A Comparison of Several Heuristic Algorithms for Solving High Dimensional Optimization Problems

Preliminary Communication

Emmanuel Karlo Nyarko
J. J. Strossmayer University of Osijek,
Faculty of Electrical Engineering, Department of Computer Engineering and Automation
Kneza Trpimira 2B, 31000 Osijek, Croatia
nyarko@etfos.hr

Robert Cupec
J. J. Strossmayer University of Osijek,
Faculty of Electrical Engineering, Department of Computer Engineering and Automation
Kneza Trpimira 2B, 31000 Osijek, Croatia
robert.cupec@etfos.hr

Damir Filko
J. J. Strossmayer University of Osijek,
Faculty of Electrical Engineering, Department of Computer Engineering and Automation
Kneza Trpimira 2B, 31000 Osijek, Croatia
damir.filko@etfos.hr

Abstract *– The number of heuristic optimization algorithms has exploded over the last decade with new methods being proposed constantly. A recent overview of existing heuristic methods has listed over 130 algorithms. The majority of these optimization algorithms have been designed and applied to solve real-parameter function optimization problems, each claiming to be superior to other methods in terms of performance. However, most of these algorithms have been tested on relatively low dimensional problems, i.e., problems involving less than 30 parameters. With the recent emergence of Big Data, the existing optimization methods need to be tested to find those (un)suitable to handle highly dimensional problems. This paper represents an initial step in such direction. Three traditional heuristic algorithms are systematically analyzed and tested in detail for problems involving up to 100 parameters. Genetic algorithms (GA), particle swarm optimization (PSO) and differential evolution (DE) are compared in terms of accuracy and runtime, using several high dimensional standard benchmark functions.*

Keywords *– heuristic optimization, high dimensional optimization, nature-inspired algorithms, optimization techniques*

1. INTRODUCTION

Optimization is the process of minimizing or maximizing a goal (or goals) taking into consideration the existing constraints. Optimization algorithms are basically iterative in nature and as such the *quality* of an optimization algorithm is determined by the *quality* of the result obtained in a finite amount of time. Global optimization algorithms can generally be divided into two categories: deterministic and probabilistic algorithms [1]. The main difference between the two categories is that deterministic algorithms are designed such that the optimal solution is always found in a finite amount of time. Thus, deterministic algorithms can only be implemented in situations where the search space can efficiently be explored. In situations where the search space cannot be efficiently explored, e.g., high dimensional search space, implementing a deterministic algorithm might result in exhaustive search which would be unfeasible due to a time constraint. In such situations, probabilistic algorithms are used. Probabilistic algorithms generally optimize a problem by iteratively trying to improve a candidate solution with respect to a given measure of quality. They make few or no assumptions about the problem being optimized and can search very large spaces of candidate solutions. However, probabilistic algorithms provide no guarantee of an optimal solution being found, only a good solution in a finite amount of time.

Examples of deterministic optimization algorithms include the pattern search or direct search by Hooke and Jeeves [2], the Nelder-Mead method [3] and the Branch and Bound algorithm [4], while examples of probabilistic algorithms include Genetic Algorithms (GA) [5,6], Differential evolution (DE) [7,8], Particle Swarm Optimization (PSO) [9,10] and Ant Colony Optimization (ACO) [11,12], to name but a few.

Heuristics used in global optimization are functions or methods that help us decide which solution candidate is to be examined or tested next or how the next solution candidate can be produced. Deterministic algorithms usually employ heuristics in order to define the processing order of solution candidates. Probabilistic methods, on the other hand, may only consider those elements of the search space in further computations that have been selected by the heuristic [1]. In this paper, the term heuristic algorithms refers to probabilistic algorithms employing heuristic methods.

Real-world optimization problems are often very challenging to solve, and are often NP-hard problems. Thus, heuristic algorithms are usually employed. Many heuristic algorithms using various optimization techniques have been developed to deal with these challenging optimization problems. The number of heuristic optimization algorithms has exploded over the last decade with new methods being proposed constantly. A recent overview of the existing heuristic methods has listed over 130 algorithms [13]. These algorithms can be classified into four main groups: biology-, physics-, chemistry-, and mathematics-based algorithms depending on the source of inspiration for the researchers. The largest group of heuristic optimization algorithms is biology-based, i.e., bio-inspired. Two of the most important subsets of heuristic algorithms, which are coincidentally bio-inspired, are Evolutionary Algorithms (EA) and Swarm Intelligence (SI). GA and DE are the most well-known evolutionary algorithms, while PSO is a well-known swarm intelligence algorithm.

The majority of these optimization algorithms have been designed and applied to solve real-parameter function optimization problems, each claiming to be superior to other methods in terms of performance [13, 14]. However, most of these algorithms have been tested on relatively low dimensional problems, i.e., problems involving less than 30 parameters. With the recent emergence of Big Data, the existing optimization methods need to be tested to find those (un)suitable to handle highly dimensional problems. This paper represents an initial step in such direction. The main focus of this paper is to analyze, test and compare in detail, GA, DE and PSO in solving high dimensional real-parameter optimization problems, especially in terms of accuracy and runtime. Standard benchmark functions with up to 100 parameters are used. All tests and analyses are conducted using Matlab. The rest of this paper is structured as follows. In Section 2, a description of the optimization problem is provided. An overview of the heuristic algorithms GA, DE and PSO is provided in Section 3, while the test results obtained while solving three examples of high dimensional real-parameter optimization problems are given and analyzed in Section 4. Finally, the paper is concluded with Section 5.

2. PROBLEM DESCRIPTION

Many real-world optimization problems from engineering, biology and other disciplines can often be expressed as optimization of a continuous function. These functions depend on a set of parameters, the choice of which affects the performance or objectives of the system concerned. The optimization goal is often measured in terms of objective or fitness functions in qualitative models.

The problem considered in this paper can be formulated as follows. Given an objective function

$$f : \mathbb{X} \mapsto Y \,, \tag{1}$$

where $\mathbb{X} \subseteq \mathbb{R}^D$ and $Y \subseteq \mathbb{R}$, one has to estimate the optimal parameter vector $\mathbf{x}^*$ such that

$$f\left(\mathbf{x}^*\right) = \max_{\mathbf{x} \in \mathbb{R}^D} f\left(\mathbf{x}\right) , \tag{2}$$

where $\mathbf{x} = \left(x_1, \ldots x_D\right)$ represents a vector of real parameters of dimension D.
Since $\min\left\{f\left(\mathbf{x}\right)\right\} = -\max\left\{-f\left(\mathbf{x}\right)\right\}$, the restriction to maximization is without loss of generality. The domains of real parameters are defined by their lower and upper bounds: $low_j, up_j; j \in \left\{1, 2, \ldots, D\right\}$.

In practice, no a priori knowledge of the objective function exists, and it can generally be assumed that the objective function is nonlinear and may have multiple local minima. In this paper, the objective function will also be referred to as the fitness function or the *quality* of the parameter vector. The quality or fitness of a candidate solution $\mathbf{x}_i$ is defined by

$$f_i = f\left(\mathbf{x}_i\right) \tag{3}$$

3. AN OVERVIEW OF GA, DE AND PSO

GA, DE and PSO have been implemented in a wide variety of real-parameter function optimization problems, some of which include speech synthesis, antenna design, genes design, neural network learning, modeling of chemical and biochemical processes [15], radio network design [16], segmentation of brain MR images [17], etc.

GA, DE and PSO are population based algorithms and as such, they always *work* on a set of candidate solutions $\mathbf{X} = \left(\mathbf{x}_{1,} \ldots \mathbf{x}_N\right)^{\mathrm{T}}$ during each iteration of the algorithm. N represents the number of candidate solutions and is usually kept constant during the execution of the algorithm.

3.1. GENETIC ALGORITHMS (GA)

Concepts from biology, genetics and evolution are freely borrowed to describe GA. The element x_j, j=1,...,D, a candidate solution $\mathbf{x}_i$, i=1,...,N, the set of candidate solutions $\mathbf{X}$ and an iteration of the algorithm are referred to as a gene, an individual, a population and a generation, respectively. During the execution of the algorithm, a candidate solution or parent is modified in a particular way to create a new candidate solution or child.

The basic evolutionary computation algorithm first constructs an initial population, then it iterates through three procedures. It first assesses the quality or fitness of all individuals in the population. Then it uses this fitness information to reproduce a new population of children. Finally, it merges the parents and children in some fashion to form a new next-generation population, and the cycle continues. This procedure is outlined in Algorithm 1 [14].

kmax denotes the maximum number of iterations to be performed, while $\mathbf{x}_{BEST}$ represents the best solution found by the EA. All algorithms analyzed herein generate the initial population or set of candidate solutions randomly according to equation :

$$\mathbf{X}_{i,j} \sim U\left(low_j, up_j\right), \tag{4}$$

for i=1,...,N and j=1,...D, where $\mathbf{X}_{i,j}$ denotes the j-th element, x_j, of the i-th vector, $\mathbf{x}_i$. $U(low_j, up_j)$ is a random number in $[low_j, up_j]$drawn according to uniform distribution and the symbol ~ denotes sampling from the given distribution.

Algorithm 1: An abstract Evolutionary Algorithm (EA)

Input: N, *kmax*
Output: $\mathbf{x}_{BEST}$
1: $\mathbf{x}_{BEST} \leftarrow \emptyset$, f_{BEST}=0
2: Build an initial population $\mathbf{X}$
k:=0
3: **repeat**
4: k:=k+1
4: for each individual $\mathbf{x}_i$ in $\mathbf{X}$
5: Calculate f_i
6: **If** $\mathbf{x}_{BEST} = \emptyset$ **or** , $f_i > f_{BEST}$, **then**
7: $\mathbf{x}_{BEST} \leftarrow \mathbf{x}_i$
$f_{BEST} \leftarrow f_i$
8: **end if**
9: **end**
10: $\mathbf{X} \leftarrow$ Merge($\mathbf{X}$, Reproduce($\mathbf{X}$))
11: **until** $\mathbf{x}_{BEST}$ is the ideal solution or k > *kmax*

Evolutionary algorithms differ from one another largely in how they perform the *Reproduce* and *Merge* operations. The *Reproduce* operation usually has two parts: Selecting parents from the old population, then creating new individuals or children (usually mutating or recombining them in some way) to generate children. The *Merge* operation usually either completely replaces the parents with the children, or includes fit parents along with their children to form the next generation [14].

The stopping condition of the algorithm is often defined in a few ways, i.e., 1) limiting the execution time of the algorithm. This is normally done either by defining the maximum number of iterations, as shown in Algorithm 1, or by limiting the maximum number of fitness function evaluations; 2) f_{BEST} does not change appreciably over successive iterations; 3) attaining a pre-specified objective function value.

One of the first EA is GA invented by John Holland in 1975 [5]. The standard GA consists of three genetic operators, i.e., selection, crossover and mutation. During each generation, parents are selected using the selection operator. The selection operator selects individuals such that individuals with better fitness values have a greater chance of being selected. Then new individuals, or children, are generated using the crossover and mutation operators. The *Reproduce* operation used in Algorithm 1 consists of these 3 operators. There are many variants of GA due to the different selection, crossover and mutation operators proposed, some of which can be found in [1, 5-6, 14, 18-20]. The GA analyzed in this paper is available in the Global Optimization Toolbox of Matlab R2010a. The implemented genetic operators used in this study are defined as follows.

Selection

The selection function used in this paper is the Stochastic Universal Sampling (SUS) method [20]. Parents are selected in a biased fitness-proportionate way such that fit individuals get picked at least once. This method can be explained by means of Fig. 1, which shows an array of all individuals sized by their fitness values (N = 7). It can be noticed that $f_4 > f_7 > f_2 > f_1 > f_6 > f_3 > f_5$. The total fitness range f_Σ is initially determined by using equation :

$$f_\Sigma = \sum_{i=1}^{N} f_i \ . \tag{4}$$

Then, the sampling length f_Σ / N_S is determined, where N_S denotes the number of individuals that need to be selected from the entire population.

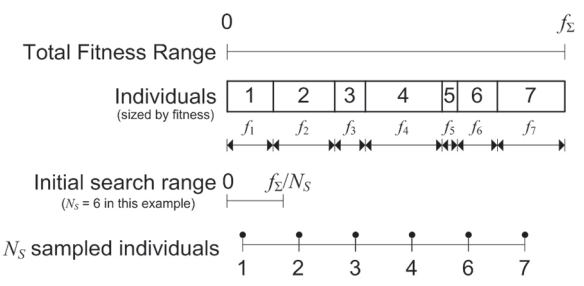


Fig.1. Array of individual ranges, initial search range, and chosen positions in Stochastic Universal Sampling.

A random position is generated between 0 and f_Σ / N_S and the individual covering this position is selected as the first individual. The value f_Σ / N_S is then added to this initial position to determine the second position and, thus, the second individual. Hence, each subsequent individual is selected by adding the value f_Σ / N_S to the previous position. This process is performed until N individuals have been selected.

Crossover

The representation of an individual in GA determines the type of crossover and mutation operators that can be implemented. By far, the most popular representation of an individual in GA is the vector representation. Depending on the problem, the individual can be defined using a boolean vector, an integer vector or a real-valued vector as is the case in this paper.

The crossover operator used in this paper is the Scattered or Uniform crossover method. Assuming the parents $\mathbf{x}_i$ and $\mathbf{x}_k$ have been selected, a random binary vector or *mask* is generated. The children $\mathbf{x}_{i,\,new}$ and $\mathbf{x}_{k,\,new}$ are then formed by combining genes of both parents. This recombination is defined by equations (6) and (7):

$$\mathbf{x}_{i,new}(j) = \begin{cases} \mathbf{x}_i(j), & if\ mask(j) = 1 \\ \mathbf{x}_k(j), & otherwise \end{cases} \tag{6}$$

$$\mathbf{x}_{k,new}(j) = \begin{cases} \mathbf{x}_k(j), & if\ mask(j) = 1 \\ \mathbf{x}_i(j), & otherwise \end{cases} \tag{7}$$

The number of children to be formed by the crossover operator is provided by a user defined parameter $P_{crossover}$ which represents the fraction of the population involved in crossover operations.

The crossover operator tends to improve the overall quality of the population since better individuals are involved. As a result, the population will eventually converge, often prematurely, to copies of the same individual. In order to introduce new information, i.e., to move to unexplored areas of the search space, the mutation operator is needed.

Mutation

The Uniform mutation operator is used in this paper. Uniform mutation is a two-step process. Assuming an individual has been selected for mutation, the algorithm selects a fraction of vector elements for mutation. Each element has the same probability, $R_{mutation}$, of being selected. Then the algorithm replaces each selected element by a random number selected uniformly from the domain of that element. For example, assuming the element x_j of the individual $\mathbf{x}_i$ has been selected for mutation, then the value of element x_j is changed by generating a random number from $U(low_j, up_j)$.

In order to guarantee convergence of GA, an additional feature - elitism is used. Elitism ensures that at least one of the best individuals of the current generation is passed on to the next generation. This is often a user defined value, N_{elite}, and it indicates the top N_{elite} individuals, ranked according to their fitness values that are copied on to the next generation directly.

3.2. DIFFERENTIAL EVOLUTION (DE)

DE is a very powerful yet simple real-parameter optimization algorithm proposed by Storn and Price about 20 years ago [7, 8]. As with GA, a lot of variants of the basic algorithm with improved performance have been proposed [21, 22]. The evolutionary operations of classical DE can be summarized as follows [8].

Mutation

Mutation of a given individual $\mathbf{x}_i$ is defined by

$$\mathbf{v}_i = \mathbf{x}_k + F \cdot (\mathbf{x}_m - \mathbf{x}_n) \tag{8}$$

where $i,k,m,n \in [1,N]$ are mutually different, $F > 0$ is the mutation scale factor used to control the differential variation $\mathbf{d}_i = (\mathbf{x}_m - \mathbf{x}_n)$.

Crossover

The crossover operator is defined by equation (9):

$$\mathbf{u}_i(j) = \begin{cases} \mathbf{v}_i(j), & if\ U(0,1) < CR \\ \mathbf{x}_i(j), & otherwise \end{cases} \tag{9}$$

where $CR \in (0,1)$ is the crossover rate and it controls how many elements of an individual are changed. $\mathbf{u}_i$ is the new individual generated by recombining the mutated individual $\mathbf{v}_i$ and the original individual $\mathbf{x}_i$. This operator is basically the Uniform crossover ((6) or (7)), except for the fact that only one child is generated.

Selection

The selection operator is defined by equation :

$$\mathbf{x}_{i,new} = \begin{cases} \mathbf{u}_i, & if\ f(\mathbf{u}_i) > f(\mathbf{x}_i) \\ \mathbf{x}_i, & otherwise. \end{cases} \tag{10}$$

Thus, the individual $\mathbf{x}_i$ is replaced by the new individual $\mathbf{u}_i$ only if $\mathbf{u}_i$ represents a better solution.

Based on these equations, it can be noticed that DE has three main control parameters: F, CR and N, which are problem dependent. Storn and Price [8] recommended N to be chosen between 5*D and 10*D, and F to be between 0.4 and 1. A lot of papers and research have been published indicating methods to improve the ultimate performance of DE by tuning its control parameters [23-25]. In this paper, a variant of the types of DE discussed in [22] is used, where the mutation scale factor and the crossover rate are generated randomly from continuous uniform distributions.

3.3. PARTICLE SWARM OPTIMIZATION (PSO)

PSO belongs to the set of swarm intelligence algorithms. Even though there are some similarities to EA, it

is not modeled after evolution but after swarming and flocking behaviors in animals and birds. It was initially proposed by Kennedy and Eberhart in 1995 [9]. A lot of variations and modifications of the basic algorithm have been proposed ever since [26, 27]. A candidate solution in PSO is referred to as a particle, while a set of candidate solutions is referred to as a swarm. A particle *i* is defined completely by 3 vectors: its position, $\mathbf{x}_i$, its velocity, $\mathbf{v}_i$, and its personal best position $\mathbf{x}_{i,Best}$. The particle moves through the search space defined by a few simple formulae. Its movement is determined by its own best known position, $\mathbf{x}_{i,Best}$, as well as the best known position of the whole swarm, $\mathbf{x}_{BEST}$. First, the velocity of the particle is updated using equation (11):

$$\mathbf{v}_{i,new} = c_0 \cdot \mathbf{v}_i + c_1 \cdot r_1 \cdot \left(\mathbf{x}_{i,Best} - \mathbf{x}_i\right) + c_2 \cdot r_2 \cdot \left(\mathbf{x}_{BEST} - \mathbf{x}_i\right), \tag{11}$$

then the position is updated using equation (12):

$$\mathbf{x}_{i,new} = \mathbf{x}_i + \mathbf{v}_{i,new} \tag{12}$$

where r_1 and r_2 are random numbers generated from $U(0,1)$, c_0 is the *inertia weight*, and c_1 and c_2 are the *cognitive* and *social acceleration* weights, respectively. Modern versions of PSO such as the one analyzed in this paper do not use the global best solution, $\mathbf{x}_{BEST}$, in equation (11) but rather the local best solution $\mathbf{x}_{i,LBest}$ [26, 28]. Hence, the velocity update equation is given by

$$\mathbf{v}_{i,new} = c_0 \cdot \mathbf{v}_i + c_1 \cdot r_1 \cdot \left(\mathbf{x}_{i,Best} - \mathbf{x}_i\right) + c_2 \cdot r_2 \cdot \left(\mathbf{x}_{i,LBest} - \mathbf{x}_i\right). \tag{13}$$

The local best solution of a given individual is determined by the best-known position within that particle's neighborhood. Different ways of defining the neighborhood of a particle can be found in [26, 28-31]. The analyzed PSO algorithm in this paper uses an adaptive random topology, where each particle randomly informs *K* particles and itself (the same particle may be chosen several times), with *K* usually set to 3. In this topology, the connections between particles randomly change when the global optimum shows no improvement [26, 28].

4. EXPERIMENTAL ANALYSIS

Standard benchmark test functions are used to test the accuracy, robustness and speed of convergence of optimization algorithms. Such benchmark functions are necessary especially when the quality of a proposed optimization algorithm needs to be assessed or when several optimization algorithms need to be compared under the same conditions. Examples of such benchmark test functions can be found in [32, 33]. These benchmark suites include unimodal, multimodal and composition functions. Among other functions, these benchmark suites always include three standard test functions, i.e., the Ackley function, the Rastrigin function and the Rosenbrock function, as well as their shifted and rotated variations. In this paper, these three standard test functions are used in their original form in the analyses.

Ackley's function, (14), is in its 2D form characterized by a nearly flat outer region, has a lot of local minima and a large hole at the center (Fig 2).

$$f(\mathbf{x}) = 20 + \exp(1) - 20\exp\left(-0.2\sqrt{\frac{1}{D}\sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D}\sum_{i=1}^{D}\cos(2\pi x_i)\right). \tag{14}$$

The domain is defined on the hypercube $x_i \in [-5,5]$, $\forall i = 1,\ldots,D$, with the global minimum $f(\mathbf{x}^*) = 0$, at $\mathbf{x}^* = (0,\ldots,0)$.

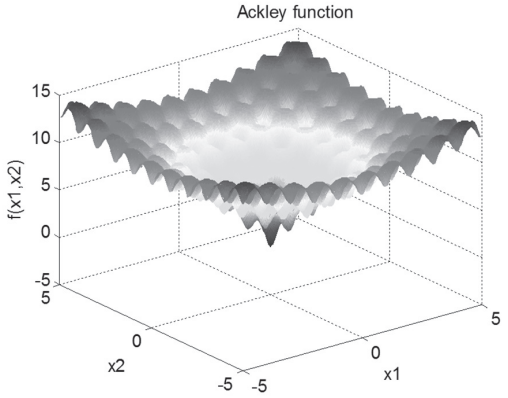


Fig. 2. Ackley function for $D = 2$

Rastrigin's function, (15), also has several local minima and is highly multimodal. The 2D form is shown in Fig 3.

$$f(\mathbf{x}) = 10D + \sum_{i=1}^{D}\left[x_i^2 - 10\cos(2\pi x_i)\right] \tag{15}$$

The domain is defined on the hypercube $x_i \in [-5.12, 5.12]$, $\forall i = 1,\ldots,D$, with the global minimum $f(\mathbf{x}^*) = 0$ also, at $\mathbf{x}^* = (0,\ldots,0)$

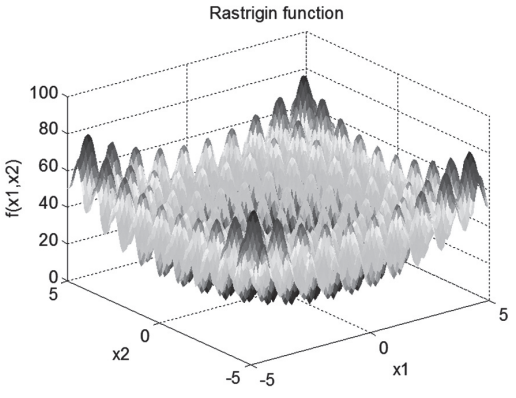


Fig. 3. Rastrigin function for *D* = 2

On the other hand, the Rosenbrock function, (16), which is a popular test problem for gradient-based optimization algorithms, is unimodal, and the global minimum lies in a narrow, parabolic valley. Even though this valley is easy to find, convergence to the minimum is difficult [34]. The 2D plot is shown in Fig 4.

$$f(\mathbf{x}) = \sum_{i=1}^{D-1}\left[100\left(x_{i+1} - x_i^2\right)^2 + \left(x_i - 1\right)^2\right] \quad (16)$$

The domain is defined on the hypercube $x_i \in [-5,5], \forall i = 1,\ldots,D$, with the global minimum , $f(\mathbf{x}^*) = 0$ at $\mathbf{x}^* = (1,\ldots,1)$.

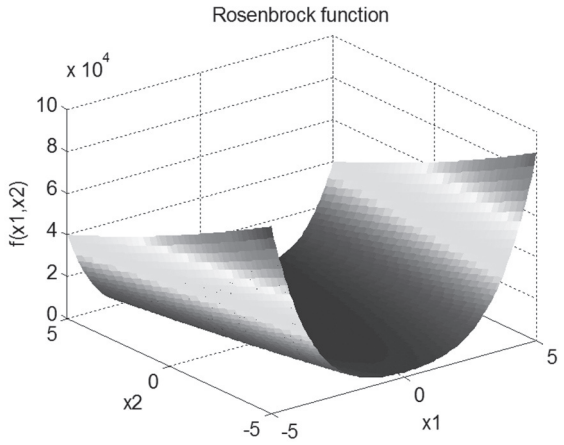


Fig. 4. Rastrigin function for D = 2

GA, DE and PSO were tested on these 3 test functions for D = 2, 5, 10, 50 and 100. All analyses were performed in Matlab. The algorithm specific control parameters values are given in Table 1. Similarly to [32, 33], all three algorithms had the same maximum number of fitness function evaluations defined in order to ensure a fair comparison. Hence, the common control parameters for the heuristic algorithms were:

- The size of the solution set, N = 50.
- The maximum number of iterations, *kmax* = 3,000.

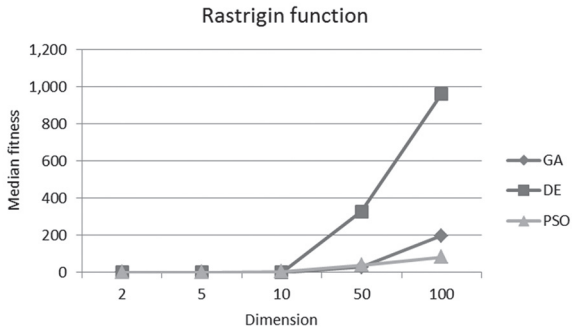
All experiments were performed 100 times. Details of the results are presented in Figures 5 and 6. Analyzing the results displayed in Figures 6 and 7, it can be noticed that:

- for optimization problems with $D \leq 10$, all three algorithms had comparable results;
- for $D \geq 50$, GA and PSO performed better than DE especially for the Rastrigin and Rosenbrock functions. However, GA showed degraded performance with the Ackley function;
- the runtime of PSO rapidly increases with an increase in the dimensionality of the problem, while that of GA and DE remains relatively low.

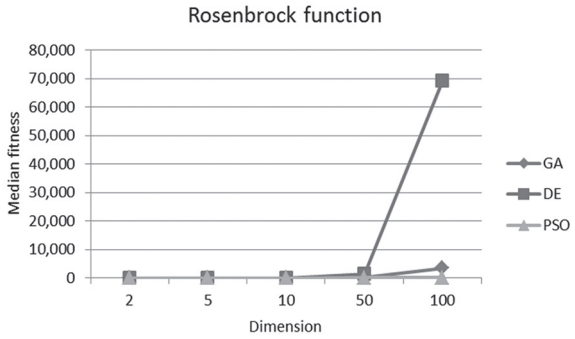
It can be concluded that PSO, in general, has better accuracy for high dimensional problems but with very poor runtime performance. If the runtime is the main condition, then GA is a better optimization tool. However, care should be taken if the optimization problem is similar to the Ackley function.

Table 1. Algorithm specific control parameter values used in the experiments.

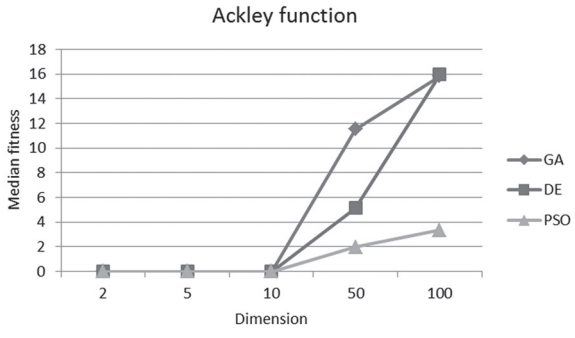
Algorithm	Control parameters
GA	Nelite = 2; $P_{crossover}$ = 0.8; $R_{mutation}$ = 0.01
DE	$F \sim U(0.5,2)$; $CR \sim U(0.2,0.9)$
PSO	$c_0 = \frac{1}{2 \cdot \ln(2)}$; $c_1 = c_2 = 0.5 + \ln(2)$



(a)



(b)



(c)

Fig. 5. Accuracy performance of the heuristic algorithms for 100 trials for a) Rastrigin b) Rosenbrock and c) Ackley function .

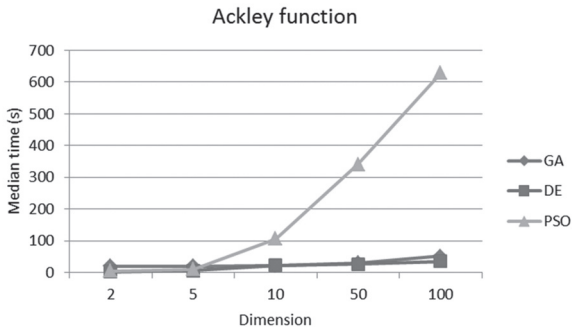


Fig. 6. Runtime performance of heuristic algorithms for 100 trials for the Ackley function (similar results are obtained for the Rastrigin and Rosenbrock functions).

5. CONCLUSION

In this paper, three heuristic algorithms are systematically analyzed and tested in detail for high dimensional real-parameter optimization problems. These algorithms are GA, DE and PSO. An overview of the implemented algorithms is provided. The algorithms are tested on three standard optimization functions, namely, the Rastrigin, Rosenbrock and Ackley functions. For lower dimensional problems, i.e., problems involving at most 10 parameters, all three algorithms had comparable results. However, for higher dimensional problems, PSO outperformed the other algorithms in terms of accuracy but had very poor runtime performance. On the other hand, the runtime performances of GA and DE did not change much with an increase in problem dimensionality.

6. REFERENCES

[1] T. Weise, Global Optimization Algorithms: Theory and Application; http://www.it-weise.de/projects/book.pdf, (accessed: 20 August 2014)

[2] R. Hooke, T. A. Jeeves, "'Direct Search' Solution of Numerical and Statistical Problems", Journal of the Association for Computing Machinery (ACM) Vol. 8(2), pp. 212–229, 1961.

[3] J. A. Nelder, R. Mead, "A Simplex Method for Function Minimization", Computer Journal, Vol. 7, pp. 308–313, 1965.

[4] A. H. Land, A. G. Doig, "An Automatic Method of Solving Discrete Programming Problems", Econometrica, Vol. 28(3), pp. 497–520, 1960.

[5] J. H. Holland, Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. The University of Michigan Press, Ann Arbor, 1975. Reprinted by MIT Press, NetLibrary, Inc., April 1992.

[6] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley Longman Publishing Co. Inc., Boston, MA, USA, 1989.

[7] R. Storn, "On the Usage of Differential Evolution for Function Optimization", Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), Berkeley, CA, USA, 19-22 June 1996, pp. 519–523.

[8] R. Storn, K. Price, "Differential Evolution - a Simple and Efficient Heuristic for Global optimization over Continuous Spaces", Journal of Global Optimization, Vol. 11, pp. 341–359, 1997.

[9] J. Kennedy, R. Eberhart, "Particle Swarm Optimization", Proceedings of IEEE International Conference on Neural Networks, Perth, WA, USA, 27 November – 1 December 1995, Vol. 4, pp. 1942–1948.

[10] Y. Shi, R.C. Eberhart, "A Modified Particle Swarm Optimizer", Proceedings of IEEE International Conference on Evolutionary Computation, Anchorage, AK, USA, 4-9 May, 1998, pp. 69–73.

[11] M. Dorigo, "Optimization, Learning and Natural Algorithms", PhD thesis, Politecnico di Milano, Italy, 1992.

[12] M. Dorigo, V. Maniezzo, A. Colorni, "The ant system: Optimization by a Colony of Cooperating Agents", IEEE Transactions on Systems, Man, and Cybernetics Part B: Cybernetics, Vol. 26(1), pp. 29–41, 1996.

[13] B. Xing, W. – J. Gao, "Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms", Intelligence Systems Reference Library, Vol. 62, Springer International Publishing, 2014.

[14] S. Luke, "Essentials of Metaheuristics, Lulu, 2nd edition"; http://cs.gmu.edu/~sean/book/metaheuristics/, (accessed: 20 August 2014)

[15] O. Roeva (ed.), "Real-World Applications of Genetic Algorithms", InTech Ltd., Rijeka, Croatia, February 2012.

[16] S.P. Mendes, J.A Gómez-Pulido, M.A.V., Rodríguez, M.D Jaraiz simon, J.M. Sánchez-Pérez, "A Differential Evolution Based Algorithm to Optimize the Radio Network Design Problem", Second IEEE International Conference on Science and Grid Comput-

ing, Amsterdam, The Netherlands, December 2006, p.119.

[17] N. Forghani, M. Forouzanfar, A. Eftekhari, S. Mohammad-Moradi, M. Teshnehlab, "Application of Particle Swarm Optimization in Accurate Segmentation of Brain MR Images", Particle Swarm Optimization, Lazinica, A. (ed.). InTech, January 2009. pp. 203-222.

[18] M. Srinivas, L. Patnaik, "Adaptive Probabilities of Crossover and Mutation in Genetic Algorithms", IEEE Transactions on System, Man and Cybernetics, Vol. 24(4), pp. 656–667, 1994.

[19] J. Zhang, H. Chung, W.L. Lo, "Clustering-Based Adaptive Crossover and Mutation Probabilities for Genetic Algorithms", IEEE Transactions on Evolutionary Computation, Vol. 11(3), pp. 326–335, 2007.

[20] J. E. Baker, "Reducing Bias and Inefficiency in the Selection Algorithm", Genetic Algorithms and Their Applications, Proceedings of the Second International Conference on Genetic Algorithms (ICGA), 1987, pp. 14–21.

[21] Z. Yang, K. Tang, X. Yao, "Differential evolution for high-dimensional function optimization", IEEE Congress on Evolutionary Computation, CEC 2007, Singapore, 25-28 September, 2007, pp. 3523-3530.

[22] S. Das, P.N. Suganthan, "Differential Evolution: A Survey of the State-of-the-Art", IEEE Transactions on Evolutionary Computation, Vol. 15(1), 2011, pp. 4-31.

[23] J. Brest, S. Greiner, B. Boskovic, M. Mernik, V. Zumer, "Self-Adapting Control Parameters in Differential Evolution: a Comparative Study on Numerical Benchmark problems", IEEE Transactions on Evolutionary Computation, Vol. 10(6), pp. 646–657, 2006.

[24] J. Zhang, A.C. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive", IEEE Transactions on Evolutionary Computation, Vol. 13(5), pp. 945–958, 2009.

[25] A.K. Qin, P.N. Suganthan, "Self-Adaptive Differential Evolution Algorithm for Numerical Optimization", IEEE Congress on Evolutionary Computation, CEC 2005, Edinburgh, Scotland, 2-5 September, pp. 1785–1791, 2005.

[26] M. Zambrano-Bigiarini, M. Clerc, R. Rojas, "Standard Particle Swarm Optimisation 2011 at CEC-2013: A baseline for future PSO improvements", IEEE Congress on Evolutionary Computation (CEC), Cancun, Mexico, 20-23 June 2013, pp. 2337 -2344.

[27] J. J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, "Comprehensive Learning Particle Swarm Optimizer for Global Optimization of Multimodal Functions", IEEE Transactions on Evolutionary Computation, Vol. 10(3), pp. 281–295, 2006.

[28] M. Clerc, "Standard Particle Swarm Optimisation, Particle Swarm Central, Technical Report, 2012"; http://clerc.maurice.free.fr/pso/SPSO descriptions.pdf, (accessed: 24 August 2014)

[29] J. Kennedy, "Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance", Proceedings of the 1999 Congress on Evolutionary Computation, Washington, DC, USA, July 6-9, 1999, Vol. 3. p. 1938.

[30] J. Kennedy, R. Eberhart, Y. Shi, Swarm Intelligence, Morgan Kaufmann Publishers Inc. San Francisco, CA, USA, 2001.

[31] J. Kennedy, R. Mendes, "Population Structure and Particle Swarm Performance", Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02, Honolulu, HI, USA, 12-17 May, 2002, pp. 1671–1676.

[32] J. J. Liang, B-Y. Qu, P. N. Suganthan, A. G. Hernández-Díaz, "Problem Definitions and Evaluation Criteria for the CEC 2013 Special Session and Competition on Real-Parameter Optimization, Technical Report 201212", Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and "Technical Report", Nanyang Technological University, Singapore, January 2013.

[33] J. J. Liang, B-Y. Qu, P. N. Suganthan, "Problem Definitions and Evaluation Criteria for the CEC 2014 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization, Technical Report 201311", Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and "Technical Report", Nanyang Technological University, Singapore, December 2013.

[34] V. Picheny, T. Wagner, D. Ginsbourger, "A Benchmark of Kriging-Based Infill Criteria for Noisy optimization", Structural and Multidisciplinary Optimization, Vol. 48(3), pp. 607-626, 2013.