Paper

Benchmarking Procedures for Continuous Optimization Algorithms

Karol Opara[a] and Jarosław Arabas[b]

[a] Systems Research Institute, Polish Academy of Sciences, Warsaw, Poland
[b] Institute of Electronic Systems, Warsaw University of Technology, Warsaw, Poland

Abstract—Reliable comparison of optimization algorithms requires the use of specialized benchmarking procedures. This paper highlights motivations which influence their structure, discusses evaluation criteria of algorithms, typical ways of presenting and interpreting results as well as related statistical procedures. Discussions are based on examples from CEC and BBOB benchmarks. Moreover, attention is drawn to these features of comparison procedures, which make them susceptible to manipulation. In particular, novel application of the weak axiom of revealed preferences to the field of benchmarking shows why it may be misleading to assess algorithms on basis of their ranks for each of test problems. Additionally, an idea is presented of developing massively parallel implementation of benchmarks. Not only would this provide faster computation but also open the door to improving reliability of benchmarking procedures and promoting research into parallel implementations of optimization algorithms.

Keywords—black-box optimization, comparing optimization algorithms, evaluation criteria, parallel computing.

1. Introduction

The question, which optimization algorithm performs "best", seems to be of both practical and scientific importance. However, the notion of the "best" algorithm is not well defined and, more importantly, there are little theoretical clues for its choice. For these reasons, in practice optimization algorithms are compared using specialized sets of test problems under appropriate evaluation criteria. Before going into details it may beneficial to briefly remind some of the crucial notions in the field of benchmarking.

Continuous optimization problem involve finding an argument x_{opt} minimizing a certain objective function $f: D \to \mathbb{R}$, where $D \in \mathbb{R}^n$.

$$x_{opt} = \arg\min_{x \in D} f(x). \tag{1}$$

Domain of the objective function D is called feasible set and it is often a hypercube $D = [l_1, u_1] \times [l_2, u_2] \times ... \times [l_n, u_n]$. The minimal function value is denoted by $f_{opt} = f(x_{opt})$. Optimization algorithm is a method of producing a series of points $x_1, x_2, ..., x_m \in D$. The best function value reached by an optimizer is denoted by $f_{best} = \min_{i \in \{1,2,...,m\}} f(x_i)$, while difference $f_{best} - f_{opt}$ is called optimization error.

The "best" optimization algorithm solves a problem accurately (effectively) and fast (efficiently). A combination of these two characteristics is referred to as performance, however formal definition of this notion is not clear. Moreover, all optimization methods yield nondeterministic results. This is due to two factors: random initialization of an algorithm, e.g., choice of start point (or start population) and stochastic nature of many of the state of the art optimization methods, most notably evolutionary and similar nature-inspired algorithms. Therefore, empirical estimation of any measure of performance requires many independent restarts of an algorithm.

Comparison between optimization methods is usually performed by means of running simulations for specially designed sets of optimization problems. Such process, called benchmarking, is not a trivial task and requires both specialized skills and knowledge. This paper provides an overview of benchmarking procedures discussing the major issues in this field: theoretical grounds of algorithm comparison, available benchmarks, evaluation criteria, interpretation of results and testing their statistical significance. The aim of this contribution is to promote the use of systematic procedures for comparison of algorithms and to highlight some of the most important aspects of benchmarking. Furthermore, two novel concepts are presented: criticism of rank-based comparison methods and an idea of parallel implementation of test problems, which may become a qualitative improvement to the benchmarking procedures.

1.1. Fair Comparison of Algorithms

Benchmarking emerges from the need of fair comparison of optimization algorithms. Choosing an algorithm which would perform "best" on a new function whose characteristic is unknown can be done through measuring performance over a wide range of test problems and aggregating the obtained results. A test problem consists of a test function accompanied by some additional criteria such as the feasible set, initialization area, stopping conditions etc. Benchmarking yields meaningful results only when competing algorithms are compared on the same test problems with the same performance criteria.

Comparing algorithms can be performed quite easily with the use of some readily available benchmarks. This approach has some major advantages:

- There is no need to develop testbeds and performance criteria, which saves work and protects from possible methodological mistakes.

- Using benchmarks does not require special effort, since their implementations are usually available in a few programming languages. It is enough to link one of them to an optimizer and assign processor time.
- Comparison with other algorithms, which were previously tested on a benchmark, is possible without repeating those experiments.
- Results may be postprocessed and their presentation can be standardized.

1.2. Benchmarking and Free Lunches

Although benchmarking is generally approved among scientists and practitioners, its sense is sometimes criticized on the basis of the *no free lunch theorem* introduced and formalized in papers [1], [2]. The theorem states that for any algorithm, any performance gain over one class of problems is offset by performance loss over another class. Therefore, no algorithm can be be considered consistently "best" for all possible problems. This statement highlights limitations to benchmarking, as there is no "best" general purpose optimizer. However, for a fixed class of problems, there are methods which yield better results than others.
Performance of optimization algorithms depends on exploiting problem regularities [3] such as symmetry or convexity of objective function. There is a large variety of "typical" optimization problems, which have random character with no structure [4]. On the other hand, typically solved problems usually reveal some regularities and therefore constitute only a narrow subset of all problems. For instance, the power of the set of all functions $f : \mathbb{R} \to \mathbb{R}$ equals $2^{\mathfrak{c}}$, while the power of continuous functions $f : \mathbb{R} \to \mathbb{R}$ is strictly lower and equals continuum $\mathfrak{c}$. The latter follows from the fact that a continuous function is uniquely defined by its restriction to rational numbers $f : \mathbb{Q} \to \mathbb{R}$ [5]. Thus, benchmarking can be used to find algorithms outperforming others on a narrow but quite important class of continuous functions. The resulting low performance for a wide class of discontinuous functions often bears little practical significance.

1.3. What do Benchmarks Measure?

Benchmarking is a way of evaluating an algorithm's ability to exploit regularities of certain class of problems. It is therefore important to remember that test problems and evaluation criteria were developed in order to address issues such as [6]:

- high cost of single function evaluation,
- high dimensionality of search space,
- linear and nonlinear constraints,
- high conditioning of a function,
- noisiness of a function,
- large number of local optima (thousands),
- linearly non-separable functions,
- global optimum located on a boundary of feasible set.

Some of these issues can be observed when looking at plots of two-dimensional variants of test problems from CEC'05 benchmark [7], Fig. 1. Most of the test functions in CEC and BBOB families of benchmarks are typical optimization problems known in the literature. They become, however,

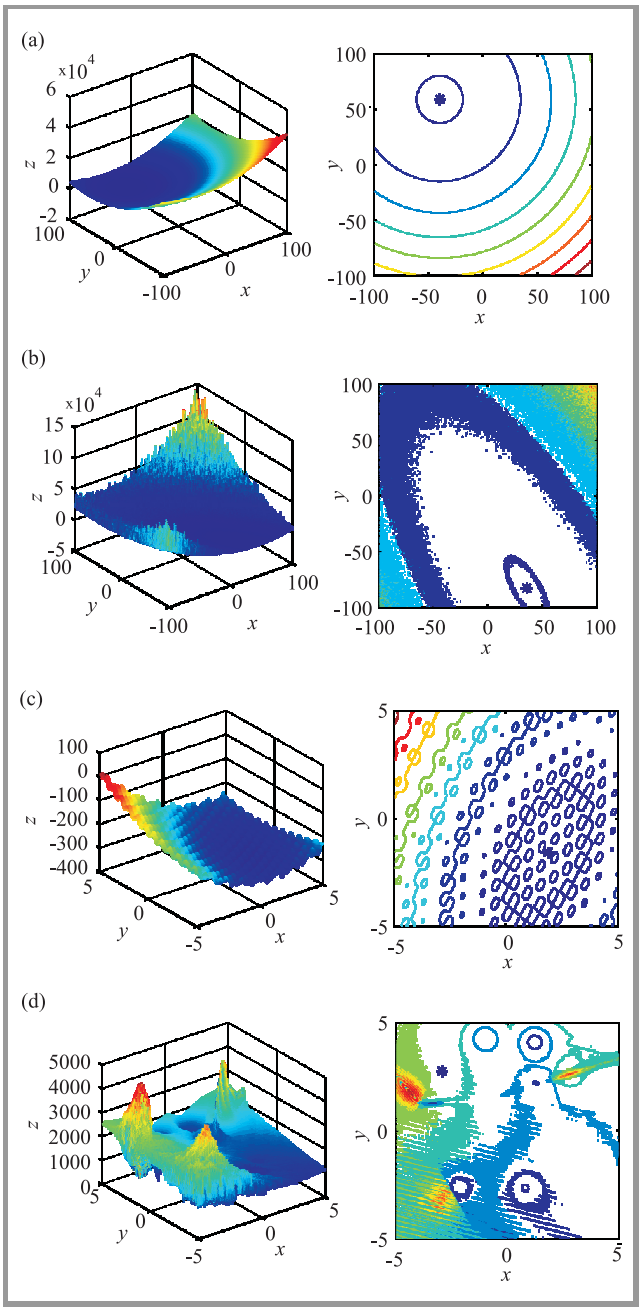

Fig. 1. Two-dimensional variants of CEC'05 benchmark problems: (a) sphere, (b) noisy ellipsoid, (c) Rastrigin, (d) hybrid. Each function is plotted within its feasible set, asterisks denote the location of global minimum.

subject to transformations such as shifting $f'(x) = f(x+c)$, adding a constant $f'(x) = f(x) + c$ and rotating. This is done in order to promote algorithms which are invariant to such changes of a coordinate system of the objective function. By this means, results obtained for a single test problem can be extended to whole class of problems.
Evaluation of algorithms is a two-step procedure. First, the algorithm's performance is measured for each test problem independently. Then, results obtained for each problem are aggregated to form a more general picture. Discussion of either of these steps is provided in Sections 2 and 3.
Benchmarking procedures for continuous domain optimizers are developed along CEC and GECCO conferences for special sessions devoted to comparing performance of optimization algorithms. Availability of these results facilitates comparing new optimization methods to the state of the art ones. CEC benchmarks cover a wide range of specialized optimization problems, Table 1. GECCO benchmarks are called BBOB (Benchmarking Black-Box Optimizers) and specialize in single-objective and noisy problems. BBOB is a carefully-developed platform providing motivated test problems, experimental setup and postprocessing of results [8].

Table 1
Scope of real-parameter benchmarks

Problem kind	Benchmarks
Single-objective	CEC'05, BBOB'09, BBOB'10
Constrained	CEC'06, CEC'10
Multi-objective	CEC'07, CEC'09
Large scale	CEC'08, CEC'10
Dynamic	CEC'09
Noisy	BBOB'09, BBOB'10
Real world	CEC'11

2. Measuring Performance for a Single Problem

There are two main ways for measuring the algorithm's performance for a single problem. The fixed cost approach consists of checking the final optimization error $f_{best} - f_{opt}$ after running the algorithm for a certain period of time. The fixed target approach consists in measuring time necessary to find a solution at an accuracy target Δf_t. In order to compare algorithms rather than their implementations and hardware used to run benchmark, computing time is expressed as the number of objective function evaluations (FEs), which is a standard approach in the literature on optimization.

2.1. Fixed Cost

Figure 2 shows convergence curves, i.e., the optimization error as a function of computing time, for four independent runs of an algorithm. Fixed cost approach can be illustrated with a vertical cut [8]. A set of error values

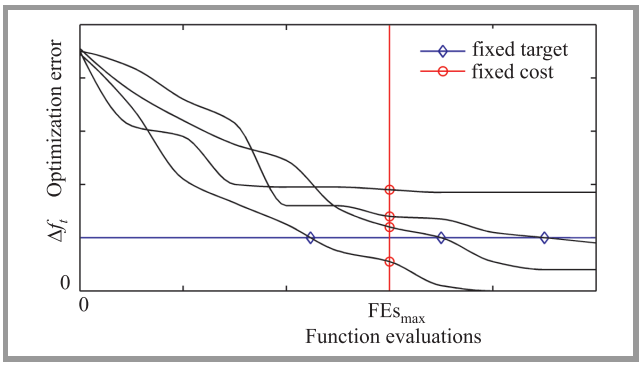

Fig. 2. Fixed cost and fixed target stopping criteria [8].

at a fixed cost can thus serve for comparisons. An algorithm A is better than algorithm B if it yields lower error values. Comparison is done on ordinal scale, which gives qualitative information (which algorithm is better?). However, no quantitative information (how much better it is?) is provided, as it is not clear how much more difficult is to reach a smaller error value [8].

2.2. Fixed target

Instead of fixing computing cost and comparing final optimization errors one can fix the desired error value Δf_t and compare the average runtime of algorithms required to reach it. Due to nondeterministic performance of algorithms, comparisons are reliable after aggregating the runtime values from multiple runs, for instance, by estimating their expected value. Again, computing time is measured as the number of function evaluations. The fixed target approach can be more precisely stated as comparing the estimates of the expected runtime needed for each algorithm to reach optimization error not greater than required, i.e., to satisfy condition $f_{best} - f_{opt} \leq \Delta f_t$. This criterion is illustrated in Fig. 2 with a horizontal line. Expected runtime values for different algorithms can be compared on the interval scale: it is possible to quantitatively state how much faster is one algorithm than another. This facilitates interpretation of results and it is the reason for choosing the fixed target approach for BBOB benchmarks [8].
There are, however, two problems with calculating the expected runtime. First, the required accuracy Δf_t must be somewhat arbitrarily chosen. This can be partially overcome by setting several accuracy targets. Second, expected runtime estimation is somewhat problematic. Some algorithm runs may fail to solve optimization problem at all. This may happen, for instance, due to premature convergence to a local optimum [9]. Consequently, the stopping criterion can not be solely dependent on the accuracy but it must also involve some "safety breaks", e.g., the maximum cost of simulation. Then, a single run of an algorithm is called successful if it reaches accuracy target within a given time limit. Without such limit, the number of successful runs in Fig. 2 would equal three. The presence of additional stopping criterion fixing maximal cost

reduces this number to one, as three runs stop due to exceeding time limit rather than reaching solution accurate enough.

Expected runtime $\hat{E}(RT(\Delta f_t))$ for a given accuracy target Δf_t is estimated as the sum of the expected number of function evaluations for one successful run $\hat{E}(N^s_{eval})$ and the maximum cost $\hat{E}(N^u_{eval})$ multiplied by odds against a successful run $(1-\hat{p})/\hat{p}$. Value $\hat{p}$ is an estimate for probability of solving a problem within a single run, in other words the fraction of successful algorithm runs [8].

$$\hat{E}\left(RT(\Delta f_t)\right) = \hat{E}(N^s_{eval}) + \frac{1-\hat{p}_s}{\hat{p}_s}\hat{E}(N^u_{eval}) \qquad (2)$$

Denoting the number of successful runs by $\#s$ and unsuccessful by $\#u$, one can note that $\hat{p} = \#s/(\#s+\#u)$. If a successful run is terminated right after meeting accuracy target then the above formula can be transformed to:

$$\hat{E}\left(RT(\Delta f_t)\right) = \frac{\#s \cdot \hat{E}(N^s_{eval}) + \#u \cdot \hat{E}(N^u_{eval})}{\#s} = \frac{N^t}{\#s}, \qquad (3)$$

where N^t denotes the total number of FEs within all algorithm runs. This estimator is however strongly dependent on the choice of maximal advisable cost in case of not meeting the target accuracy [8]. If the vertical line if Fig. 2 was shifted to the right, the number of successful runs $\#s$ would increase to two and then to three changing the value of the expected runtime estimate Eq. (3).

2.3. Statistical Analyses of Results

To check whether differences between algorithm performance are significant rather than observed purely by coincidence results should be subject to statistical testing. In papers comparing continuous optimization algorithms Student's t-test seems to be a common choice. This is a parametric test, as it relies on an assumption that sample (here final error values from several runs) is normally distributed. Alternatively, one can use non-parametric tests, which do not assume any distribution of a sample. An extensive study [10] conducted on CEC'05 results showed that conditions for safe use of parametric tests are usually not fulfilled, nevertheless results of parametric and non-parametric analysis are quite similar.

The use of non-parametric tests is encouraged in a multiple-problem analysis. In paper [10] the following non-parametric tests are suggested for analysis of optimization algorithms: Friedman, Iman-Davenport, Bonferroni-Dunn, Holm, Hochberg, and Wilcoxon. In case of performing pairwise comparisons of many algorithms the power of statistical tests decreases, as control of Family Wise Error Rate (FWER) is lost [10]. To compensate for that one should decrease the statistical significance α below typical value of 0.05 by using test variants designed for multiple comparisons.

Finally, it is worthwhile to examine results not only with statistics but also in a wider context. Superiority of final error rate of $10^{-40} \pm 10^{-41}$ over $10^{-15} \pm 10^{-16}$ has neither theoretical nor practical value in case of spherical function and double precision numbers. An old proverb says that difference is a difference only when it makes a difference.

3. Aggregation of Performance Measures

3.1. Ranking of Algorithms

Comparison of optimization algorithms A and B is a multiobjective task, as it is based on certain performance measure P_{F_i} for each of several test problems $F_1, F_2, \ldots, F_k$. The algorithms can be naturally partially ordered with Pareto improvement relation

$$A \preceq B \equiv (\forall i \in \{1,2,...,k\})\ P^A_{F_i} \geq P^B_{F_i}. \qquad (4)$$

Algorithm A is here considered better than algorithm B providing it yields better results for each test problem. This is, however, only partial ordering, as some pairs of algorithms are not comparable, e.g., when one of them performs better for some problems but worse for others. Hence, the question "which algorithm performs best?" is not always answered. For this reason, performance over the whole test set is often aggregated into a single number, which allows creating a linear ordering (ranking) among competing algorithms. Such aggregation method must be carefully chosen to ensure fairness of the comparison.

Issues such as designing decision rules to choose the best alternative out of a certain set have been widely investigated by economists and mathematicians in such fields as theory of the public sector. Some of these results can be applied to analyze ranking methods of optimization algorithms. The Weak Axiom of Revealed Preferences (WARP) [11], plays an important role among them. Its definition is equivalent to simultaneous satisfaction of the following conditions known as α rule and β rule:

- α if an algorithm performs best in a certain set of algorithms then it is also the best in its subset (of course if it is available in the subset),
- β if two algorithms are equally good, best algorithms, then in every superset either both are the best or none.

If WARP is not met, the results of comparison can be influenced by such means as shortlisting, pairwise comparisons or introducing other algorithms to increase relative advantages.

Multi-problem aggregation methods are based on results achieved by algorithms for sets of single problems. Some of them do not satisfy the β rule, which makes them susceptible to manipulation. This can be illustrated with the following example. Suppose that two algorithms, A and B, were tested on a set of four problems F_i, $i \in \{1,2,3,4\}$ and aggregated by comparing the mean rank achieved for all test problems. The ranks of final optimization errors

are shown in the upper part of Table 2. The mean rank for both algorithms equals 1.5, therefore one may conclude that A and B are equally good. Suppose now that another algorithm C had been added to this comparison. The performance of C is slightly worse than of A, therefore for each test problem C is ranked one notch lower than A, Table 2. The mean rank of algorithm A remains 1.5 while

Table 2
If mean rank is used as performance criterion, introduction of irrelevant alternative C changes preference between A and B

$\{A, B\}$		Test problem				Mean rank
		F_1	F_2	F_3	F_4	
Rank	1	A	A	B	B	$A = 1.5$
	2	B	B	A	A	$B = 1.5$

$\{A, B, C\}$		Test problem				Mean rank
		F_1	F_2	F_3	F_4	
Rank	1	A	A	B	B	$A = 1.5$
	2	C	C	A	A	$B = 2.0$
	3	B	B	C	C	$C = 2.5$

algorithm B now performs poorer, since its mean rank equals 2. This shows that introduction of an irrelevant alternative C changed the preference between algorithms A and B in favor of A. Therefore, competitions, for which mean (median, etc.) rank is used as multi-problem aggregation criterion are susceptible to manipulations. On the other hand, WARP is satisfied whenever the performance measure of an algorithm is independent from other, competing algorithms.

3.2. Empirical Runtime Distribution

Empirical runtime distribution is a way of aggregating and comparing performance of different algorithms measured for many runs over a set of test problems. It can be visualized by plotting the proportion of solved problems against the expected runtime of an algorithm. The general idea behind this plot can be illustrated with a following example. Consider benchmarking an algorithm on three test functions. Expected runtime values, over many independent runs, are given in Table 3 for fixed targets Δf_1 and Δf_2. Dotted graph in Fig. 3 presents runtime distribution for required accuracy Δf_1. Its shape represents an

Table 3
Estimated runtime values for three test problems and two accuracy targets

		Test function		
		F_1	F_2	F_3
Target	Δf_1	$3 \cdot 10^2$	$1 \cdot 10^3$	$4 \cdot 10^4$
	Δf_2	$2 \cdot 10^3$	$5 \cdot 10^3$	$7 \cdot 10^4$

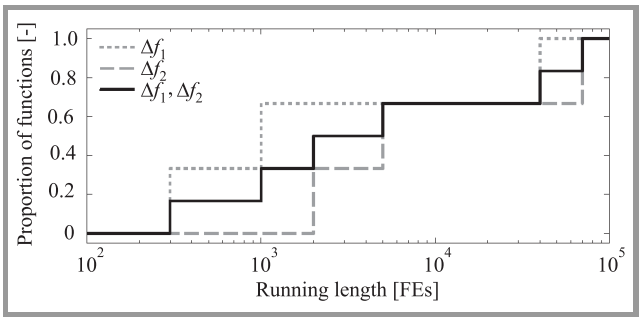

Fig. 3. Construction of empirical runtime distribution plots.

answer to question how many problems can one expect to solve for a given FEs budget? Below 300 FEs no problems can be solved, between 300 and 1000 only one (which is 33% of all problems), from 1000 to 40000 two (66%) and over 40000 three (100%). Dashed graph represents analogous data for more strict accuracy criterion Δf_2. Both lines can be aggregated by treating each of the six pairs $(F, \Delta f) \in \{F_1, F_2, F_3\} \times \{\Delta f_1, \Delta f_2\}$ as independent problems and using analogous manner to draw a new graph plotted in Fig. 3 with solid black line. Such aggregation among required accuracy targets is a way to overcome arbitrarily in choosing their levels Δf.

4. CEC and BBOB Benchmarks

Various versions of rank-based aggregation are used in practice. During CEC'06 competition ranking was based on three statistics of performance on test problems: feasible rate, success rate and expected runtime (also known as success performance) [12]. During CEC'07, ranks were assigned using two different multiobjective performance measures: R indicator and hypervolume difference to a reference set which resulted in two, quite similar rankings [13]. During CEC'08 and '09 each team was ranking others and the positions were averaged [14], [15]. In CEC'10 the Formula 1 point system was applied to 300 optimization problems and the algorithm with the highest score sum won [16].

In BBOB benchmarks [17] no ranking is created, as aggregation of results is based on empirical runtime distribution. Figure 4 presents the results of the BBOB'09 competition [17]. The horizontal axis shows runtime measured in function evaluations divided by dimension of search space n, which in this case equals 10. The vertical axis depicts the proportion of problems solved on all functions with accuracy targets $\Delta f_t \in \{10^{1.8}, 10^{1.6}, 10^{1.4}, \ldots, 10^{-8}\}$. Figure 4 was created in a slightly more complicated fashion than Fig. 3. Instead of estimating expected runtime values, for each pair $(F, \Delta f)$, 100 instances of simulated runtime were created by drawing algorithm runs at random with replacement until a successful run is found. Runtime instance is then computed as a sum of function evaluations from all runs drawn [17].

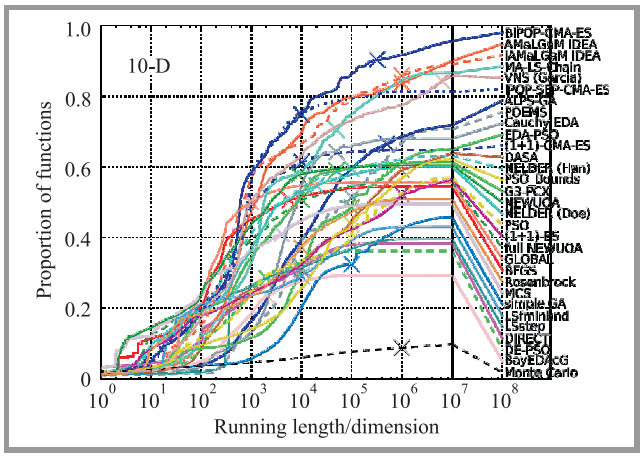

Fig. 4. Empirical runtime plots for ten-dimensional BBOB'09 benchmark for 31 algorithms [17].

Each algorithm was restarted many times and the cross indicates the maximum number of function evaluations. It is suggested [17] that a decline in steepness right after the cross (e.g., for IPOP-SEP-CMA-ES) may indicate that the maximum number of function evaluations should have been chosen larger, while a steep increase right after the cross (e.g., for simple GA) could be a sign that a restart should have been invoked earlier.

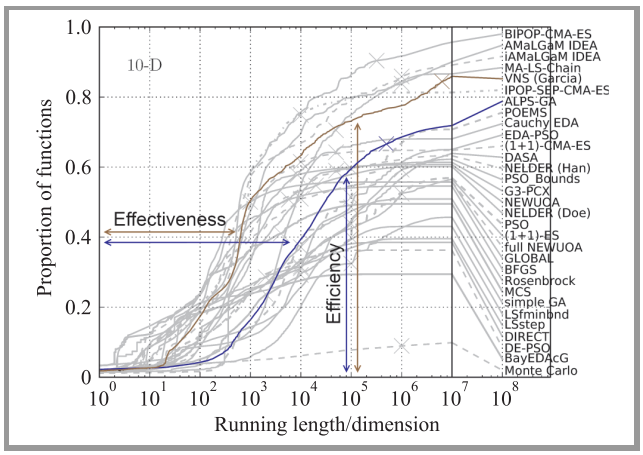

Fig. 5. Comparison of two algorithms VNS and ALPS-GA in terms of effectiveness (vertical distance between curves) and efficiency (horizontal distance).

Expected runtime plots provide significant interpretation possibilities, which are illustrated in Fig. 5 with an example of comparison of two algorithms, VNS and ALPS-GA. Within the budget of $10^5 \cdot n$ FEs, where n denotes search space dimension, ALPS-GA solves 60% of problems, while VNS over 70%, which is indicated by vertical arrows. Difference and ratio between these values show how much more efficient is one algorithm than another. On the other hand, horizontal arrows indicate how much computing time is required by each algorithm to solve a given ratio of test problems (here 40%). For VNS it is approximately $400 \cdot n$, while for ALPS-GA this time equals $10000 \cdot n$. One can therefore conclude that VNS is 25 times faster than ALPS-GA in solving 40% of problems. The area under a graph of an algorithm is, according to authors of the benchmark [17], arguably the most useful aggregated performance measure (averaged on a log scale). It might be therefore a good criterion for parameter tuning, for example, by means of metaoptimization.

5. Benchmarking and Parallel Computing

5.1. Parallel Implementation of Benchmarks

Both CEC and BBOB benchmarks are recognized and mature tools for measuring performance of optimization algorithms. Moreover, they can be easily parallelized. We believe that it would be beneficial to develop an implementation of these benchmarks exploiting massively parallel general purpose graphical processing units (GPUs). Availability of fast and scalable benchmark implementation would have significant consequences:

- GPU-based metaoptimizer could be developed for each benchmark. Fair comparison of algorithms requires choice of an "optimal" parameter set for each of them according to evaluation criteria. Currently, some optimization methods may not be tuned adequately, which decreases their performance. This might prove unjustified conclusions about superiority of some algorithms over others. To solve this problem, one can encourage all participating teams to perform metaoptimization, whose huge computing cost could be balanced by massive parallelism of GPUs.
- Additional evaluation criteria promoting research into the choice of the best parallelization methods of optimization algorithms could be introduced.
- The number of test functions could be increased. This would decrease statistical uncertainty, which is especially important in case of multiple comparison analysis, where significance α must be decreased to control FWER.
- Algorithms could be tested for larger FEs budgets.

5.2. Evaluation criteria for parallel algorithms

Evaluation criteria of optimization algorithms are based on the number of the objective function evaluations. This facilitates implementation independent comparisons and is practical, since evaluating the objective function is often the most time-consuming. On the other hand, many optimization algorithms can be easily parallelized by such means as simultaneous evaluation of the objective function in case of population-based methods. Therefore, the real execution

time can arguably be better described by the number of iterations rather than function evaluations required to solve a given problem.

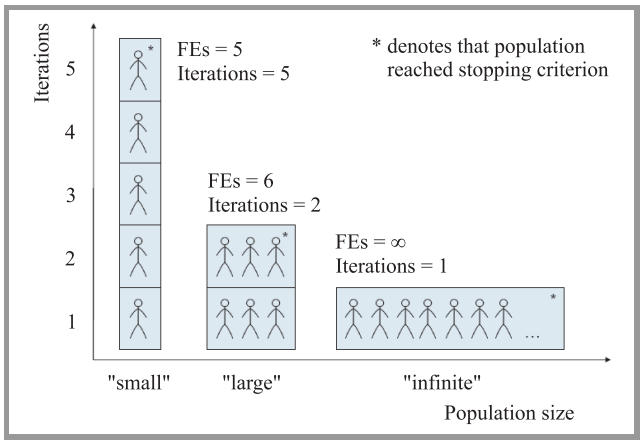

Fig. 6. Illustration of tradeoff between number of iterations and number of function evaluations in parallel computing.

Figure 6 illustrates a possible tradeoff between the number of function evaluations and the number of iterations. In this example, a population-based algorithm running with single individual requires 5 FEs and 5 iterations to solve a problem. A three-individual variant requires 6 FEs but only 2 iterations, hence it is slower for the sequential computation but faster for the parallel one. This feature would not be noticed with a traditional approach based on function evaluations. Direct adaptation of iteration counting might be, however, confusing, since for an infinite population any problem can be solved in the first iteration. For this reason, a number of iterations for some (arbitrarily chosen) maximal number of parallel processing units $\#proc$ seems to be a better criterion. It is worth noting that in case of full use of all processing units the number of iterations $\#iter$ is a product of number of function evaluations $\#FEs$ and processing units

$$\#iter = \#FEs \cdot \#proc\,. \tag{5}$$

Such criterion could additionally encourage researchers to look for the most appropriate parallelization models for their algorithms. This issue is illustrated in Fig. 7. In this simple example a test problem was solved using four processing units. For a four-individual population it took 4 iterations, for two two-individual populations it took 3 and 4 iterations and for four single-individual populations it took from 4 up to over 7 iterations. In case of many parallel instances one can stop computation after the problem is solved by any of them. Consequently, only these iterations (and function evaluations) should be taken for comparison, which are shaded in Fig. 7. In this example, problem was solved fastest when there were two populations each using two processing units. Many other parallelism models are possible and their choice is an interesting and algorithm-specific question.

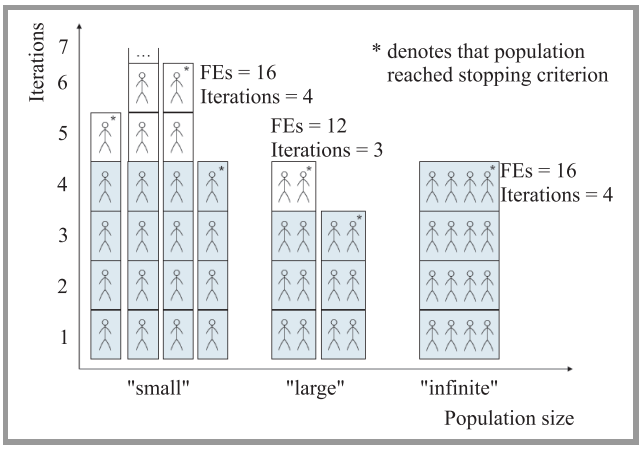

Fig. 7. Performance of three parallel optimization models for four processing units.

The proposed evaluation criteria are simplified, as they disregard some limitations of parallelism such as the Amdahl's law or memory transfer bottlenecks. For instance, selection operators in evolutionary algorithms require synchronization within population (or subpopulations in case of tournament selection). Nevertheless, popularization and further development of appropriate benchmarks seem to be an important issue for the whole community developing and using optimization algorithms.

6. Summary

This paper provides an outlook on procedures of measuring performance of optimization algorithms, emphasizing a need for standard and systematic approach in this field. Attention is paid to motivations and intuitive premises behind benchmarking as well as evaluation and comparison criteria for both single- and multiple-problem analyses. A brief summary of state of the art benchmarks and interpretation of their results is provided. Fairness of comparisons is discussed with respect to rank-based aggregation in multiple-problem analysis and testing of statistical significance of benchmarking results. Finally, a discussion of parallel implementation of test sets is provided. Not only can this make benchmarking more effective but also promote further research into parallelization schemes. Moreover, reliability of comparisons of algorithms can be improved by implementing a parallel and thus fast metaoptimizer.

References

[1] M. Köppen, D. H. Wolpert, and W. G. Macready, "Remarks on a recent paper on the "no free lunch" theorems", *IEEE Trans. Evol. Comput.*, vol. 5, no. 3, pp. 295–296, 2001.

[2] D. Wolpert and W. Macready, "No free lunch theorems for optimization", *IEEE Trans. Evolution. Comput.*, vol. 1, no. 1, pp. 67–82, 1997.

[3] T. Weise, M. Zapf, R. Chiongand, and A. Nebro, "Why is optimization difficult?", in *Nature-Inspired Algorithms for Optimisation*, R. Chiong, Ed., vol. 193 of *Studies in Computational Intelligence*. Springer, 2009, pp. 1–50.

[4] T. English, "Optimization is easy and learning is hard in the typical function", in *Proc. 2000 Congr. Evol. Comput. CEC 2000*, San Diego, CA, USA 2000, pp. 924–931.

[5] J. Kraszewski, *Wstęp do matematyki*. Warszawa: WNT, 2007 (in Polish).

[6] P. Suganthan, "Special session on real-parameter optimization at CEC 2005. Summary of feedbacks received from potential participants", 2005, access: May 2010 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[7] P. Suganthan *et al.*, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization", Tech. Rep., Nanyang Technological University, Singapore and KanGAL Report Number 2005005, 2005, access June 2010 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[8] N. Hansen, A. Auger, S. Finck, and R. Ros, "Real-parameter black-box optimization benchmarking 2009: Experimental setup", Tech. Rep., INRIA, 2009, access: July 2010 [Online]. Available: http://coco.gforge.inria.fr/

[9] J. Arabas, *Wykłady z Algorytmów Ewolucyjnych*. Warszawa: WNT, 2004 (in Polish).

[10] S. García, D. Molina, M. Lozano, and F. Herrera, "A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC 2005 special session on real parameter optimization", *J. Heuristics*, vol. 15, no. 6, pp. 617–644, 2009.

[11] A. K. Sen, "Choice functions and revealed preference", *Rev. Economic Studies*, vol. 38, no. 3, pp. 307–317, 1971.

[12] J. J. Liang and P. N. Suganthan, "Comparison of results on the 2006 CEC benchmark function set", 2006, access: Sept. 2011 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[13] P. N. Suganthan, "Performance assessment on multi-objective optimization algorithms", 2007, access: Oct. 2011 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[14] K. Tang, "Summary of results on CEC'08 competition on large scale global optimization", 2008, access: Sept. 2011 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[15] Q. Zhang and P. N. Suganthan. "Final report on CEC'09 MOEA Competition", 2009, access: Sept. 2011 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[16] K. Tang, T. Weise, Z. Yang, X. Li, and P. N. Suganthan, "Results of the competition on high-dimensional global optimization at WCCI 2010", 2010, access: Sept. 2011 [Online]. Available: http://www.ntu.edu.sg/home/epnsugan/

[17] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posik, "Comparing results of 31 algorithms from the black-box optimization benchmarking bbob-2009", in *Proc. Genetic Evol. Computat. Conf. GECCO 2010*, Portland, Oregon, USA, 2010. New York: ACM, 2010, pp. 1689–1696.

Karol Opara received M.Sc. degree in Computer Science from Warsaw University of Technology in 2010 and M.Sc. degree in Quantitative Methods in Economics and Information Systems from Warsaw School of Economics in 2011. Currently, he works as an assistant in the Systems Research Institute, Polish Academy of Sciences. His research interests focus on stochastic optimization algorithms, applied statistics and economic aspects of road management.
E-mail: karol.opara@ibspan.waw.pl
Systems Research Institute
Polish Academy of Sciences
Newelska st 6
01-447 Warszawa, Poland

Jarosław Arabas – for biography, see this issue, p. 10.