solutions were proposed in the literature. One of them is the use of mutation operators as found in evolutionary algorithms. The use of mutation operators was researched by Andrews (2006), Higashi et al. (2003), Stacey et al. (2003), Esquivel et al. (2003) and Ratnaweera et al. (2004). All of those studies use a small set of benchmark problems to test relative performance of PSO without mutation and PSO with various mutation operators applied. Also they use average value of the best solution over several runs to measure the performance of algorithms in question. We feel that these are shortcommings that should be addressed. First of all any set of benchmark problems is arbitrary. The properties of commonly used test problems are usually fixed, except for the dimensionality. This can be a problem since it is not clear what properties of the problem affect optimizer performance. We try to fix this by using an algorithm that generates test problems with given properties. We use these generated test problems to measure performance of PSO with various mutation operators. Also instead of using averaged optimized function values to measure performance we used percentage of successful runs. A run is deemed successful if during it the algorithm manages to locate global minimum within specified precision. We feel that this better reflects the rationale behind using mutation operators — namely to reduce the tendency of the algorithm to get stuck in local minima.

In section 2 we give an introduction to basic concepts of PSO algorithms, a popular variant is given as used in the experiments in this work. In section 3 we describe PSO mutation operators and rationale behind their use. In section 4 the theory behind test functions and an algorithm for generating them is given. In section 5 we describe how the experiments performed in this work were prepared. In section 6 we present the results and analysis of said results in light of work done by others. In section 7 we give concluding remarks and present practical advice.

2 Particle Swarm Optimization

Particle Swarm Optimization is a global optimization metaheuristic designed for continuous problems. It was first proposed by Kennedy (1995) and Eberhart (1995). Eventually it was expanded in to a book by Kennedy et al. (2001). The idea is to have a swarm of particles (points in multi-dimensional space) each having some other particles as neighbours and exchanging information to find optimal solutions. Particles move in solution space of some function by adjusting their velocities to move towards the best solutions they found so far and towards the best solutions found by their neighbours. These two attractors are further randomly weighted to allow more diversity in the search process. The idea behind this algorithm are observations from societies acting in nature. For example one can imagine a flock of birds looking for food by flying towards other birds who are signaling a potential food source as well as by remembering where this particular bird itself has seen food before and scouting areas nearby. It can also be viewed as modeling the way we ourselves solve problems - by imitating people we know, who we see as particularly successful, but also by learning on our own. Thus problem solving is being guided by our own experience and by the experiences of people we know to have solved similar problems particularly well. The original algorithm is not presented here since it is very rarely used today and we go straight to more modern implementations.

The problem of global optimization is that of finding the global minimum of a real valued function. For a function $f : X \to \mathbb{R}$ the global minimum x^* is such that $f(x^*) \leq f(x)$ for all $x \in X$. Here $X \subseteq \mathbb{R}^d$ and d is the dimensionality of the problem. Set X is also refered to as the search space of the function. The method described here is not able to find the actual global minimum but in practice it is sufficient to get close to it within given precision.

Proposed by Clerc et al. (2002) it is a variant of the original PSO algorithm. It guarantees convergence through the use of the constricting factor χ. It also has the advantage of not having any parameters, except for ϕ_1 and ϕ_2 which represent the influence of the personal best solution and the best solution of particles neighbours on the trajectory of that particle. Both of these parameters are usually set to 2.05 as per suggestion in the original paper. Moving the particle in solution space is done by adding the velocity vector to the old position vector as given in (1) equation.

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i \tag{1}$$

Updating velocity involves taking current velocity and adjusting it so that it will point the particle more in the direction of its personal best and the best of its most successful neighbour. It is laid out in (2) formula.

$$\boldsymbol{v}_i \leftarrow \chi\left(\boldsymbol{v}_i + \boldsymbol{\rho_1} \otimes (\boldsymbol{p}_i - \boldsymbol{x}_i) + \boldsymbol{\rho_2} \otimes (\boldsymbol{g}_i - \boldsymbol{x}_i)\right) \tag{2}$$

where

$$\boldsymbol{\rho}_k = \boldsymbol{U}(0, \phi_k), \quad k \in \{1, 2\} \tag{3}$$

$$\chi = \frac{2}{\phi - 2 + \sqrt{\phi^2 - 4\phi}} \tag{4}$$

and where $\phi = \phi_1 + \phi_2$ with ϕ_1 and ϕ_2 set to 2.05, $\boldsymbol{U}(a, b)$ is a vector of random numbers from the uniform distribution ranging from a to b in value, it has the same dimensionality as particle positions and velocities. Here $\boldsymbol{p}_i$ is the best personal solution of particle i and $\boldsymbol{g_i}$ is the solution found by a neighbour of particle i. Operator $\otimes$ stands for element wise vector multiplication. Vectors $\boldsymbol{\rho_k}$ have the same dimensionality as particle positions and velocities. Which particle is a neighbour of which other particle is set in advance.

The canonical variant of the PSO algorithm is given in Algorithm 1 and can be explained in plain words as follows: for each particle with index j from n particles in the swarm, initialize the position vector $\boldsymbol{x}_j$ to random values from the range specific to function f and initialize the velocity vector to the zero vector, for k iterations (set by the user) update the position vector according to formula (1) and update velocity according to formula (2), recording best positions found for each particle.

Particle swarm topology is a graph that defines neighbourhood relationships between particles. In such a graph particles are represented as vertices and if two particles share an edge they are called neighbours. Only neighbours directly exchange information about their best solutions. As such swarm topology has a profound impact on particle swarm performance as has been shown in studies by Kennedy (1999), Kennedy et al. (2002) or many others. There are many popular swarm topologies, for example a fully connected graph or a graph where particles are connected in to a ring. We used a topology where particles are connected in a von Neumann neighbourhood (a two dimensional rectangular lattice), since it was shown to give results better than other popular simple topologies.

Algorithm 1 Canonical PSO algorithm.

```
1:  for $j \leftarrow 1$ to $n$ do
2:      $\boldsymbol{x}_j \leftarrow \boldsymbol{U}(a, b)$
3:      $\boldsymbol{v}_j \leftarrow \boldsymbol{0}$
4:  end for
5:  for $i \leftarrow 1$ to $k$ do
6:      for $j \leftarrow 1$ to $n$ do
7:          $\boldsymbol{x}_j \leftarrow \boldsymbol{x}_j + \boldsymbol{v}_j$
8:          if $f(\boldsymbol{x}_j) < f(\boldsymbol{p}_j)$ then
9:              $\boldsymbol{p}_j \leftarrow \boldsymbol{x}_j$
10:         end if
11:         Update $\boldsymbol{v}_j$ according to (2) formula
12:     end for
13: end for
```

3 PSO Mutation Operators

It is generally accepted that PSO converges very fast. For example in a study done by Vesterstrom et al. (2004), where they compare the performance of Differential Evolution, Particle Swarm Optimization and Evolutionary Algorithms they conclude that PSO always converges the fastest of the examined algorithms. In practice this is a double-edged sword - fast convergence is obviously attractive in an optimization algorithm, however it is feared that it can lead the algorithm to stagnate after finding a local minimum. There are several strategies to slow down convergence and thus increase the amount of time that the algorithm spends in the initial exploratory stage as opposed to local search indicative of later stages of PSO operation. One solution is to use different swarm topologies since it was shown that using a different topology can affect the swarm operation in terms of convergence speed and allow to adjust the trade-off between exploration and exploitation. See for example a paper by Kennedy (1999) or Kennedy et al. (2002). Another attempt at a solution is to change the velocity update formula to use an inertia coefficient w that the speed it multiplied by during each iteration, see for example work by Eberhart et al. (2000) or by Shi et al. (1998).

The third approach is through the introduction of the mutation operator. A mutation operator is used to modify particle positions or velocities outside the position and velocity update rules. In all of the cases examined here mutation is applied after position and velocity updates and only to particle positions. Each coordinate of each particle has a certain probability of being mutated. The probability can be calculated from Equation (5) if mutation $rate$ is provided. Parameter $rate$ means how many particle dimensions will be mutated during each algorithm iteration. For example if $rate = 1$ one dimension of one particle in the swarm will be mutated on average during each iteration.

$$probability = \frac{rate}{particles \times dimensions} \tag{5}$$

We examine five different mutation operators that are found in literature. The first one given in Equation (6) simply reinitializes a single dimension of a particle to a uniformly distributed random value $U_{(a_d, b_d)}$ from the permissible range. It is used to test whether it is useful to rely on the previous value x_{id} or not, it is the only of the operators that does not rely on it. It can be found, for example, in an overview of mutation operators by Andrews (2006). Here a_d and b_d are lower and upper bounds for dimension d in the search space.

$$x_{id} \leftarrow U_{(a_d,b_d)} \tag{6}$$

Another operator, also proposed by Andrews (2006) is based on the Gaussian distribution and given in Equation (7).

$$x_{id} \leftarrow x_{id} + N(\sigma, 0) \tag{7}$$

Another operator based on the Gaussian distribution is given in Equation (8) and can be found in a work by Higashi et al. (2003).

$$x_{id} \leftarrow x_{id}(1 + N(\sigma, 0)) \tag{8}$$

A similar operator to the one given in Equation (7) is given in Equation (9), the only difference is that this one is based on the Cauchy distribution. It is presented in a work by Stacey et al. (2003). In the case of Cauchy distribution p.d.f. of the distribution is given by $f(x) = \frac{a}{\pi}\frac{1}{x^2+a^2}$ and $a = 0.2$.

$$x_{id} \leftarrow x_{id} + cauchy(a) \tag{9}$$

A different kind of mutation operator proposed by Michalewitz (1996). It was proposed for use in PSO by Esquivel et al. (2003). While the original operator changes it behaviour with regards to algorithm iteration we used a static version to keep it in line with the other operators. It is given in Equation (10), where $flip$ is a random value in the range $(0, 1)$, generated before applying the operator.

$$x_{id} \leftarrow \begin{cases} x_{id} + (b_d - x_{id})U_{(0,1)} & \text{, if } flip < 0.5 \\ x_{id} - (x_{id} - a_d)U_{(0,1)} & \text{, if } flip \geq 0.5 \end{cases} \tag{10}$$

Here $a = \sigma = 0.1(b_d - a_d)$, where a_d is the lower bound for coordinate d and b_d is the upper bound. Mutation operators are applied to particles position after the particle has completed it's position and velocity updates. This moves the particle to a new, randomized position, possibly dependant on the particles previous position.

4 Test Function Generator

Usually global optimization algorithms like PSO are evaluated against a small set of test functions. These functions serve as benchmarks against which relative worth of algorithms is judged. While there is no agreed set of functions to be used most papers usually use the same small set of functions. Examples of such functions can be found in a paper by Ali et al. (2005), Floudas et al. (1999), Horst et al. (2002) or indeed a large number of others. While having such a benchmark set is very convenient from the standpoint of the algorithm developer it has several big disadvantages. First of all details of test function implementation will leak in to the design of algorithms — namely research will fine tune the algorithms (often subconsciously) to solve those particular problems. Second any such set is always arbitrary. Why these test functions and not some others? Things are further complicated by theoretical results like the No-Free Lunch Theorem as can be found, for example, in the work of Wolpert et al. (1997). Third issue is that properties of such functions, such as the number and locations of local minima, etc. are often not known. We instead opted to use a method for generating test functions given a set of properties those test functions should satisfy.

We used a system for generating global optimization test functions proposed by Gaviano et al. (2011). It allows one to create three types of functions called ND-type, D-type and D2-type. The functions have a lot of parameters that influence their properties from the stand-point of global optimization. Those parameters are shared among all the types and have similar meanings in all of them. The main difference between the types is in their differentiability - ND-type functions are not differentiable in all of the argument space, D-type functions are once differentiable everywhere and D2-type functions are twice differentiable everywhere. Otherwise they are very similar. All of the parameters have the same meaning for all function types. An example of a D-type function is given in Figure 4.

$$f(x) = \begin{cases} C_i(\boldsymbol{x}), & \boldsymbol{x} \in S_i, i \in \{2, \ldots, m\}, \\ g(\boldsymbol{x}), & \boldsymbol{x} \notin S_2 \cup \ldots \cup S_m. \end{cases} \tag{11}$$

In the Equation (11) $\boldsymbol{S}_i$ is a "ball" (a hypersphere) defining the attraction region of the i-th local minimizer. In case of a two-dimensional function it is the circle that marks the area that the local minimizer occupies in solution space. See Figure 4 for an example of such a function.

$$g(\boldsymbol{x}) = \|\boldsymbol{x} - \boldsymbol{T}\|^2 + t, \boldsymbol{x} \in \Omega \tag{12}$$

$$C_i(\boldsymbol{x}) = \left(\frac{2}{\rho_i^2}\frac{\langle \boldsymbol{x} - \boldsymbol{M_i}, \boldsymbol{T} - \boldsymbol{M}_i\rangle}{\|\boldsymbol{x} - \boldsymbol{M}_i\|} - \frac{2}{\rho_i^3}A_i\right)\|\boldsymbol{x} - \boldsymbol{M}_i\| + \left(1 - \frac{4}{\rho_i}\frac{\langle \boldsymbol{x} - \boldsymbol{M}_i, \boldsymbol{T} - \boldsymbol{M}_i\rangle}{\|\boldsymbol{x} - \boldsymbol{M}_i\|} + \frac{3}{\rho_i^2}A_i\right)\|\boldsymbol{x} - \boldsymbol{M}_i\|^2 + f_i \tag{13}$$

$$A_i = \|\boldsymbol{T} - \boldsymbol{M}_i\|^2 + t - f_i \tag{14}$$

The parameters are summarized in Table 1. Given the large number of them it is not practical to always specify them by hand. Some kind of algorithm that would randomly fill in values of these parameters with respect to certain requirements is desirable. For example we may wish to simply specify number m of local minima and have that number of minima placed in random locations of our function. Gaviano (2011) et al. give just such an algorithm. The algorithm let's the user specify the values enumerated below. It then proceeds to randomly generate a test function in accordance to these values. In Algorithm 2 we give the pseudocode for this procedure as it was used in the experiments in this article. Below we enumerate the parameters that this algorithm takes.

1. The number of problem dimensions N, $N \geq 2$.
2. Number of local minima m, $m \geq 2$, including the minimizer $\boldsymbol{T}$ of the main paraboloid.
3. Value of the global minimum f_2. It must be chosen in such a way that $f_2 < t$. This is done to prevent the creation of easy functions where the vertex of the main paraboloid is the actual global minimum.
4. Radius of the attraction region of the global minimizer ρ_2. It must satisfy $0 < \rho_2 \leq 0.5r$ so that global minimizer attraction region does not overlap with the vertex of the main paraboloid thus making it trivial to find.
5. Distance r from the global minimizer to the vertex of the main paraboloid. Must be chosen to satisfy $0 < r < 0.5 \min\limits_{1 \leq j \leq N} |b(j) - a(j)|$ in order to make sure that $\boldsymbol{M}_2$ lies within Ω for all possible values of $\boldsymbol{T}$.

Next let us summarize the operation of this procedure.

Line 1 Initialize the vertex of the main paraboloid randomly and so that it lies within Ω we used $\Omega = [-1, 1]^N$. Note that here and elsewhere $\boldsymbol{U}_{(a,b)}$ is a vector of uniformly distributed random numbers each of which lies within (a, b) and $U_{(a,b)}$ is a similarly defined scalar value.

Lines 2-5 Initialize the location of the global minimum. It is initialized in such a way that it lies on the boundary of the sphere with radius ρ_2 and centered at $\boldsymbol{T}$. Generalized spherical coordinates are used for this aim. Here $\phi_1 \leftarrow U_{(0,\pi)}$ and $\phi_k \leftarrow U_{(0,2\pi)}$ for $2 \leq k \leq N$.

Lines 6-10 If some coordinate of the global minimum falls outside Ω this is used to adjust them to fall within Ω.

Lines 11-13 Place local minima at random. However, while this is not made clear in the pseudo-code an additional requirement has to be satisfied which is $\|\boldsymbol{M}_i - \boldsymbol{M}_2\| - \rho_2 > \gamma, \gamma > 0$ where $\gamma = \rho_2$ and the purpose of which is to make sure that local minima don't lie too close to the global minimum.

Lines 14-22 Here we set the radii of the local minima $\rho_i, 3 \leq i \leq m$. At first they are set to half the distance to the closest other minimum. Then we try to enlarge the radii also making sure they don't overlap. And finally we multiply them by 0.99 so that they do not touch.

Lines 23-26 Finally we set the values for local minima making sure that $f_2 < f_i, 3 \leq i \leq m$. Here B_i is the boundary of the sphere S_i and Z_{B_i} is the minimum of the main paraboloid over B_i.

5 Experimental Procedure

Twelve experiments were performed overall. Each experiment corresponds to a set of different set of test function generator parameters. The different parameter sets are given by a Cartesian product $\{2, 6, 10\} \times \{0.4, 0.6\} \times \{1.0, 1.5\}$ which results in a set of triplets the first element of which is the number of local minima m, second element is the radius of the global minimizer attraction region ρ_2 and the third element is the distance from the global minimum to the vertex of the main paraboloid r. Each triplet was

Algorithm 2 Algorithm for parameter selection for D-type functions.

```
 1: $\boldsymbol{T} \leftarrow \boldsymbol{U}_{(-1,1)}$
 2: for $j \leftarrow 1, \dots, N-1$ do
 3:     $M_{2j} \leftarrow T_j + r \cos\phi_j \prod_{k=1}^{j-1} \sin\phi_k$
 4: end for
 5: $M_{2N} \leftarrow T_N + r \prod_{k=1}^{N-1} \sin\phi_k$
 6: for $j \leftarrow 1, \dots, N$ do
 7:     if $M_{2j} \notin \Omega$ then
 8:         $M_{2j} \leftarrow 2T_j - x_j$
 9:     end if
10: end for
11: for $i \leftarrow 3, \dots, m$ do
12:     $\boldsymbol{M}_i \leftarrow \boldsymbol{U}_{(-1,1)}$
13: end for
14: for $i \leftarrow 3, \dots, m$ do
15:     $\rho_i \leftarrow 0.5 \min_{2 \le j \le m, j \ne i} \|M_i - M_j\|$
16: end for
17: for $i \leftarrow 3, \dots, m$ do
18:     $\rho_i \leftarrow \max\left(\rho_i, \min_{2 \le j \le m, j \ne i} (\|M_i - M_j\| - \rho_j)\right)$
19: end for
20: for $i \leftarrow 3, \dots, m$ do
21:     $\rho_i \leftarrow 0.99\rho_i$
22: end for
23: for $i \leftarrow 3, \dots, m$ do
24:     $\gamma_i \leftarrow \min(U_{(\rho_i, 2\rho_i)}, U_{(0, Z_{B_i} - f_2)})$
25:     $f_i \leftarrow \min\{g(\boldsymbol{x}) : \boldsymbol{x} \in B_i\} - \gamma_i$
26: end for
```

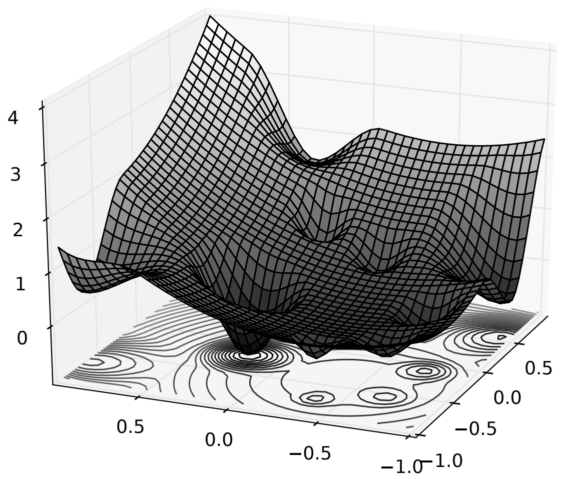

Fig. 1. An example of a D-type function generated using the algorithmic procedure.

Parameter	Meaning
N	Dimensionality of the test function.
t	Value of the function at the minimum of the main paraboloid.
$\boldsymbol{T}$	Location of the minimum of the main paraboloid.
f_i	Minimum value of the i-th local minimizer, where $i \in (2, \ldots, N)$ and $f_1 = t$.
$\boldsymbol{M}_i$	Location of the i-th local minimizer.
ρ_i	Attraction radius of the i-th local minimizer. The smaller this value is the harder it will be to locate that local minimum.

Table 1. Test function parameters.

tested against different mutation rates. Mutation rate can take values 0.1, 0.5, 1.0, 2.0 and 5.0. For each triplet and mutation rate combination 100 runs of the PSO algorithm were performed 1000 iterations each. During each run a new test function is generated using the parameters in the triplet and using Algorithm 2 to generate a corresponding set of test function parameters. After running the algorithm for 1000 iterations we attempt to determine if the run was successful. A run is deemed successful if the best solution found is within $\rho_2/2$ distance from the location of global minimum, which means it is in the attraction region of the global minimizer and thus we treat it as having found the globally best solution. The swarm consisted of 25 particles. The only two parameters of the PSO algorithm we used were set to $\phi_1 = \phi_2 = 2.05$. As particle swarm topology von Neumann (a 5×5 grid) neighbourhood was used.

After the results were obtained the percentage of successful runs was ploted against mutation rate for each type of mutation operator.

6 Results

Results are given graphically in Figures 2-13. In each figure percentage of successful runs is ploted against mutation rate. Each mutation operator is represented by a distinct dashed line.

The first conclusion to be made from the plots is that mutation certainly seems to improve PSO performance significantly, if performance of the algorithm is to be measured as that algorithm being able to locate the global minimum. In all cases PSO without mutation performed the worst of the bunch. The two operators based on the Gaussian distribution also consistently performed worse than the rest. This can probably be explained by the sharply decaying “tails” of the Gaussian distribution that are not enough to bring enough variation to discover the global minimum it is further from the current attraction points of the swarm. Further, it can be seen that the lines in the plots usually plot two groups of three. The first group is Uniform, Cauchy and Michalewitz operators and the second group is Gaussian 1, Gaussian 2 and None operators. The uniting thing among the members of the first group is that they are not limited to values near the current value of the coordinate being mutated. While Cauchy operator may seem similar to Gaussian operators Cauchy distribution has far fatter “tails” than Gaussian and thus chances are that the mutated coordinate will end up further from it’s

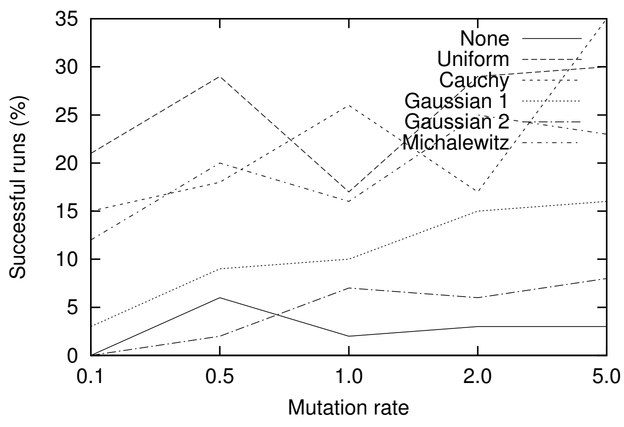

Fig. 2. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 2$, $\rho_2 = 0.4$ and $r = 1.0$.

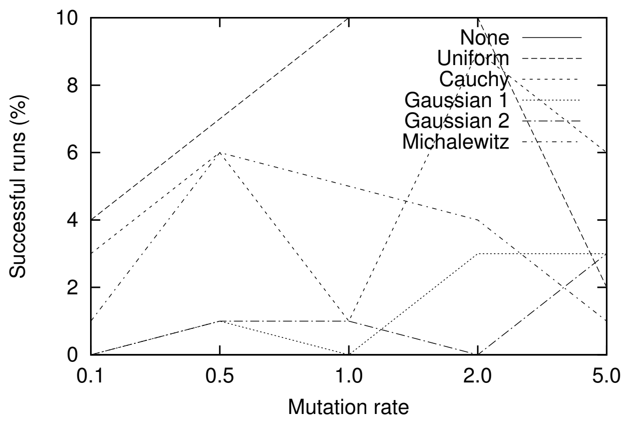

Fig. 3. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 2$, $\rho_2 = 0.4$ and $r = 1.5$.

current position. This allows to explore the solution space better. There are no reasons to suppose that global minimum will be near the current attraction regions of the particle swarm. As such it seems to as misguided to use operators that are dependent on the current position such as Gaussian 1, Gaussian 2 or even Cauchy operators. Results seem to support this and even though Cauchy gives results comparable to those of Uniform and Michalewitz operators it is more complicated than them and it's use does not seem to justify. It is our recommendation thus to start with the simplest operator — Uniform operator, since there does not seem to be enough justification to use the more complicated ones. Obviously we cannot exclude the possibility that there may be functions

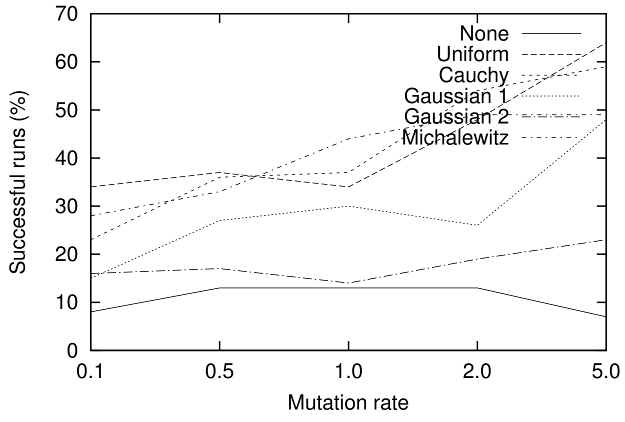

Fig. 4. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 2$, $\rho_2 = 0.6$ and $r = 1.0$.

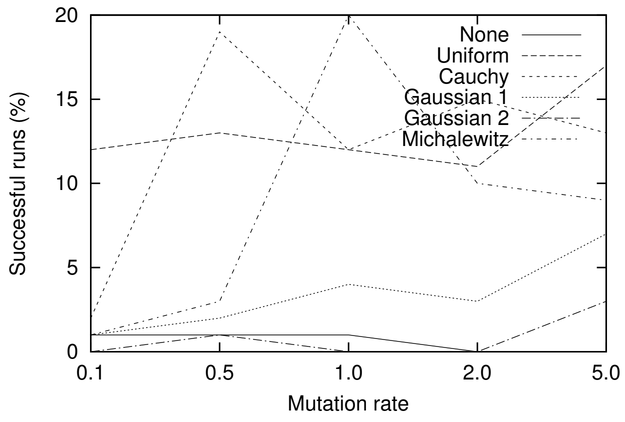

Fig. 5. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 2$, $\rho_2 = 0.6$ and $r = 1.5$.

that may benefit from them as per No-Free Lunch theorem as described, for example, in a work by Wolpert et al. (1997).

Another issue that arises when using mutation operators is the question of the mutation rate. What should the mutation rate value be? Should it be kept constant or should it change with time? We have only examined constant mutation rates. From our results it seems that fairly high values of mutation rate are beneficial. This is in contrast with recommendations from researchers like Andrews (2006) who recommends mutation rates of just 0.5 to be used, mutation rate 1 is recommended by Higashi et al. (2003). In our experience this is far too low. In many cases we found that mutation rates of 2 or 5 or possibly going even higher could be justified. We feel that this disrepancy in the results

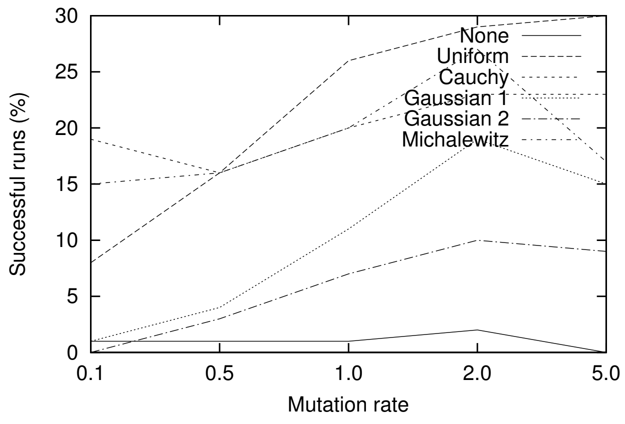

Fig. 6. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 6$, $\rho_2 = 0.4$ and $r = 1.0$.

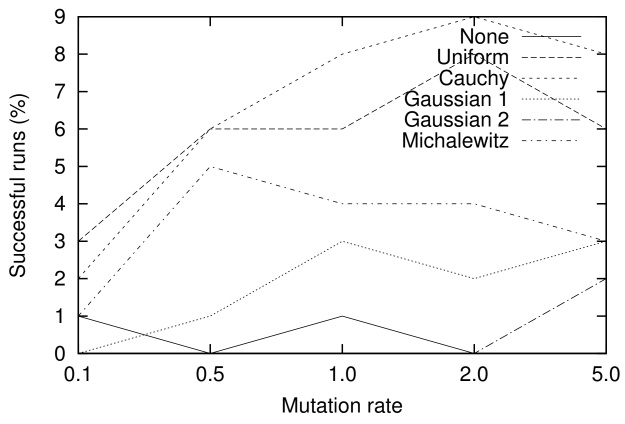

Fig. 7. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 6$, $\rho_2 = 0.4$ and $r = 1.5$.

is the result of the different performance metric used. Most researchers use average value of the lowest result obtained during each run to measure swarm performance over some test function. This works well if the goal to measure how well does local search works. However it does not work so well if we want to know if the global minimum of a multi-modal function was detected. Higher values of mutation rate will tend to reduce local search performance because it prevents the swarm from converging as easily. As such if the average value method of measuring performance is used it will tend to favor lower mutation rates on many test functions, especially unimodal ones. We feel that finding the global minimum is a more important goal. Fine tuning of the solution once the global minimum was detected can be performed by simply switching mutation of

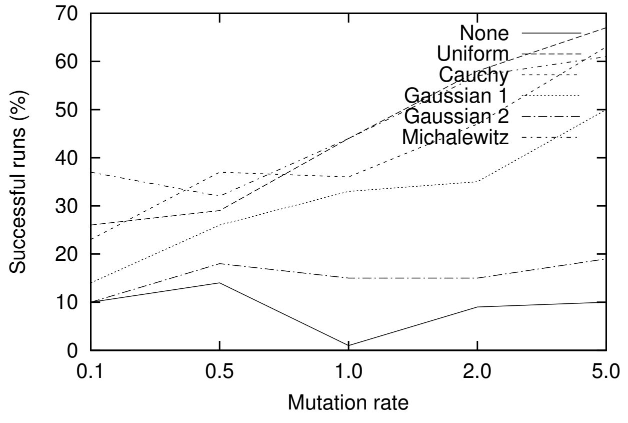

Fig. 8. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 6$, $\rho_2 = 0.6$ and $r = 1.0$.

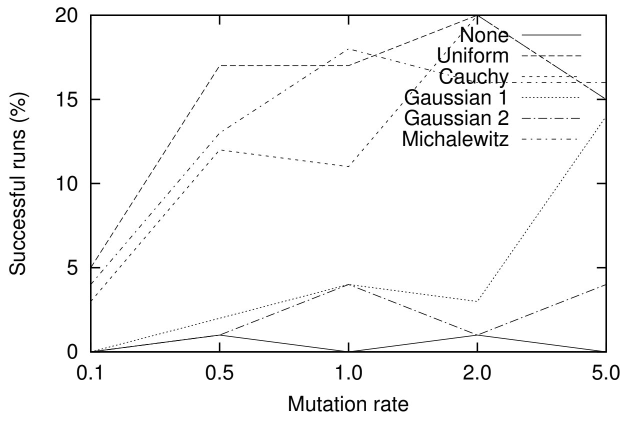

Fig. 9. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 6$, $\rho_2 = 0.6$ and $r = 1.5$.

completely. Another solution is to start with a high mutation value and decrease it with time.

7 Conclusions

In this work we have empirically evaluated the performance of several mutation operators when applied to PSO algorithm. We did this by using the algorithm to optimize several test functions that were generated using different parameters. Results were analyzed and following conclusions can be offered:

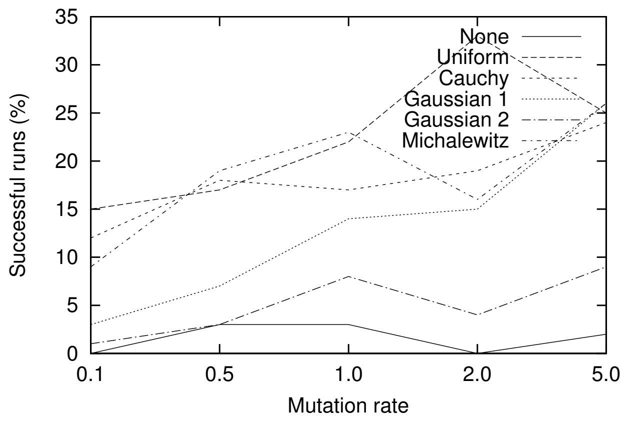

Fig. 10. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 10$, $\rho_2 = 0.4$ and $r = 1.0$.

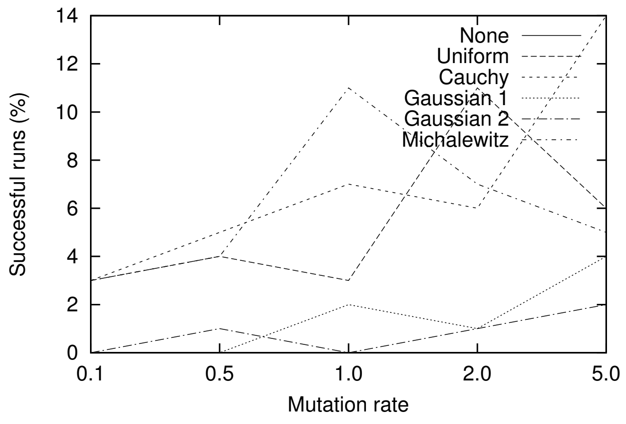

Fig. 11. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 10$, $\rho_2 = 0.4$ and $r = 1.5$.

1. Using mutation operators significantly improves the performance of PSO algorithm.
2. Mutation rates that are higher than those usually reported in literature should be examined. Namely we got best results with mutation rates $2 - 5$ in most of the cases.
3. There is little need to use elaborate mutation operators based on Cauchy or Gaussian distributions. A simple reinitialization of the coordinate using a random uniformly distributed number in the acceptable interval is sufficient and indeed usually outperforms more elaborate operators.

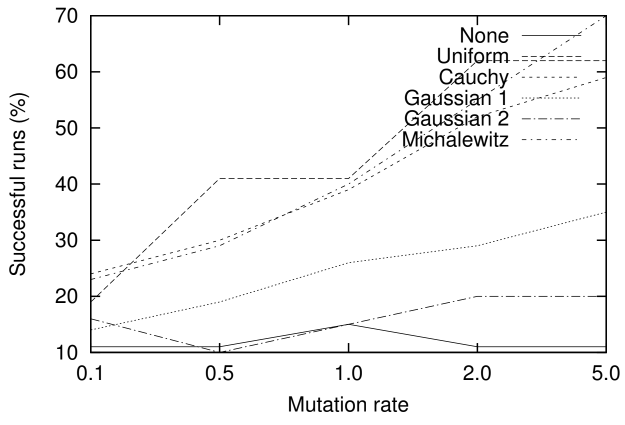

Fig. 12. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 10$, $\rho_2 = 0.6$ and $r = 1.0$.

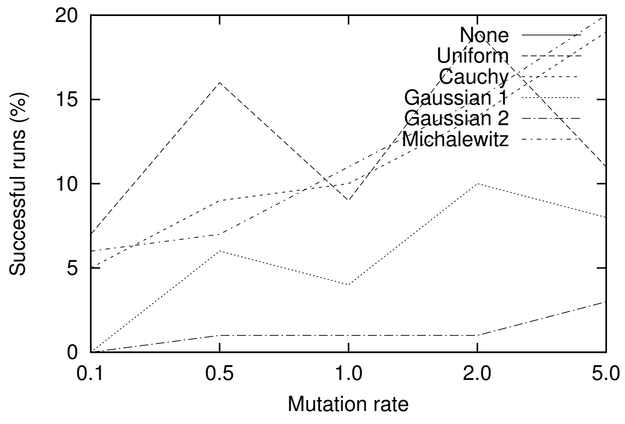

Fig. 13. Percentage of successful runs vs. mutation rate for test functions generated with parameters $n = 10$, $f_2 = -1.0$, $m = 10$, $\rho_2 = 0.6$ and $r = 1.5$.

Acknowledgements

This research is supported by the Research Council of Lithuania under Grant No. MIP-063/2012.

References

Ali, M. M., Khompatraporn, C., Zabinsky, B. Z. (2005). A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems. *Journal of Global Optimization 31(4)*, 635–672.

Andrews, S. P. (2006). An investigation into mutation operators for particle swarm optimization. In *IEEE Congress on Evolutionary Computation, 2006*, 1044–1051.

Clerc, M., Kennedy, J. (2002). The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation 6(1)*, 58–73.

Eberhart, C. R., Shi, Y. (2000). Comparing inertia weights and constriction factors in particle swarm optimization. In *Proceedings of the 2000 Congress on Evolutionary Computation*, 84–88.

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

Esquivel, C. A., Coello Coello, C. A. (2003). On the use of particle swarm optimization with multimodal functions. In *The 2003 Congress on Evolutionary Computation*, 1130–1136.

Floudas A. C., Pardalos M. P, Adjiman S. C., Esposito R. W., Gumus H. Z., Harding S. T., Klepeis L. J., Meyer A. C., and Schweiger A. C. (1999). *Handbook of test problems in local and global optimization*.

Gaviano, M., Kvasov, E. D., Lera, D., Sergeyev, D. Y. (2011). Software for generation of classes of test functions with known local and global minima for global optimization. *ACM Transactions on Mathematical Software (TOMS)*, 469–480.

Higashi, N., Iba, H. (2003). Particle swarm optimization with gaussian mutation. In *Proceedings of the 2003 IEEE Swarm Intelligence Symposiom*, 72–79.

Horst, R., Pardalos, M. P., Romeijn, H. E. (2002). *Handbook of global optimization*.

Eberhart, C. R., Kennedy, J. (1995). Particle swarm optimization. *Proceedings of IEEE International Conference on Neural Networks*, 1942 – 1948.

Kennedy, J. (1999). Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Proceedings of the 1999 Congress on Evolutionary Computation*.

Kennedy, J., Mendes, R. (2002). Population structure and particle swarm performance. In *Proceedings of the 2002 Congress on Evolutionary Computation*, 1671–1676.

Kennedy, J., Eberhart, C. R. (2001). *Swarm intelligence*. Morgan Kaufmann.

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs*. Springer.

Ratnaweera A., Halgamuge K. S., Watson C. H. (2004). Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients. *IEEE Transactions on Evolutionary Computation*, 240–255.

Shi, Y., Eberhart, C. R. (1998). Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, 591–600.

Stacey, A., Jancic, M., Grundy, I. (2003). Particle swarm optimization with mutation. In *The 2003 Congress on Evolutionary Computation*, 1425–1430.

Vesterstrom, J., Thomsen, R. (2004). A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Congress on Evolutionary Computation, 2004.*, 1980–1987.

Wolpert, H. D., Macready, G. W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 67–82.

Received June 18, 2014 , revised October 8, 2014, accepted October 8, 2014