Information Sciences 654 (2024) 119766

Contents lists available at ScienceDirect

Information Sciences

journal homepage: www.elsevier.com/locate/ins

To what extent evolutionary algorithms can benefit from a longer search?

Adam P. Piotrowski [a,*], Jaroslaw J. Napiorkowski [a], Agnieszka E. Piotrowska [b]

[a] Institute of Geophysics, Polish Academy of Sciences, Ks. Janusza 64, 01-452 Warsaw, Poland
[b] Faculty of Polish Studies, University of Warsaw, Krakowskie Przedmiescie 26/28, 00-927 Warsaw, Poland

ARTICLE INFO

Keywords:
Metaheuristics
Performance
Time
Differential evolution
Particle swarm optimization
Swarm intelligence

ABSTRACT

In many comparison papers Evolutionary Algorithms are stopped after a pre-defined number of function calls, which is a code-independent measure of computational time. This number of function calls is either defined by the inventors of particular benchmark problems, or set subjectively by the user for the specific problem. The question how much improvement could be achieved by Evolutionary Algorithms if they were given much more time is surprisingly rarely asked. In the present study we analyze improvements obtained by thirty Evolutionary Algorithms on four different benchmark sets, including one composed of 22 real-world problems, when the allowed number of function calls is extended ten times with respect to the values defined in the comparison criteria for the specific benchmark sets. We analyze how the prolonged search would affect the ranking of algorithms, how much improvement could be obtained by Evolutionary Algorithms in general, on what kind of problems the improvement will be achieved, and which type of algorithms would benefit most from such an extended search. We show that the improvements obtained in prolonged search are higher for real-world problems than for mathematical functions, and such improvements are mainly achieved by similar kinds of adaptive algorithms proposed relatively recently. Many metaheuristics fail to benefit from the extended search on most benchmark problems, or benefit only marginally.

1. Introduction

Metaheuristics are commonly compared on sets of carefully chosen benchmark problems that are frequently accompanied by some pre-specified rules [1]. Such rules often include the number of runs to be repeated (frequently between 25 and 51), the measure of quality (most commonly mean, median, or best performance from many runs), and the stopping criteria. There are two major ways to define when to stop metaheuristic optimization algorithms. In the first approach, commonly applied in BBOB benchmark sets [2], algorithms are stopped when the pre-defined value of the objective function is found. In such a case, the desired quality is often defined for each problem within the benchmark set, and an algorithm that finds the solution with such quality using fewer function calls is called the better one. In the second approach, used commonly in IEEE benchmarks [3] and many journal papers, algorithms search for the best possible solution during a predefined number of function calls, which is a code-independent measure of computational time. Note that the number of function calls should not be misled with the number of generations. Hence, to compare metaheuristics, often either the solution quality that is to be found (BBOB way), or the number of allowed function calls for search (IEEE way) are set fixed.

* Corresponding author.
E-mail address: adampp@igf.edu.pl (A.P. Piotrowski).

https://doi.org/10.1016/j.ins.2023.119766
Received 14 February 2023; Received in revised form 2 October 2023; Accepted 5 October 2023
Available online 7 October 2023

As fixing the number of function calls is very frequent in comparisons among metaheuristics, we follow the IEEE way in the present study. We aim to analyze what would happen if we give metaheuristics much more time than is set in the rules of major competition benchmarks.

There are two main reasons to compare metaheuristics. The first is to justify the high-quality performance of the newly proposed algorithm. One could give hundreds of references here. The second is to test different features of metaheuristics, or analyze how various rules set for comparison affect the performances. Papers addressing such issues help us understanding the general behavior of various methods and focus our attention on their specific features. Some comparison studies are theoretical [4,5], but the majority of papers are based on numerical experiments that often use specific benchmark sets and their rules. Such comparison papers shed light on various issues related to metaheuristics. However, metaheuristics are methodological tools, and the main reason for their existence is to serve end users in solving their problems.

Practical approaches and rules applied by end users are frequently different than those assumed by the inventors of metaheuristics. A noticeable example is the number of allowed function evaluations: In various practical applications of metaheuristics the values set by practitioners may differ by orders of magnitude. Unfortunately, in relatively few papers the relation between the performance of optimization algorithms and the number of allowed function calls has been addressed in more detail. Two studies [6,7] compared metaheuristics and mathematical programming methods with different computational budgets. Posik et al. [6] compared the evolutionary algorithm BIPOP-CMA against four mathematical programming methods and found that the evolutionary algorithm is superior only if the number of function calls is sufficiently large. Sergeyev et al. [7] compared five evolutionary algorithms against three mathematical programming methods on various computational budgets using 800 randomly generated problems and found complicated relations between the performances of various methods and the assumed maximum numbers of function calls. Price et al. [8] presented an interesting comparison among 38 algorithms that solve ten specific problems with 10-digit precision, regardless of the computational budget. It turned out that the algorithms developed especially for this task performed much better than the classical metaheuristics, which were developed to solve problems with a fixed number of function calls. Piotrowski et al. [9] presented a study on the relationship between the performance of various metaheuristics and the computational budget. Authors compared 33 metaheuristics on 22 real-world problems from CEC 2011 [10], using from 5000 up to 500,000 function calls. It was confirmed that the computational budget has a significant impact on the ranking of metaheuristics: Particle Swarm Optimization algorithms performed relatively better than other methods for the lowest computational budget, and specific variants of Differential Evolution and Genetic Algorithms outperformed competitors when the computational budget was large. Ezugwu et al. [11] tested 12 relatively simple variants of metaheuristics on 36 typical benchmark problems with 100,000, 300,000, and 500,000 function calls. Authors found that Differential Evolution and Ant Colony Optimization performed relatively better with a larger number of function calls, when some other nature-inspired algorithms performed relatively better when the number of function calls was low. Kazikova et al. [12] studied the relative performance of seven metaheuristics on CEC 2015 [13] and CEC 2017 [3]benchmarks of various dimensionalities under two computational budgets, set to the classical 10,000D (defined in [3]), and extended to 70,000D, where D is the problem dimensionality. Authors found that the extended computational time unevenly impacted the performances obtained on CEC 2015 and CEC 2017 problems, and the improvements achieved by various algorithms.

In general, the previous studies [7–9,11,12] showed that the performances of various metaheuristics very unevenly depend on the computational budget. Some algorithms benefit only marginally from the extended search, others - significantly. The more complex, diversity-preserving algorithms benefit from longer searches to a larger extent than simple heuristics. Such conclusions may be drawn from each referred study, irrespectively that [7,8,11] addressed mainly low-dimensional problems, when [9,12] also included higher-dimensional tests. Moreover, the conclusions seem to be valid both for mathematical functions [8,11,12], randomly generated functions [7], and real-world problems [9]. However, the previous studies were generally based on a very limited number of problems.

The present paper shares some similarities with [9,11,12], but is organized at a much larger scale. We compare 30 different algorithms on four different sets of problems: thirty 50-dimensional CEC 2014 [14], thirty 50-dimensional CEC 2017 [3], ten 15-dimensional CEC 2020 [15], and twenty-two various-dimensional real-world problems from CEC 2011 [10]. Two computational budgets are used. In the first, the classical number of function calls (CNFC), as defined by the authors of the specific benchmark sets, is used; in the second, the number of function calls is extended ten times (10CNFC), relating to the classical value. Studying the tenfold extension of the number of function calls is computationally demanding and, as far as we know, has never been performed for a large number of problems and algorithms so far. Among the thirty algorithms tested in the present study, we have included some historical variants and recent methods proposed in 2022. Tested algorithms include some classical methods published in reputable journal papers, some winners of the Competitions in Evolutionary Computations (CEC) published in conference proceedings, and some less-known emerging methods. We would like to define four main goals of the present study:

a. to show how much improvement may be obtained by metaheuristics in general when the computational budget is increased by an order of magnitude. In other words, we ask whether the gain would be worth the computational effort.
b. to determine to what extent the increased number of function calls would affect the ranking of algorithms.
c. to learn what kind of algorithms achieve more significant improvements when the search is longer: those that already performed well, or those that performed poorly under the classical budget? Those that are adaptive, or those with fixed structure? Is the performance of various algorithms more diversified, or more similar when the search is ten times longer?
d. finally, we would like to note, on what kind of problems the largest improvements will be observed: unimodal or multimodal, artificial benchmarks or real-world problems.

In this study, we analyze only the single-objective, static numerical benchmark sets and algorithms. Benchmarks aiming at multi-

objective or dynamic optimization are beyond the scope of the present study. The remaining part of the paper includes details of the methodology used: test problems, performance measures, and compared algorithms (section 2), presentation and discussion of the results (section 3), and conclusions (section 4). A brief literature review has been given above, in the Introduction section.

2. Methods

2.1. Test problems

In the present study, we analyze the improvements obtained by different metaheuristics when the computational budget is extended tenfold for four different sets of problems: 50-dimensional CEC 2014 [14], 50-dimensional CEC 2017 [3], 15-dimensional CEC 2020 [15] and real-world problems from CEC 2011 [10]. The maximum numbers of function calls were defined in the source papers that introduced these benchmarks to: $10{,}000D$ for CEC 2014 and CEC 2017 sets [3,14], 3,000,000 for the 15-dimensional version of CEC 2020 [15], and 150,000 for real-world CEC 2011 problems [10]. These numbers are called in the further text as CNFC: the classical number of function calls. The search with ten times more function calls will be called 10CNFC. Both CEC 2014 and CEC 2017 benchmarks are composed of 30 numerical problems each, which include unimodal, multimodal, and hybrid-composition mathematical functions. The CEC 2020 set comprises ten problems similar to those included in CEC 2014 or CEC 2017 sets, but with much lower dimensionality and a much higher maximum number of function calls. The CEC 2011 set fully differs from the other benchmarks, as it includes 22 real-world problems from different fields of science and engineering, with dimensionalities spanning from 1 to 216.

2.2. Performance measures

In this study, we focus on two different kinds of comparisons. Firstly, we analyze how the rankings of algorithms would differ when two computational budgets are used: CNFC and 10CNFC. Secondly, we analyze how much improvement was obtained by each algorithm, or by groups of similar algorithms, when the computational time has been extended ten times, from CNFC to 10CNFC.

We must point out that the search for CNFC and 10CNFC are independent: the longer search (10CNFC) is NOT the continuation of the runs performed for the classical computational budget (CNFC). If the CNFC is, e.g. for 50-dimensional CEC 2014 problems, set to 500,000 function calls, an algorithm may use 500,000 function calls and terminate. Runs with 10CNFC (hence $10 \cdot 500{,}000 = 5{,}000{,}000$ function calls) are started from random initialization of the population, and do not benefit from the experience obtained during the search that was performed for the CNFC. Note that various adaptive features of the algorithms related to the remaining computational time (e.g. linear inertia weight modification in some PSO algorithms, population size decrease in some L-SHADE algorithms, etc.) will, during run, be scaled differently for CNFC and 10CNFC. Because runs for classical and extended computational time are independent, the performance reached by the particular algorithm after the search with 10CNFC may be poorer than the performance obtained by the same algorithm with CNFC budget.

To compare multiple metaheuristics on many problems the averaged ranking of metaheuristics is constructed as follows. Every algorithm is run 51 times on each problem [3] from every set of benchmarks, separately with CNFC and 10CNFC computational budgets. For each problem, algorithm, and computational budget the mean objective function values from 51 runs are computed. Each algorithm receives a performance rank, (1 – means the best rank, 30 – means the worst rank) on every problem and computational budget based on the mean objective function value (as minimization problems are solved, the lower the mean objective function value, the better). If for the particular problem and computational budget, the mean objective function values of two or more algorithms differ by fewer than 10^{-8}, these algorithms are given equal rank. Finally, the ranks are averaged, separately for CNFC and 10CNFC computational budgets, over all problems within the particular benchmark set (30 for CEC 2014 and CEC 2017, 22 for CEC 2011, or 10 for CEC 2020), and such values are called averaged ranks in the further text. Based on these averaged ranks (the lower, the better), the final ranking of algorithms is created for each benchmark set and each computational budget. This way we obtain eight rankings, for four benchmark sets and two separate computational budgets. The position in the particular ranking indicates how good, or how poorly the specific algorithm performs on the particular benchmark and the computational budget – with respect to other metaheuristics. We may observe which algorithms improve and which deteriorate their relative positions (relative with respect to other algorithms tested) when the computational budget is extended ten times.

However, the ranking of algorithms does not say how much improvement is de facto achieved by a particular method on 10CNFC with respect to CNFC. The best way to learn that would be to compare raw performances (mean objective function values) obtained by each algorithm for every problem after CNFC and after 10CNFC. We perform such a comparison in this study, but only in a limited way, due to the sheer number of algorithms and problems. For a more general discussion, we need some aggregation measures of the relative improvement (relating the performances obtained with 10CNFC to those noted with CNFC). Based on the idea of various convergence rate measures [16] we define such relative improvement obtained by the specific algorithm after the extended number of function calls as the difference in performance after CNFC and 10CNFC search, divided by the difference between the performance reached after CNFC and the objective function value in the global optimum:

$$IMPR = -100\left(\frac{f_{CNFC} - f_{10CNFC}}{f_{CNFC} - f^*}\right) \tag{1}$$

In Eq. (1) f^* is the function value in the global optimum in the case of CEC 2020, CEC 2017, and CEC 2014 problems, for which the

Table 1
Algorithms compared. D – problem dimensionality.

Chronological number	Short name	Long/descriptive name	Reference	Year	Comments
1	ADDE	Adaptive distributed DE	[19]	2020	Population size is variable, depending on the success achieved in recent generations by the current population. The initial and the maximum population sizes are set to 10D; the minimum value is set to 2D. The initial version of the algorithm was tested in [19] on twenty two real-world problems from CEC 2011 (with 150,000 function calls) and on thirty 30- and 100-dimensional CEC 2014 problems (with 10,000D function calls).
2	AGSK	Adaptive gaining-sharing knowledge algorithm	[20]	2020	Population size was set by the authors of the algorithm to 40D if $D < 6$ and to 100 otherwise. AGSK is the second-best algorithm in the IEEE Competition on Evolutionary Computation 2020. The initial version of the algorithm was tested in [20] on ten, 5-, 10-, 15-, and 20-dimensional CEC 2020 problems, with 50,000, 1,000,000, 3,000,000, and 10,000,000 function calls, respectively.
3	AHA	Artificial hummingbird algorithm	[21]	2022	Population size = 50. This is a new bio-inspired algorithm. The algorithm has been tested in [21] on fifty basic functions of different dimensionalities (with 50,000 function calls), on thirty 30-dimensional CEC 2014 functions (with an unusual number of function calls set to 25,000), and on ten, generally low-dimensional real-world problems (number of function calls depended on the problem, but was within 9000–50,000).
4	ALC-PSO	PSO with ageing leader	[22]	2013	Population size = 20. The algorithm has been tested in [22] on seventeen 30-dimensional basic functions from different sources, including some from the CEC 2005 benchmark (the number of function calls was set to 200,000).
5	ATPS-DE	JADE with adaptive population tuning	[23]	2013	Population size is adaptive during run, initialized with 5D (but within [50,200]). The algorithm has been tested in [23] on twenty five 30- and 100-dimensional CEC 2005 problems, with a number of function calls set to 10,000D.
6	CDE	Clustering DE	[24]	2011	Population size = 100. The algorithms have been tested in [24] on thirty functions from various sources, including the CEC 2005 benchmark. The dimensionalities of the problem varied from 2 to 30, and the number of function calls from 10,000 to 500,000.
7	CIJADE	Hybrid DE	[25]	2020	Population size = 100. The algorithm was tested in [25] on twenty eight 50- and 100-dimensional problems from CEC 2013 (with 10,000D function calls) and on two 20- and 30-dimen-sioanl real-world problems (with just 6000 function calls).
8	CLPSO	Comprehensive learning PSO	[26]	2006	Population size = 40. State-of-the-art PSO variant. The algorithm has been tested in [26] on sixteen 10- and 30-dimensional functions from different sources (with the number of function calls set between 20,000 and 300,000).
9	CoBiDE	DE based on Covariance Matrix learning	[27]	2014	Population size = 60. DE algorithm that adaptively rotates coordinates when the search moves are performed, and uses a bimodal distribution setting of parameters. State-of-the-art algorithm. The algorithm has been tested in [27] on twenty five 30-dimensional CEC 2005 problems with the number of function calls set to 10,000D.
10	CS-DE	Cooperative strategy-based DE	[28]	2021	Population size is variable, initialized with $25\ln(D)\sqrt{D}$(but not fewer than 25). The algorithm has been tested in [28] on twenty eight CEC 2013 problems and thirty CEC 2014 problems. The problems were 10-, 30- and 50-dimensional, and the number of function calls was set to 10,000D.
11	DEPSO	Dual environmental PSO	[29]	2019	Population size = 50. The algorithm has been tested in [29] on twenty eight 30-dimensional CEC 20,013 problems with the number of function calls set to 10,000D.
12	Di-DE	Depth information-based DE	[30]	2020	Population size = 10D. The algorithm has been tested in [30] on twenty eight CEC 2013 problems and thirty CEC 2017 problems. The problems were 10-, 30- and 50-dimensional, and the number of function calls was set to 10,000D.
13	ELSHADE-SPACMA	Enhanced L-SHADE-SPACMA	[31]	2021	Population size is linearly reduced during run from 18D at the beginning to 4 at the end. Hybrid algorithm. The algorithm has been tested in [31] on thirty 10-, 30-, 50-, and 100-dimensional CEC 2017 problems with the number of function calls set to 10,000D.
14	EnsDE	Ensemble of DE algorithms	[32]	2018	Population size = 100. State-of-the-art ensemble of DE variants. The algorithm has been tested in [32] on twenty five 30- and

(*continued on next page*)

Table 1 (*continued*)

Chronological number	Short name	Long/descriptive name	Reference	Year	Comments
					50-dimensional CEC 2005 problems with the number of function calls set to 10,000D.
15	EPSO	Ensemble of PSO variants	[33]	2017	Population size = 40, divided into two uneven swarms. The algorithm has been tested in [33] on twenty five 10- and 30-dimensional CEC 2005 problems with the number of function calls set to 10,000D.
16	FDBSFS	Fitness distance-based stochastic fractional search	[34]	2021	Population size = 50. The algorithm has been tested in [34] on thirty basic functions from different sources, thirty CEC 2014 problems, thirty CEC 2017 problems and five real-world applications. The dimensionality of benchmark problems ranged from 30 to 100, and for real-world applications from 3 to 7. The number of function calls was set, depending on the problem, to either 10,000D or 500,000.
17	HARD-DE	Hierarchical archive-based DE	[35]	2019	Population size is parabolically (quicker at the end of the search) decreased during run from $25\ln(D)\sqrt{D}$ to 4. The algorithm has been tested in [35] on twenty eight CEC 2013 problems and thirty CEC 2017 problems of dimensionality 10, 30 and 50 (with the number of function calls set to 10,000D), and on two 6- and 20-dimensional real-world problems from CEC 2011 (with the number of function calls set to 150,000).
18	HIP-DE	Historical population-based adaptive DE	[36]	2021	Population size is variable, initialized to 15D. The algorithm has been tested in [36] on twenty eight 10-, 30-, and 50-dimensional CEC 2013 problems. The number of function calls was set to 10,000D.
19	HIVBBO	Hybrid Invasive Weed and Biogeography-based optimization	[37]	2017	Population size = 100. The algorithm has been tested in [37] on twenty five 30-dimensional CEC 2005 problems, twenty four 10-dimensional BBOB2015 problems, and two 36- and 47-dimensional real-world problems. The number of function calls was set to 10,000D.
20	HSES	Hybrid Sampling Evolution Strategy	[38]	2018	Population size = 200. The winner of the IEEE Competition on Evolutionary Computation in 2018. The algorithm has been tested in [38] on twenty nine 10-, 30-, 50-, and 100-dimensional CEC 2018 problems. The number of function calls was set to 10,000D.
21	jDE-div	DE with diversity and adaptive population size	[39]	2019	Population size is initialized with 50 but varies adaptively during search within [8,5D] range. The algorithm adds diversity-based mechanism to jDE method. The algorithm has been tested in [39] on thirty 10-, 30-, 50-, and 100-dimensional CEC 2014 problems. The number of function calls was set to 10,000D.
22	L-SHADE	SHADE with linear population size reduction	[40]	2014	A version of Successful History Adaptive DE with population size linearly reduced during the run from 18D at the beginning to 4 at the end. State-of-the-art algorithm. The algorithm has been tested in [40] on thirty 10-, 30-, 50-, and 100-dimensional CEC 2014 problems. The number of function calls was set to 10,000D.
23	L-SHADE-50	Simplification of L-SHADE variants	[41]	2018	Population size is linearly decreased from 18D at the beginning of the search to 4 at the end. In contradiction to the majority of new variants, this algorithm is simpler than the algorithms on which it is based. The algorithm has been tested in [41] on thirty 50-dimensional CEC 2014 problems (with the number of function calls set to 10,000D) and on twenty two CEC 2011 real-world problems of different dimensionalities (with the number of function calls set to 150,000).
24	MaDE	Bayesian hyperparameter-optimization DE	[42]	2021	Population size is linearly decreasing during the search, from initial $2D^2$ to 4 at the end of the run. The algorithm has been tested in [42] on ten 10- and 20-dimensional CEC 2021 problems. The number of function calls was set to 200,000 and 1,000,000, respectively.
25	N-L-SHADE	Spatial-neighborhoods-based L-SHADE	[43]	2022	Population size is linearly decreased from 18D at the beginning of the search to 4 at the end. Control parameters F and CR are adaptively modified based on successful sets of parameters in the neighborhood of particular individual. The algorithm has been tested in [43] on twenty eight CEC 2013, thirty CEC 2014, fifteen CEC 2015, thirty CEC 2017 benchmarks, and twenty two real-world problems from CEC 2011. The dimensionality of the problems ranged from 10 to 100 for benchmarks and from 1 to 216 for real-world problems. The number of function calls was

(*continued on next page*)

Table 1 (*continued*)

Chronological number	Short name	Long/descriptive name	Reference	Year	Comments
					set to 10,000D for benchmarks, and 150,000 for real-world problems.
26	OLSHADE-CS	DE orthogonal array-based initialization and new selection strategy	[44]	2022	New DE selection technique is proposed, which considers the relative performance of the trial solution among solutions in its neighborhood. A new initialization method is also introduced. The population size is decreasing linearly from $6D^2$ to 4 during the search. The algorithm has been tested in [44] on ten 5-, 10-, 15-, and 20-dimensional CEC 2020 problems with the number of function calls set to 50,000, 1,000,000, 3,000,000, and 10,000,000, respectively.
27	PSO-sono	PSO for single-objective problems	[45]	2022	Population size = 100. A new fully informed search scheme is proposed, and control parameters are adaptively modified during the search. The algorithm has been tested in [45] on twenty eight CEC 2013, thirty CEC 2014 and thirty CEC 2017 benchmarks. The dimensionality of the problems ranged from 10 to 100. The number of function calls was set to 10,000D.
28	SOMA T3A	Version of self-organizing migrating algorithm	[46]	2019	Population size = 1500. One among recently proposed Self-Organizing Migration algorithms that has a 20-year history of development. The algorithm has been tested in [46] on ten 9-to-18-dimensional problems with an unspecified number of function calls.
29	TAPSO	Triple-archive PSO	[47]	2020	Population size = 60. The algorithm has been tested in [47] on thirty 10-, 30-, and 50-dimensional problems collected from various sources. The number of function calls was set to 10,000D.
30	TbL-SHADE	Tuning-based mutation L-SHADE	[48]	2020	Population size = 100. The algorithm has been tested in [48] on ten 5-, 10-, 15-, and 20-dimensional CEC 2020 problems with the number of function calls set to 50,000, 1,000,000, 3,000,000, and 10,000,000, respectively.

global optimum is known. In the case of CEC 2011 problems, for which the global optimum is not known, f^* is the best objective function value found for the particular problem by any algorithm used in this study with any computational budget. f_{CNFC} and f_{10CNFC} are 51-runs averaged objective function values obtained by the specific algorithm on the particular problem with CNFC and 10CNFC, respectively.

According to the Eq. (1), the lower the value of *IMPR*, the larger improvement is obtained by the algorithm during 10CNFC. The best possible improvement equals −100, which means that the global solution is (almost) found in each run by the particular algorithm after 10CNFC, but such solution has not been found after CNFC. If the global optimum is found during CNFC, the denominator of E q. (1) would be 0, hence we skip such a problem from comparisons between computational budgets (as it is meaningless), but do not skip this problem when we compare performances between algorithms.

Note that the lower values of *IMPR*, the better (larger) improvement is obtained during 10CNFC. We may expect *IMPR* values to be negative – what means that the solutions found after 10CNFC would be better than the solutions found with CNFC computational budget. However, positive values of *IMPR* may happen, and they mean that the algorithm obtained worse results when it was run for 10CNFC than for CNFC. For example, −50 means that on average twice better solutions were found after 10CNFC than with CNFC, and + 50 means that 50 % poorer solutions were obtained after 10CNFC than after CNFC. The positive values may come up by chance (for example, if an algorithm sticks once in 51 runs in a very poor local optimum), but may also indicate problems with algorithm operators or adaptation procedures that could change too slowly when the search is ten times longer. We remind here that the maximum improvement equals −100, which means that the global optimum was found after 10CNFC ($f_{10CNFC} = f^*$), but it was not found after CNFC. However, there is no limit on the deterioration of the results: a single big failure noted by an algorithm in one run during 10CNFC may highly affect (increase) *IMPR* value, as it is based on 51-runs averaged performance.

2.3. Evolutionary algorithms compared

Thirty optimization algorithms used in this study are introduced in Table 1. The control parameters of these algorithms are kept as suggested in the source papers, with a few exceptions clarified in Table 1. The population size for each algorithm is specified in Table 1, as the default values may vary highly from one paper to another (e.g. [17]). At the end of each line, we add some comments on the specific algorithms, including information on the previous successes or specific requirements of the particular method. Among the comments given in Table 1 we also provide detailed information on benchmarks on which particular algorithm was tested in the initial publication, including the name of the benchmark sets, the number of problems in each set, and the number of function calls used. This may suggest for what kind of applications the particular algorithm has been originally tuned by its inventors. Although the choice of metaheuristics used in this study is arbitrary and subjective, we have included into the competition some classical algorithms (e.g. CLPSO, L-SHADE), some winners of the IEEE Competitions on Evolutionary Computation (e.g. HSES, AGSK), some recent variants of famous methods (N-L-SHADE, PSO-sono), and some more exotic, nature-inspired algorithms (AHA, HIVBBO). Part of algorithms used

in this study share similar roots (i.e. belong to the same family of methods, like PSO or L-SHADE), but each algorithm has individual operators or features that make it a distinctive approach. It is well known that the original naming of some algorithms does not necessarily mean the originality of the operators, and that the algorithms introduced under novel inspirations frequently do not guarantee high-quality performance [4]. Hence, we tried to avoid methods whose novelty has been disputed [4]. The oldest algorithm considered in this study was proposed in 2006 (CLPSO), and four algorithms were presented in the year 2022 (AHA, OL-SHADE, N-L-SHADE, PSO-sono). We hope that the spectrum of metaheuristics is sufficiently wide to represent the general impact of the extended number of function calls on the performance and relative ranking of metaheuristics.

2.4. Statistical tests

This study does not focus on any specific algorithm, and the sheer comparison among different algorithms is not the central part of the research. However, some statistical tests are still needed to validate differences in performance obtained by different algorithms on the specific benchmark sets and computational budgets. This study performs a pair-wise comparison between algorithms on each benchmark set, following [18]. We verify the statistical significance of the pair-wise comparisons among 30 algorithms on each benchmark set by means of the Friedman's test with the post-hoc Shaffer's static procedure at $\alpha = 0.05$, as suggested in [18].

3. Results and discussion

3.1. Impact of the extended number of function calls on the ranking of metaheuristics

In this section, we analyze how the tenfold extension of computational time, in other words stopping the algorithm after 10CNFC instead of CNFC, affects the ranking of metaheuristics. We aim to note which algorithms benefit more than the others from extending the number of function calls to 10CNFC. Note that the best algorithm after CNFC cannot improve its position in the specific ranking after an extended search, as it already has been a winner; it may only keep the first place, or lose it. In this sub-section, we do not

Table 2
Averaged ranks of 30 metaheuristics on four various benchmark sets with classical (abbreviated as CNFC in the paper text) and ten times extended (10CNFC) maximum number of function calls. The classical value refers to the number of function calls suggested in the source paper for the specific benchmark set.

			50-dimensional CEC 2014		50-dimensional CEC 2017		15-dimensional CEC 2020		CEC 2011	
Number	Algorithm	Year	Averaged ranks classical	Averaged ranks extended	Averaged ranks classical	Averaged ranks extended	Averaged ranks classical	Averaged ranks extended	Averaged ranks classical	Averaged ranks extended
1	ADDE	2020	14.48	13.45	11.37	11.63	14.25	16.65	16.66	12.09
2	AGSK	2020	18.15	15.47	18.17	14.42	6.90	4.60	19.25	15.89
3	AHA	2022	18.47	20.08	26.10	26.97	22.70	21.30	17.61	20.73
4	ALC_PSO	2013	26.40	26.25	26.73	27.20	24.80	24.90	22.43	23.32
5	ATPS_DE	2013	12.48	13.75	13.82	16.87	17.05	18.85	11.25	13.18
6	CDE	2011	18.72	18.15	19.98	18.87	16.25	16.25	15.75	14.16
7	CIJADE	2020	12.52	15.88	12.80	15.73	16.15	18.25	8.98	13.55
8	CLPSO	2006	19.03	18.97	19.75	17.57	14.90	12.40	17.89	16.68
9	CoBiDE	2014	12.92	13.97	13.22	15.20	13.65	17.95	9.02	12.30
10	CS_DE	2021	7.92	9.25	6.52	7.87	11.85	12.20	9.98	7.98
11	DEPSO	2019	18.85	20.98	16.50	18.73	26.35	26.50	21.14	24.41
12	Di_DE	2020	12.88	15.15	11.77	12.83	19.55	20.05	17.34	13.59
13	ELSHADE-SPACMA	2021	6.75	7.47	4.32	6.02	13.05	12.95	11.61	10.95
14	EnsDE	2018	13.40	13.30	12.72	14.37	10.55	12.05	14.16	10.30
15	EPSO	2017	17.52	16.93	17.83	17.23	17.95	14.90	14.61	14.70
16	FDBSFS	2021	18.47	19.32	23.40	24.63	17.75	20.50	17.39	19.50
17	HARD_DE	2019	7.55	9.28	5.87	7.33	11.05	12.55	8.11	7.68
18	HIP_DE	2021	8.78	9.87	5.72	8.52	10.85	12.45	9.16	9.84
19	HIVBBO	2017	15.73	17.20	15.57	16.53	24.20	20.90	15.80	18.00
20	HSES	2018	8.93	10.60	5.58	6.17	19.75	20.95	18.68	18.05
21	jDE_div	2019	12.65	11.93	11.97	11.53	11.75	12.65	15.52	14.05
22	L_SHADE	2014	8.02	9.03	6.12	7.65	10.95	12.35	8.75	8.61
23	L_SHADE_50	2018	6.45	7.68	4.98	5.92	13.35	14.95	9.02	9.73
24	MaDE	2021	23.18	10.13	24.67	10.90	6.85	7.90	18.98	15.82
25	N_L_SHADE	2022	13.45	14.57	11.63	13.83	13.85	15.25	10.75	9.36
26	OLSHADE_CS	2022	22.00	16.47	25.87	15.92	4.95	4.95	26.25	25.16
27	PSO_sono	2022	22.45	23.48	22.87	24.37	21.85	23.70	21.80	22.32
28	SOMA_T3A	2019	29.53	26.33	29.97	28.17	21.00	6.10	26.84	25.77
29	TAPSO	2020	19.75	21.55	21.00	23.10	24.20	25.00	17.43	21.77
30	TbL_SHADE	2020	17.57	18.50	18.22	18.93	6.70	5.00	12.84	15.52

discuss the range of improvement obtained by specific algorithms on particular problems, but limit our discussion to the relative rankings of metaheuristics.

Averaged ranks on four different benchmark sets, separately for CNFC and 10CNFC, are summarized in Table 2. Based on Table 2, the ranking's positions obtained by 30 metaheuristics on particular benchmark sets are given in Table 3 and Fig. 1. Note that in Fig. 1 the position in particular ranking obtained by the specific algorithm for the CNFC is marked by the blue bar, and for 10CNFC by the orange bar; the lower the bar is, the better algorithm. In Table 4 and Fig. 2, we show the aggregated ranking of algorithms obtained by averaging each algorithm's ranking position over all four sets of problems; each set was given equal weight, regardless of the number of problems. Such aggregated views facilitate generalization of the results by skipping the specific features of each benchmark set. The final position given at the two last columns of Table 4 refers to the averaged ranking (with classical or extended number of function calls) shown in the same table. The statistical significances of the pair-wise comparisons between all 30 algorithms on all four benchmark sets under both computational budgets are given in Suppl. Material, Suppl. Tables 1–8.

From Fig. 1 and Tables 2–3, we see that the rankings of algorithms are relatively similar for CEC 2014 and CEC 2017 sets, but different for CEC 2020 and CEC 2011 ones. CEC 2014 and CEC 2017 benchmarks share the number of problems, the types of problems, and the number of allowed function calls. In the case of CEC 2014 and CEC 2017 benchmarks, the ranking positions of the majority of algorithms are only moderately affected when 10CNFC is used, instead of CNFC. From Fig. 1 one notes that the differences between blue and orange bars are small for most algorithms. For example, EL-SHADE-SPACMA and L-SHADE-50 are ranked as two the best methods, both for CNFC and 10CNFC. The worst two methods are SOMA-T3A and ALC-PSO, again for both CNFC and 10CNFC. In general, the changes in the algorithm's positions in rankings when 10CNFC is used instead of CNFC are often not larger than 5 locations, out of 30 competitors.

There are, however, two major exceptions on CEC 2014 and CEC 2017 benchmarks. The MaDE algorithm is ranked among the worst methods when CNFC is used (it is 28th for CEC 2014 and 26th for CEC 2017), but becomes among the best eight methods (7th for CEC 2014 and 8th for CEC 2017) when the number of function calls is extended to 10CNFC. The improvement in ranking's position achieved by the OL-SHADE-CS algorithm obtained when 10CNFC is used instead of CNFC is lower, but still noticeable; OL-SHADE improves from 26th to 18th for CEC 2014 and from 27th to 17th for CEC 2017. We must point out that these two algorithms were recently proposed in their source papers tested on the CEC 2020 benchmark set that allows a much larger number of function calls than CEC 2014 and CEC 2017 sets. The present finding confirms that both MaDE and OL-SHADE-CS perform poorly when the number of allowed function calls is low, but both can efficiently use additional computational time. On the contrary, some algorithms (AHA,

Table 3
Ranking of 30 metaheuristics on four various benchmark sets with classical (abbreviated as CNFC in the paper text) and ten times extended (10CNFC) maximum number of function calls.

			50-dimensional CEC 2014		50-dimensional CEC 2017		15-dimensional CEC 2020		CEC 2011	
Number	Algorithm	Year	Ranking classical	Ranking extended	Ranking classical	Ranking extended	Ranking classical	Ranking extended	Ranking classical	Ranking extended
1	ADDE	2020	15	11	8	10	15	18	17	9
2	AGSK	2020	19	16	19	14	4	1	25	19
3	AHA	2022	20	25	28	28	26	26	21	24
4	ALC_PSO	2013	29	29	29	29	29	28	28	27
5	ATPS_DE	2013	8	12	15	19	19	21	9	11
6	CDE	2011	22	21	22	23	18	17	15	15
7	CIJADE	2020	9	17	13	16	17	20	3	12
8	CLPSO	2006	24	23	21	21	16	9	22	20
9	CoBiDE	2014	12	13	14	15	13	19	4	10
10	CS_DE	2021	4	4	7	6	10	7	7	2
11	DEPSO	2019	23	26	17	22	30	30	26	28
12	Di_DE	2020	11	15	10	11	22	22	18	13
13	ELSHADE-SPACMA	2021	2	1	1	2	11	13	10	8
14	EnsDE	2018	13	10	12	13	5	6	12	7
15	EPSO	2017	17	19	18	20	21	14	13	16
16	FDBSFS	2021	21	24	25	27	20	23	19	23
17	HARD_DE	2019	3	5	5	4	8	11	1	1
18	HIP_DE	2021	6	6	4	7	6	10	6	6
19	HIVBBO	2017	16	20	16	18	27	24	16	21
20	HSES	2018	7	8	3	3	23	25	23	22
21	jDE_div	2019	10	9	11	9	9	12	14	14
22	L_SHADE	2014	5	3	6	5	7	8	2	3
23	L_SHADE_50	2018	1	2	2	1	12	15	5	5
24	MaDE	2021	28	7	26	8	3	5	24	18
25	N_L_SHADE	2022	14	14	9	12	14	16	8	4
26	OLSHADE_CS	2022	26	18	27	17	1	2	29	29
27	PSO_sono	2022	27	28	24	26	25	27	27	26
28	SOMA_T3A	2019	30	30	30	30	24	4	30	30
29	TAPSO	2020	25	27	23	25	28	29	20	25
30	TbL_SHADE	2020	18	22	20	24	2	3	11	17

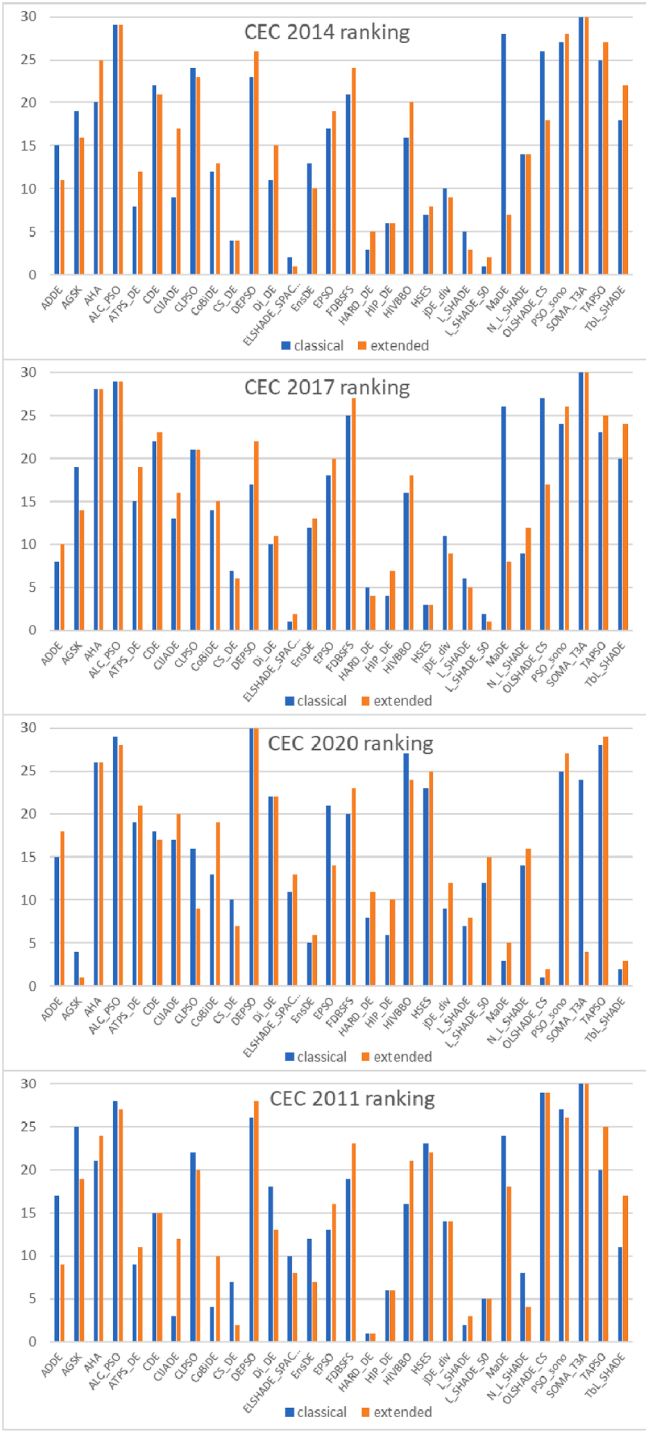

Fig. 1. Ranking of metaheuristics on four benchmark sets with classical and extended number of function calls.

CIJADE, DEPSO) noted a relatively large deterioration in ranking's position when 10CNFC is used instead of CNFC, suggesting that they could not sufficiently benefit from additional function calls.

From Suppl. Tables 1–4 we may find that for CEC 2014 and CEC 2017 benchmarks, the majority of differences between various algorithms are not statistically significant. This result may be easily understood, as we use a large number of algorithms, among which many may be considered state-of-the-art methods. However, something more important may be noted from statistical tests. By summing up cases with significant and insignificant differences given in Suppl. Tables 1–4, we find that about 34 % (in the case of CEC

Table 4
Averaged over four sets of benchmark problems ranking of algorithms with classical (abbreviated as CNFC in the paper text) and tenfold extended (10CNFC) number of function calls. When two algorithms share the same averaged ranking, both are given the same (better among the two) final position.

Number	Algorithm	Year	Averaged ranking classical	Averaged ranking extended	Final position classical	Final position extended
1	ADDE	2020	13.75	12.00	14	11
2	AGSK	2020	16.75	12.50	17	12
3	AHA	2022	23.75	25.75	25	26
4	ALC_PSO	2013	28.75	28.25	30	30
5	ATPS_DE	2013	12.75	15.75	12	16
6	CDE	2011	19.25	19.00	20	22
7	CIJADE	2020	10.50	16.25	7	17
8	CLPSO	2006	20.75	18.25	22	21
9	CoBiDE	2014	10.75	14.25	9	13
10	CS_DE	2021	7.00	4.75	6	1
11	DEPSO	2019	24.00	26.50	26	27
12	Di_DE	2020	15.25	15.25	16	15
13	ELSHADE-SPACMA	2021	6.00	6.00	5	5
14	EnsDE	2018	10.50	9.00	7	7
15	EPSO	2017	17.25	17.25	18	20
16	FDBSFS	2021	21.25	24.25	24	25
17	HARD_DE	2019	4.25	5.25	1	3
18	HIP_DE	2021	5.50	7.25	4	6
19	HIVBBO	2017	18.75	20.75	19	23
20	HSES	2018	14.00	14.50	15	14
21	jDE_div	2019	11.00	11.00	10	9
22	L_SHADE	2014	5.00	4.75	2	1
23	L_SHADE_50	2018	5.00	5.75	2	4
24	MaDE	2021	20.25	9.50	21	8
25	N_L_SHADE	2022	11.25	11.50	11	10
26	OLSHADE_CS	2022	20.75	16.50	22	18
27	PSO_sono	2022	25.75	26.75	28	29
28	SOMA_T3A	2019	28.50	23.50	29	24
29	TAPSO	2020	24.00	26.50	26	27
30	TbL_SHADE	2020	12.75	16.50	12	18

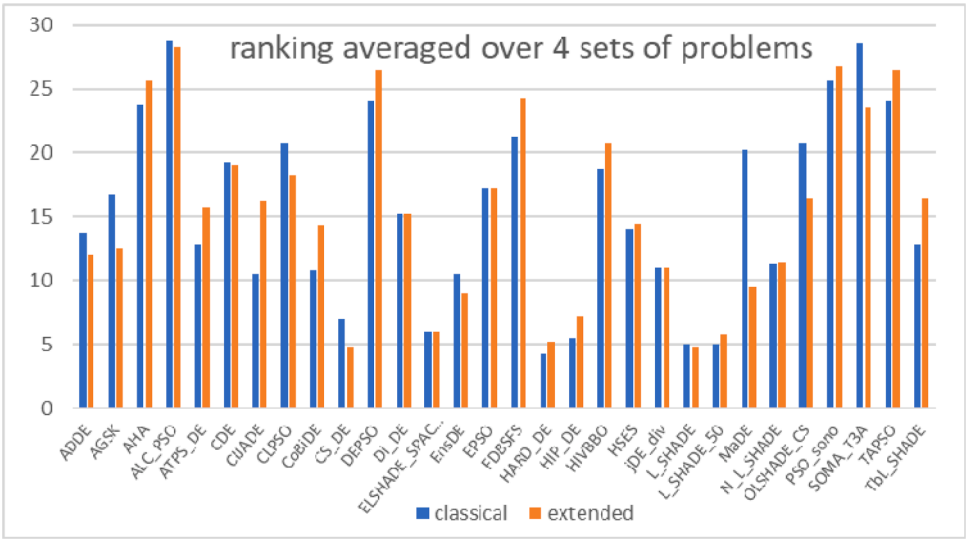

Fig. 2. Ranking of metaheuristics averaged over all four benchmark sets.

2014) and about 42 % (in the case of CEC 2017) of differences between algorithms are statistically significant when CNFC is used, but only 25 % (in the case of CEC 2014) and 38 % (in the case of CEC 2017) when the computational budget is extended to 10CNFC. It means that after the prolonged search, the differences between various algorithms are less significant, in other words – the performances of various algorithms are more unified on mathematical benchmark problems when the search is much longer. This may in turn suggest that poor algorithms benefit relatively more from additional search, and may narrow the gap to the best methods. On the

contrary, noticeable improvement of the best solutions during the extended search is more challenging.

In the case of CEC 2020 problems the observed differences in ranking positions obtained by the majority of algorithms when either CNFC or 10CNFC are used are generally low. However, large changes in ranking positions are nonetheless observed for a few algorithms. This seems to be easily explained. For CEC 2020 problems the value of CNFC is very large, compared to other benchmark sets. Hence, after searching with the CNFC many algorithms either find solutions close to the global optimum, lose diversity, prematurely converge to some local optima, or face stagnation. In any of these cases, the probability of decisive improvement during the additional computational time (e.g. when 10CNFC is used) is low. However, this general observation refers only to the majority of algorithms, not all of them. Some algorithms cannot find the global optimum within CNFC, but also avoid mentioned traps and still can achieve a significant improvement during 10CNFC.

The best positions in rankings based on CEC 2020 problems, both with CNFC and 10CNFC, are obtained by AGSK, OL-SHADE-CS, and TbL-SHADE algorithms. MaDE is also among the best five methods, regardless of the number of function calls. The worst methods on CEC 2020 benchmark are DEPSO, TAPSO, and, like for CEC 2014 and CEC 2017, ALC-PSO. This may suggest that PSO algorithms experience problems with benefiting from the long search. This would confirm findings from [9] that the PSO algorithm and some of its variants perform relatively better, with respect to other algorithms, when a low number of function calls is allowed. The algorithms that on CEC 2020 benchmarks avoid traps related to very long searches and highly benefit from the extended number of function calls (10CNFC) are SOMA-T3A (ranked 24th for CNFC and 4th for 10CNFC), EPSO (ranked 21st and 14th, respectively), CLPSO (ranked 16th and 9th) and AGSK (ranked 4th and 1st). The improvements obtained by CLPSO and EPSO, variants that are characterized by very slow information sharing among particles, show that some PSO versions may be ready for the extended search if the problem of keeping diversity and avoiding premature convergence is solved. Interestingly, in the case of CEC 2020 problems, extending the computational budget from CNFC to 10CNFC have little impact on the ranking positions obtained by the adaptive DE algorithms.

From Suppl. Tables 4 and 5, one may read that the differences between algorithms on CEC 2020 problems are rarely statistically significant, probably due to the low number of problems included in this benchmark set. However, contrary to the results from CEC 2014 and CEC 2017 benchmarks, the number of significant differences noted after 10CNFC is slightly higher (8 %) than for CNFC (6 %).

On real-world CEC 2011 problems, the differences in ranking positions of many algorithms achieved when computational budget is extended from CNFC or 10CNFC seem to be large, but generally do not extend ten locations. HARD-DE is clearly the best method under both computational budgets, SOMA-T3A is the poorest. However, when algorithms are stopped after 10CNFC, instead of CNFC, the ranking's positions obtained by ADDE, AGSK, CS-DE, Di-DE, EnsDE, and MaDE improve by between 5 and 10 locations, and the ranking's position obtained by CIJADE, CoBiDE, HIVBBO, TAPSO, and TbL-SHADE deteriorate by between 5 and 10 locations. This may suggest that the extended computational time (10CNFC) has larger impact on the performance of metaheuristics on some real-world problems. Unfortunately, it is hard to spot any clear similarities between algorithms that improve their rankings or those whose ranking positions deteriorate when the search takes 10CNFC instead of CNFC. Interestingly, among two algorithms (MaDE and OL-SHADE-CS) that highly improved their positions in CEC 2014 and CEC 2017 rankings when computational budget was prolonged to 10CNFC, on real-world problems MaDE also efficiently used additional computational time, but OL-SHADE-CS did not. We may speculate that OL-SHADE-CS operators turned out inappropriate for real-world problems.

The differences between algorithms obtained on real-world problems are frequently not statistically significant. However, contrary to CEC 2014 and CEC 2017 benchmarks, after 10CNFC differences between a larger number of algorithms (20 %) are statistically significant than after CNFC (15 %). This suggests that the extra time for search may be more important for real-world problems than for the benchmarks based on mathematical functions. For CEC 2011 problems, the diversity of solutions found by different optimization methods may increase when the search is longer. In other words, better algorithms use additional computational time more efficiently than poorer ones on real-world problems, contrary to what was observed for artificial benchmarks.

As the rankings of various algorithms could differ for specific benchmark sets, using benchmarks-averaged ranks may help finding general relations between algorithms (see Table 4). A brief look at Table 4 may focus our attention on L-SHADE or CS-DE algorithms that, although were never ranked best on any specific benchmark, share the best-averaged rank overall when the number of function calls is extended to 10CNFC. Nonetheless, there are clear differences in rankings among these two algorithms for classical and extended computational budgets. L-SHADE is overall 2nd best for CNFC, and shares the 1st position with CS-DE for 10CNFC. This is not true for CS-DE, which was 6th for CNFC, and became the co-winner with L-SHADE for 10CNFC. It looks that CS-DE benefits much more from additional computational time than L-SHADE.

To summarize: when the search is extended from CNFC to 10CNFC, the ranking's position of the majority of metaheuristics changes only modestly for mathematical benchmark functions (CEC 2014, CEC 2017, and CEC 2020). For real-world problems (CEC 2011) the additional computational time significantly affects the ranking of metaheuristics. However, some algorithms can benefit from additional computational time on very different kinds of problems, both real-world ones and mathematical functions. This is clearly seen in Fig. 2, in which a ranking of algorithms averaged over all four sets of benchmarks is presented. In that ranking an equal weight is attributed to each set, no matter how many problems are included within the particular benchmark. As may be seen, MaDE, AGSK, and SOMA-T3A are algorithms that benefit most when the search is extended from CNFC to 10CNFC, even though SOMA-T3A remains inefficient on real-world problems. These three algorithms are relatively new methods, tested in the source papers on very large computational budgets. They perform poorly when the computational budget is low, but retain a high potential to benefit from extra time. On the other hand, algorithms that are outperformed by many others because they fail to efficiently use additional computational time include CoBiDE, CIJADE, TbL-SHADE, ATPS-DE, and FDBSFS. These methods are quite diversified, but most of them use adaptive control parameters – and such adaptation schemes are probably inappropriate for the tenfold longer search. These adaptive algorithms that perform best with CNFC (L-SHADE-50 and ELSHADE-SPACMA on CEC 2014 and CEC 2017, OL-SHADE-CS on CEC 2020, and

HARD-DE on CEC 2011) often do not lose their leading positions when computational budget is extended to 10CNFC, but as leaders are obviously unable to improve. Note that algorithms, whose positions in rankings have deteriorated when the number of function calls was increased tenfold, might also improve the performance during additional time, but to a lesser degree than other metaheuristics. However, whether all metaheuristics do indeed improve when the search is extended from CNFC to 10CNFC, and how large the real improvement is, is a separate question that will be addressed in the following subsection.

3.2. Range of improvements achieved when the number of function calls is increased ten times

In this section, we discuss the range of improvements obtained by individual algorithms, and by metaheuristics in general, when the number of allowed function calls is extended ten times, from CNFC to 10CNFC. Theoretically, one could expect that the longer search would always lead to better results, and that the more computational time – the better. This is, however, not necessarily true as many metaheuristics may lose diversity and become trapped in local optima. In such a case, if the method cannot escape or regenerate, all function evaluations performed after the algorithm is trapped are wasted, and the performance depends of the averaged objective function values in the local minima in which an algorithm terminated in different runs. When the algorithm is routinely trapped in the local optima of the particular problem in the early part of the search, the length of the search become unimportant, and results obtained with CNFC budget may simply by chance be better than those achieved after 10CNFC. This should make algorithms that start with larger population sizes, use various operators, and have self-adaptive control parameters better suited for long-haul search than the simple metaheuristics with fixed operation rules. The current section will examine if this is true in numerous empirical comparisons.

In Supplementary Material (Supplementary Figs. 1–4) we show directly the 51-runs averaged performances obtained by each algorithm on every problem after CNFC and 10CNFC. We will discuss the main observed features in the text, but due to the high number of cases, the exact discussion of each problem or algorithm is impossible. After that direct comparison, we will discuss in more aggregated way the improvements obtained by metaheuristics when the computational budget is extended from CNFC to 10CNFC.

3.2.1. Improvements observed for the specific problems and algorithms

First, let us discuss briefly the raw results: performances obtained on the specific problems by various algorithms during a search when the budget is set to either CNFC or 10CNFC (see Supplementary Material, Suppl. Figs. 1–4). When looking at Suppl. Figs. 1–4, we quickly note that the improvements obtained when the search is ten times longer highly depend on both the problem and algorithm, but the problem is more important. We will first discuss specific function types from artificial benchmarks (CEC 2014, CEC 2017, and CEC 2020), and then briefly refer to real-world problems.

Mathematical functions from CEC 2014, CEC 2017, and CEC 2020 sets include some unimodal functions (these are problems no. 1–3 in CEC 2014, no. 1–2 in CEC 2017, and no. 1 in CEC 2020). For such problems exploitation and convergence speed are more important than exploration – the quicker convergence, the better results should be obtained for unimodal problems. We may expect that the best algorithms for such problems may be either those that were developed mainly for exploitation purposes, or those with flexible or adaptive operators/control parameters. From the first plots of Suppl. Figs. 1–3 we find that after CNFC (blue bar) only some algorithms (e.g. CS-DE, EL-SHADE-SPACMA, HARD-DE, L-SHADE-50) in each run converge to the results very close to the global optimum of unimodal functions. Many other algorithms converge too slowly and cannot reach the global optimum within CNFC, but do that within 10CNFC. One may also find metaheuristics that improve on unimodal problems with extended time, but not sufficiently to converge to the global optimum in each run (e.g. DEPSO, PSO-sono). Finally, some algorithms (e.g. AHA, ALC-PSO, SOMA-T3A) may be unable to converge to any reasonable solution on some unimodal problems (such as no. 1 of CEC 2014 or no. 2 of CEC 2017), even with the extended computational budget (10CNFC).

To summarize, the benefit from the extended search on unimodal problems differs highly from algorithm to algorithm, and from problem to problem. The best metaheuristics do not need additional time on unimodal problems. However, a large number of algorithms need the extra time and can efficiently use it, finally reaching results of the same quality as the best methods did. Some algorithms, though, do fail.

Other kinds of mathematical functions considered in CEC 2014, CEC 2017, and CEC 2020 benchmarks are simple multimodal functions (no. 4–16 in CEC 2014, no. 3–9 in CEC 2017, and no. 2–4 in CEC 2020). Such functions are multimodal and require exploration capabilities, but at least according to the idea of their authors, remain relatively simple. We observe that the performances of metaheuristics on multimodal functions may highly differ from function to function. For some problems (e.g. no. 5, 11, 13–16 from CEC 2014, no. 4 and 7 from CEC 2017, and no. 3 from CEC 2020) no algorithm can find the global optimum in each run, both when the computational budget is set to CNFC or 10CNFC. On some problems, the performance of different algorithms are very similar; on others – highly diversified. For example, in the case of problem no. 5 from CEC 2014 all algorithms perform almost equally, and nearly all obtain only marginal improvement when 10CNFC is used, instead of CNFC. On the other hand, for problem no. 11 from CEC 2014 large differences between the performance of algorithms are seen (even though none algorithm finds a global optimum in each run), and some metaheuristics (ADDE, MaDE) can benefit from extended search much more than the others. There are also problems (e.g. no. 12 from CEC 2014, no. 3 from CEC 2017, no. 2 from CEC 2020) for which some algorithms found global optimum within CNFC, but much more algorithms do that when the number of function calls is extended to 10CNFC. In such cases increasing the number of allowed function calls would mitigate the performance differences obtained by various metaheuristics. Finally, some multimodal problems turned out relatively simple for the majority of metaheuristics, regardless of the computational budget (e.g. no. 7 and 10 from CEC 2014, no. 6 and 9 from CEC 2017).

We may summarize that on the majority of multimodal problems, the performances of metaheuristics obtained after CNFC are highly uneven, and improvements achieved by the majority of algorithms when the computational budget is increased to 10CNFC are

diversified. Some algorithms note larger improvements on almost all multimodal problems (e.g. MaDE), and some others frequently fail to benefit from the additional time (e.g. FDBSFS). Overall, tenfold extension of the number of function calls allows the majority of metaheuristics to improve on multimodal functions, although not on all problems (e.g. problems 5 and 14 from CEC 2014, or 3 from CEC 2020 are the exceptions).

Finally, the remaining problems from CEC 2014 (no. 17–30), CEC 2017 (no. 10–30), and CEC 2020 (no. 5–10) are complicated, multimodal hybrid or composition functions. These are considered to be the most difficult problems in these three benchmark sets. However, from the perspective of improvements achieved during the extended computational time, these problems may be divided into two subsets.

The first subset includes problems on which the performances of different algorithms are diversified, and algorithms generally improve when the computational budget is extended from CNFC to 10CNFC (problems no. 17–22 and no. 29 from CEC 2014, no. 10–14, 16–17, 20, 22 and 29 from CEC 2017, no. 5–7 from CEC 2020). On this kind of problems, some algorithms clearly improve when 10CNFC is used instead of CNFC, surpassing many competitors. Examples of such improving algorithms include e.g. CDE, DEPSO, EPSO, TAPSO on problem no. 17 from CEC 2014, MaDE, OLSHADE-CS, TbL-SHADE on problem no. 18 from CEC 2014, CDE, HIVBBO on problem no. 19 from CEC 2014, about 10 algorithms on problem no. 20 from CEC 2014, DEPSO, EPSO, HIVBBO, MaDE, OLSHADE-CS, TAPSO on problem no. 21 from CEC 2014, MaDE, OLSHADE-CS, SOMA-T3A on problem no. 11 from CEC 2017, AGSK, CoBiDE, EnsDE, MaDE on problem no. 12 from CEC 2017, AGSK on problem no. 22 from CEC 2017, MaDE and SOMA-T3A on problem no. 29 from CEC 2017, AGSK, SOMA-T3A, TbL-SHADE on problem no. 6 from CEC 2020, and CLPSO or EPSO on problem no. 7 from CEC 2020.

The second subset includes problems on which the search during the extended time is wasted by almost all algorithms (problems no. 23–28 and 30 from CEC 2014, no. 15, 18–19, 21, 23–28, 30 from CEC 2017, no. 8–10 from CEC 2020). For these problems either no improvement, or marginal improvement is noted by the vast majority of algorithms (though some specific algorithms may improve). Algorithms that can improve somehow on problems from this subset include e.g. SOMA-T3A on problems no. 23–25, 27–28, on problem 30 from CEC 2014, and on problems no. 20–29 from CEC 2017, MaDE and OLSHADE-CS on problem no. 30 from CEC 2017, AGSK and OLSHADE-CS on problem no. 8 from CEC 2020, or OLSHADE-CS on problem no. 10 from CEC 2020. As may be seen, improvements on such kinds of problems are obtained generally by a few algorithms that share one similarity, namely were developed in recent years deliberately for long-haul search (SOMA-T3A, OLSHADE-CS, MaDE). Among them, SOMA-T3A frequently obtains significant improvements when the search is extended to 10CNFC, even though it fails to exceed performances obtained by the majority of other algorithms. On the contrary, MaDE and OLSHADE-CS may improve sufficiently when the search is extended, and after 10CNFC belong to the best methods for some of these problems.

To summarize, on roughly half of difficult hybrid or composition functions many algorithms efficiently use additional function calls and improve their performance. For the second half of hybrid or composition functions, the vast majority of algorithms either fail to improve during the extra time, or achieve only a marginal improvement. Only three algorithms: MaDE, OLSHADE-CS, and SOMA-T3A, were able to improve more significantly on some of these hard-to-improve problems when the search was extended from CNFC to 10CNFC. Hence, additional computational time may be helpful only for about half of hybrid or composition problems and is generally wasted for the remaining ones.

Some more detailed discussion is needed on the relative improvements obtained by metaheuristics on CEC 2011 real-world problems when the computational budget is extended from CNFC to 10CNFC. We skip 1-dimensional problem no. 3, on which every algorithm found the global optimum within the classical computational budget.

With respect to the performances obtained by different algorithms, we may divide real-world problems into three sub-sets: 1. those on which the vast majority of algorithms reach almost the same results (no. 8, 11.3, 11.5, 11.10); 2. those on which the majority of algorithms achieve similar performances, but some algorithms perform much differently (no. 5, 6, 10, 11.2, 11.4, 11.6, 11.7, 11.8) and 3; those on which different algorithms often perform differently (no. 1, 2, 4, 7, 9, 11.1, 11.9, 12, 13).

On problems from sub-set 1 (no. 8, 11.3, 11.5, 11.10), the performance of metaheuristics after CNFC and 10CNFC is almost identical. The dimensionality of these problems is diversified and ranges from 7 to 120, hence it is rather not a decisive factor for the uniform performance of algorithms. For such kinds of problems, almost all algorithms converge to the same quality solutions within CNFC, and when the search is extended to 10CNFC, the additional function calls are wasted. Only in the case of the problem 11.10 the algorithms AGSK, ALC-PSO, Di-DE, MaDE, OLSHADE-CS, and SOMA-T3A, which failed to reach the same performance as the majority of metaheuristics when the computational budget was restricted to CNFC, were able to improve and achieve similar performance to the others when the number of function calls was extended to 10CNFC (in the case of SOMA-T3A, OLSHADE-CS, and MaDE the results remained slightly poorer, but comparable to the majority of other methods). Note that the global optima of the real-world problems are not known in advance, hence we cannot know whether almost all algorithms successfully converge to the solutions located close to the global optimum, or all were stuck in similar-quality local optima. Anyway, it seems that in the case of problems from sub-set 1, some basins of attractions with solutions of almost uniform quality are found by nearly all metaheuristics, and even if some better solutions exist, they remain unknown even after the search is extended ten times.

In the case of 13- to 216-dimensional problems from sub-set 2 (no. 5, 6, 10, 11.2, 11.4, 11.6, 11.7, 11.8), when the computational budget is increased from CNFC to 10CNFC, some minor improvements are observed for the majority of algorithms, but no method is able to achieve a break-through and significantly improve the performance. This may indicate that for these problems the majority of metaheuristics localized some basins of attractions with similar quality solutions during CNFC, but could not converge to the local optima within that computational budget. The longer search (10CNFC) allowed the majority of algorithms to perform some successful exploitative steps, and find out slightly better solutions. However, no breakthrough exploration took place during the extended search.

On problems from sub-set 3 (no. 1, 2, 4, 7, 9, 11.1, 11.9, 12, 13), on which diversified performance of algorithms is observed after

CNFC, the impact of tenfold extended search (10CNFC) on the final performance is also highly different and problem-dependent. On problems 1 and 4, the improvements obtained by any algorithm when the computational budget is extended to 10CNFC are, at best, marginal. Interestingly, the problem no. 4 is 1-dimensional. The fact that the majority of metaheuristics obtain even slight improvement on the 1-dimensional problem after performing 150,000 function calls (as for CNFC) is still an intriguing result, which shows that metaheuristics may not lose diversity in 1-dimensional space even after so long search. On the 96-dimensional problem 11.9 a number of metaheuristics, which performed poorly after CNFC, were able to significantly improve when ten times more function calls were allowed. However, the vast majority of algorithms that reached better results within CNFC, were at best able to marginally improve when the computational budget was extended to 10CNFC. Nonetheless, on the majority of problems from sub-set 3 (no. 2, 7, 9, 11.1, 12, and 13) a large fraction of metaheuristics obtained some clear improvements when the search was extended from CNFC to 10CNFC. The mentioned problems belong to different fields of science and engineering, but are rather higher-dimensional ones (their dimensionality range between 20 and 126).

To summarize, the efficiency of increasing the computational budget from CNFC to 10CNFC highly depends on the specific real-world problem. One may find problems on which after both CNFC and 10CNFC the performances obtained by nearly all metaheuristics are almost identical (e.g. problems 8, 11.3, or 11.5). There are also problems for which nearly all metaheuristics clearly improve if the number of function calls is tenfold extended (e.g. problems 7, 9 or 12). As could be expected, the problems on which larger improvements are obtained are rather higher-dimensional. However, some improvements were also achieved by many algorithms even for just 1-dimensional problem no. 4.

Contrary to what was observed for the mathematical functions, on which the largest improvements during longer search were achieved mainly by some adaptive, exploration-oriented algorithms (e.g. MaDE, OLSHADE-CS, SOMA-T3A, AGSK), on real-world problems frequently the improvements obtained were either observed for the majority of metaheuristics, or for almost none of

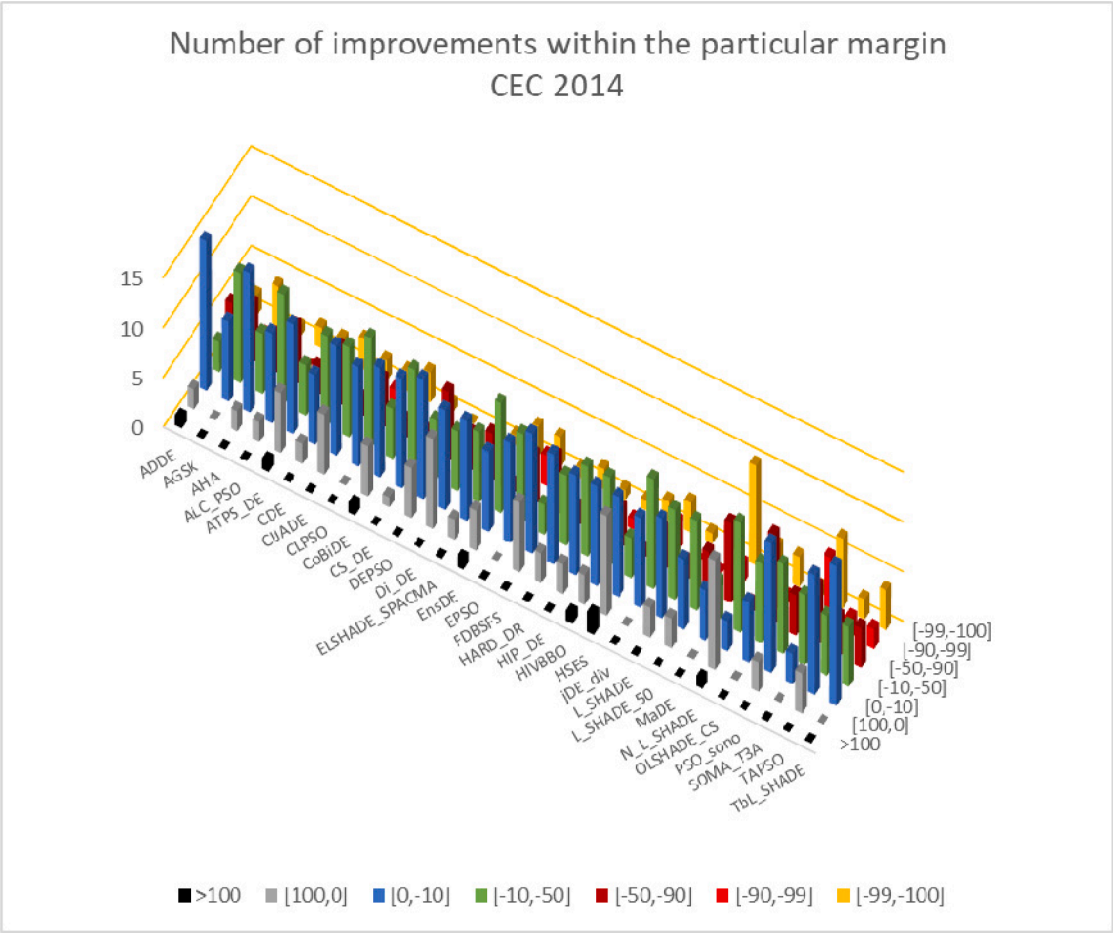

Fig. 3. Number of CEC 2014 problems within a particular range of relative percentage improvements IMPR obtained during the extended computational time. Ranges refer to the percentage of negative improvement, for example [−90, −99] range includes the number of problems for which a particular algorithm improved during the additional computation time from 90% to 99%, which means that the solution found when the computational time was extended tenfold was between 90% and 99% closer to the global optimum than the solution found by the same algorithms after the classical number of function calls.

them. This is counterintuitive and does not agree with our initial expectations expressed at the beginning of section 3.2. It seems that in the case of real-world problems after CNFC either the vast majority of metaheuristics lose diversity and converge to some optima, or almost all algorithms retain some diversity and ability for further search. It does not mean that the extra computational time is unimportant for real-world problems. It is rather the opposite – for real-world problems the improvement after the extended search is, on average, quite large. The unexpected point is that this improvement is either similarly large, or similarly non-existent, for both simple and advanced metaheuristics. Hence, the positive feedback from the complicated operators or self-adaptivity during extended search is surprisingly low.

3.2.2. Aggregated view on improvements obtained with extended computational time

To analyze the improvements obtained in extended computational time in a more aggregated way, we refer to the measure of relative improvement *IMPR* defined in Eq. (1). The values of *IMPR* have been computed for each algorithm and problem. To facilitate discussion, we define seven ranges of *IMPR* values that represent different changes in the algorithm's performance when the computational budget is extended from CNFC to 10CNFC. For each benchmark set, we count the number of problems on which particular algorithm reached the value of improvement *IMPR* within the specific range; these numbers are given in Suppl. Tables 9–12 and are illustrated graphically in Figs. 3–6. The seven *IMPR* ranges (see Suppl. Tables 9–12and Figs. 3–6) are defined as follows. The first two ranges include cases when the algorithm did not improve at all when 10CNFC was used. The first range (>100) represents cases in which at least two times poorer results were obtained in extended search (10CNFC) than during the search with CNFC. The second range [100,0] includes cases when the performance after 10CNFC was not better than the performance obtained with CNFC, but has worsened no more than twice. Why do some results deteriorate so much when the search takes longer time? As we clarified earlier, the runs for CNFC and 10CNFC are independent (the 10CNFC search starts from the random initialization of algorithms, and is

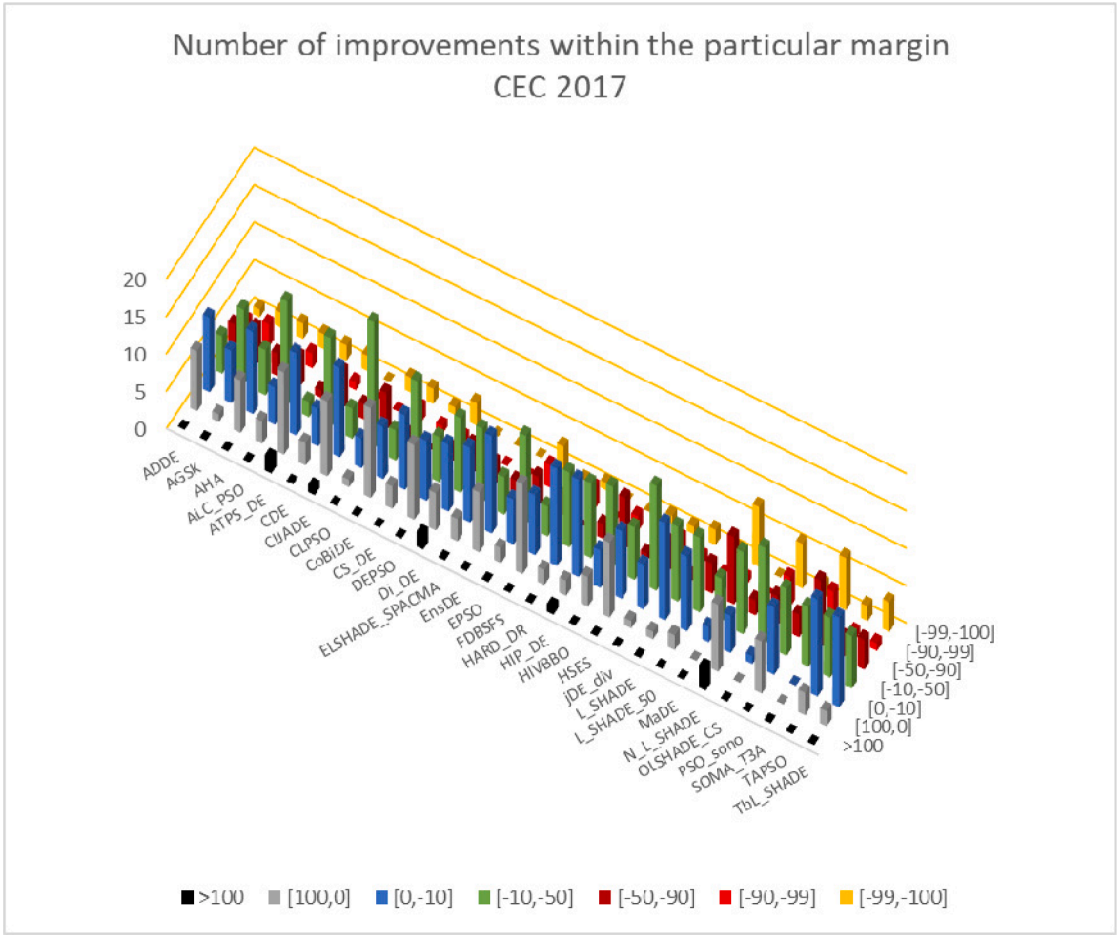

Fig. 4. Number of CEC 2017 problems within a particular range of relative percentage improvements IMPR obtained during the extended computational time. Ranges refer to the percentage of negative improvement, for example [−90, −99] range includes the number of problems for which a particular algorithm improved during the additional computation time from 90% to 99%, which means that the solution found when the computational time was extended tenfold was between 90% and 99% closer to the global optimum than the solution found by the same algorithms after the classical number of function calls.

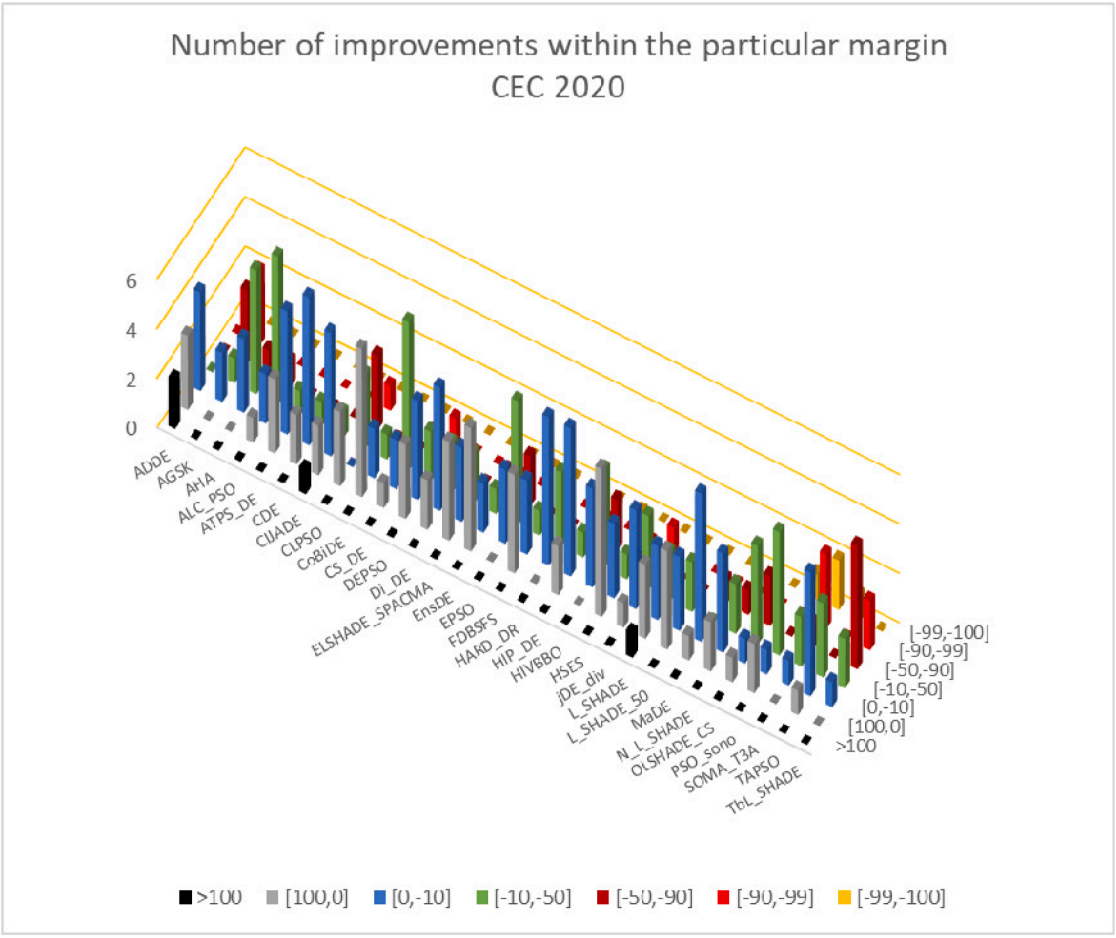

Fig. 5. Number of CEC 2020 problems within a particular range of relative percentage improvements IMPR obtained during the extended computational time. Ranges refer to the percentage of negative improvement, for example [−90, −99] range includes the number of problems for which a particular algorithm improved during the additional computation time from 90% to 99%, which means that the solution found when the computational time was extended tenfold was between 90% and 99% closer to the global optimum than the solution found by the same algorithms after the classical number of function calls.

not the continuation of the runs with CNFC). Hence, the algorithms may achieve poorer performance after 10CNFC either by chance, or because some of its operators or adaptation schemes were developed for the quick search, and do not prevent premature convergence or stagnation when the search takes a lot of time. The remaining five ranges count problems for which the particular algorithm improved, but by different percentage margins: [0, −10] means marginal improvement, [−10, −50] means that results found after 10CNFC were up to twice better (two times closer to the global optimum in terms of fitness) than solutions found after CNFC, [−50, −90] means that solutions found were more than twice better, [−90, −99] means that the solutions found within additional time were at least 10 times closer to the global optimum in terms of fitness, and [−99, −100] counts cases for which the global optimum has almost been found after 10CNFC, but was not found during CNFC (note that −100 is that highest possible improvement, and means that the global optimum was found after 10CNFC). We have to remind here that, when during CNFC the global optimum (or, in the case of CEC 2011, the best-known solution) has always been reached by the particular algorithm in 51 runs, the denominator in eq. (1) becomes 0 and the relative improvement cannot be computed. Such cases did happen, e.g. many algorithms always found the best-known solution of problem no. 3 or 8 from CEC 2011 set within the classical computational time, some algorithms (e.g. ELSHADE-SPACMA) always found global optima of 50-dimensional CEC 2014 problems no. 2 and 9, etc. We skip such cases in Suppl. Tables 9–12 and Figs. 3–6, as they are meaningless for the discussion on improvement during the extended search. As a result, an uneven number of cases may be attributed to different algorithms. Finally, the number of *IMPR* cases summarized over all algorithms for each range is given in Fig. 7.

We start our discussion from CEC 2014 and CEC 2017 problems, on which the improvements obtained by various algorithms when the search is extended from CNFC to 10CNFC are relatively low. On the majority of problems, the solutions found after 10CNFC are no more than twice better than solutions found after CNFC (this situation include ranges [100,0], [0, −10], and [−10, −50]). On many CEC 2014 and CEC2017 problems the majority of algorithms obtain minor improvements; some may even deteriorate. However, some

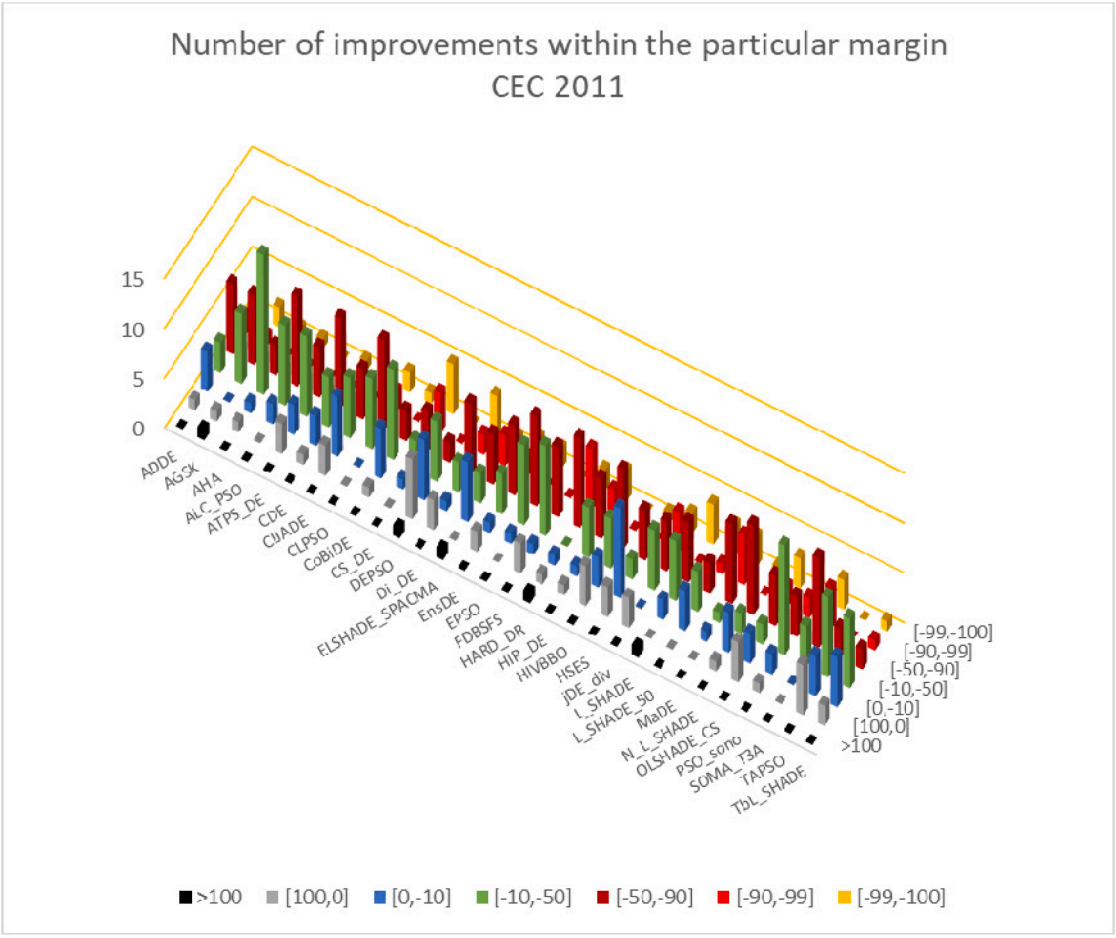

Fig. 6. Number of CEC 2011 real-world problems within a particular range of relative percentage improvements IMPR obtained during the extended computational time. Ranges refer to the percentage of negative improvement, for example [−90, −99] range includes the number of problems for which a particular algorithm improved during the additional computation time from 90% to 99%, which means that the solution found when the computational time was extended tenfold was between 90% and 99% closer to the global optimum than the solution found by the same algorithms after the classical number of function calls.

algorithms may note large improvements when the search is extended to 10CNFC, finding solutions 10–100 times better (closer to the global optimum in terms of fitness) than solutions found after CNFC. For example, when the search is extended from CNFC to 10CNFC, MaDE finds 100 times better solutions for 18 problems, SOMA-T3A for 14 problems, OLSHADE for 9 problems, and TbL-SHADE for 8 problems, out of 60 included in both CEC 2014 and CEC 2017 sets. This shows that the longer search may be very beneficial for some metaheuristics on a relatively large number of problems.

However, some algorithms surprisingly frequently perform poorer when the computational budget is extended from CNFC to 10CNFC. For example, for about 30 % of CEC 2014 and CEC 2017 problems, after extended search (10CNFC) the algorithms HSES, FDBSFS, CoBiDE, N-L-SHADE, or ATPS-DE found solutions that are not better, and in some cases even much poorer, than solutions found after CNFC. This indicates that numerous algorithms suffer from premature convergence, stagnation, or a lack of population diversity that effectively hampers their work during the latter part of the search. In the case of some algorithms, the adaptation techniques may be fitted to the specific computational budget (CNFC), but become inefficient when the search is much longer. A detailed discussion of the reasons is beyond the present study (it should be focused on details of each algorithm separately), but we note that the problem is frequent. It may be an interesting point for further research, to clarify why exactly some metaheuristics cannot benefit from additional function calls when others clearly can.

To summarize, the improvements achieved by most metaheuristics on the majority of typical benchmark problems (e.g. CEC 2014 or CEC 2017) are somewhat limited. However, generalization is difficult, as some algorithms benefit clearly from the extended search on many problems, while others frequently fail to improve.

To our surprise, very similar results were obtained on CEC 2020 problems (see Fig. 5 and Suppl. Table 11), for which the classical number of function calls (CNFC) was much higher (3,000,000 for 15-dimensional versions considered in this study) than for CEC 2014 and CEC 2017 problems (500,000 for 50-dimensional versions used in this research). The only clear difference between results

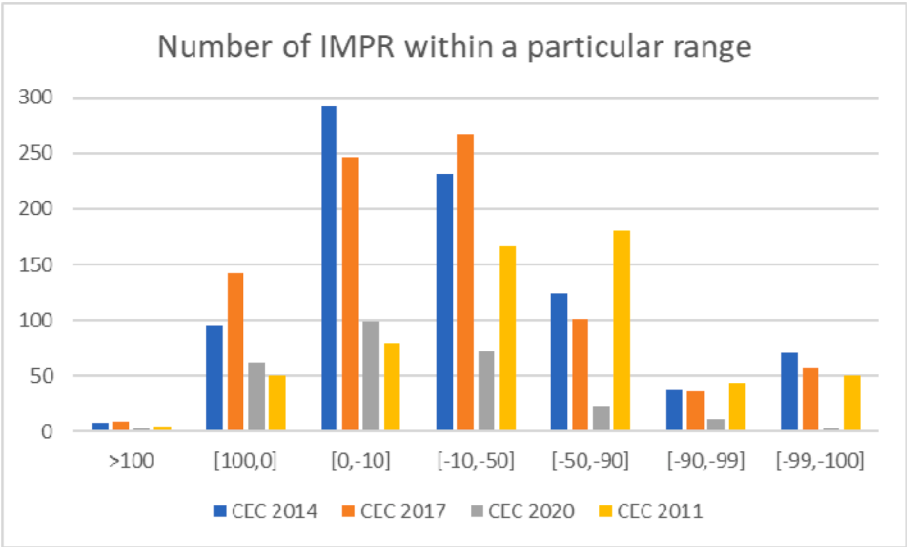

Fig. 7. All-algorithms cumulative number of improvements IMPR within each range for every benchmark set.

obtained for CEC 2020 against those discussed for CEC 2014 and CEC 2017 is that on a few CEC 2020 problems 10–100 times better solutions are found by any algorithm when the search is extended ten times. Only SOMA-T3A, TAPSO, and PSO-sono were able to find 100 times better solutions on at least a single problem when the search was extended to 10CNFC for CEC 2020 benchmarks. However, none of these three algorithms finally reached competitive performance; although these algorithms efficiently used additional time, they were still clearly behind the best methods on CEC 2020 benchmark. On the other hand, when the search is extended, many algorithms fail to improve at all on many CEC 2020 problems. The percentage of problems on which HSES, CoBiDE, or EnsDE fail to improve when the search is longer reaches 50 %. The least improving algorithms are the same as in the case of CEC 2014 and CEC 2017 problems (HSES, CoBiDE, EnsDE), what is rather not accidental. It turns out that some algorithms are simply unable to benefit from the extended search, at least on benchmark problems. This may be due to some grid-based regularities that may exist in some artificial problems, or due to insufficiently flexible operators or adaptation schemes that are incorporated into these algorithms.

Let us now discuss real-world CEC 2011 problems. From Fig. 6 and Suppl. Table 12, we note that when the search is extended from CNFC to 10CNFC, the solutions found by almost all metaheuristics on the vast majority of real-world problems are between 10 % and 90 % better (closer to the global optimum in terms of fitness). However, some particular metaheuristics, like HSES, ELSHADE-SPACMA, CIJADE, or DEPSO, may obtain only marginal improvement (lower than 10 %) on most real-world problems.

Nonetheless, when the search is extended from CNFC to 10CNFC, the majority of metaheuristics (23 out of 30) noted poorer performance on at least a single real-world problem. Among these 23 algorithms, DEPSO did not obtain any improvement on the largest number of problems (7 out of 22). This algorithm must frequently lose diversity during a search, or is trapped by some local minima. The seven algorithms that always performed either better (or at least no poorer) when the number of function calls was extended from CNFC to 10CNFC are ALC-PSO, CLPSO, CS-DE, EPSO, L-SHADE-50, MaDE, and SOMA-T3A. When we compare ranking positions on CEC 2011 real-world problems obtained by these seven algorithms after CNFC, we found that L-SHADE50 and CS-DE were ranked very high (1st and 4th, respectively), but the remaining five algorithms performed moderate-to-poor. Hence, the extended number of function calls on real-world problems turns out always beneficial for two different types of algorithms: two front runners that were able to efficiently use additional time and always improve (or at least never deteriorate) their performance when the computational budget was ten times longer, and five algorithms that deeply needed additional time to solve real-world problems. Interestingly, although these seven algorithms improved their performance (or at least never deteriorated) on each real-world problem when the search was ten times longer, the majority of them were unable to improve their positions in the CEC 2011 ranking, or did so just marginally (see Suppl. Table 12 for information on improvement in the performance and Table 3 to see the CEC 2011 rankings of algorithms after CNFC and 10CNFC). The two exceptions are MaDE, which improved its ranking position from the 24th to the 18th, and CS-DE that improved from the 7th to the 2nd position. The improvement obtained by CS-DE on real-world problems is the only case in which ten times extended search allowed an algorithm to jump from the moderate position in ranking into one among the top three locations.

When the search is extended from CNFC to 10CNFC, 25 out of 30 algorithms finds solutions at least 10 times better (closer to the global optimum in terms of fitness, i.e. those included in the ranges $[-90, -99]$ and $[-99, -100]$) on at least a single CEC 2011 problem. Algorithms that achieved such considerable improvements on the largest number of problems include CS-DE (9 problems), HARD-DE (8 problems), and SOMA-T3A (7 problems). On five problems, out of 22, CS-DE was even able to find solutions that were 100 times closer to the global optimum (in terms of fitness) when the number of allowed function calls was increased ten times. Even algorithms that performed best after CNFC (L-SHADE50 and ELSPADE-SPACMA) were able to find solutions 100 times better (closer to the global optimum in terms of fitness) on some real-world problems: L-SHADE-50 on three problems, ELSHADE-SPACMA on a single

CEC 2011 problem. This again shows that many metaheuristics have a potential to improve at least on some real-world problems when given much more time.

To conclude, on real-world CEC 2011 problems the vast majority of metaheuristics obtained small-to-moderate improvements when the number of function calls was extended ten times, from CNFC to 10CNFC. Numerous algorithms also achieved impressive improvements on at least a few problems and found solutions much better than those found during CNFC. Still, in about 9 % of cases, the additional computational time on real-world problems is entirely wasted and does not lead to any improvement. However, this number is much lower than in the case of artificial benchmark problems. With some surprise we may conclude that current metaheuristics more efficiently use the extra computational time on real-world problems than on the artificial benchmark functions.

3.3. Limitations of the study

As in any research, this study is inevitably limited by initial assumptions and methodological difficulties. We have focused solely on single-objective static numerical optimization problems tackled by metaheuristics. We restricted our attention to CEC benchmarks composed of mathematical functions or real-world problems. Other kind of problems, or algorithms to solve them, are not considered. We did not consider very high-dimensional problems, or extremely low computational budgets that are often solved with surrogate-assisted algorithms [1].

How the algorithms are applied may also highly affect the conclusions from our study. This is especially true because of the lack of tuning of control parameters of tested algorithms. In recent years one may observe a trend to tune control parameters of all compared algorithms separately for each problem, or at least for the particular benchmark set [5]. Without a doubt, the lack of tuning has affected the results obtained in the present study for many reasons. Different algorithms have different numbers of control parameters, and the impact of these control parameters is also versatile. Some values of control parameters would be better suited to lower, others to higher computational budgets [49]. Some values of control parameters may better fit unimodal, others – multimodal problems. Very different values of control parameters may especially be needed for real-world problems [49]. Some algorithms may be highly influenced by numerous control parameters (hence tuning them would highly improve the performance). Others, mainly those self-adaptive, may be less affected by the specific choice of control parameter values. Therefore, we may expect that some algorithms could benefit from tuning much more than the others. Pre-tuning is, however, a very delicate matter. It requires a careful determination of all control parameters that are present in the particular algorithm, and choosing which among them are to be tuned. Tuning is often very time consuming [5,49], and each algorithm should be tuned under the same conditions [50]. This is, however, difficult to guarantee due to the already mentioned reason: some algorithms have much more control parameters than the others, and the sensitivity of algorithms to different control parameters may highly vary. The choice of tuning approach and its settings (for example the number of function evaluations during pre-tuning) is difficult on its own [5,50], and may highly affect the effect of tuning and comparison. In the case of studies like the present one, the tuning should be performed separately at least for each benchmark set and each computational budget, which would make such a study extremely time-consuming and hard to follow. Finally, an event of over-tuning [50] may also hamper the results. Having all that in mind, there is little hope that any specific choice of tuning would lead to fairer results than keeping the control parameters from the literature, despite the fact that literature-based values are for sure sub-optimal for many problems solved in this study. However, readers should be aware of the evident limitations of the approach used in this study.

4. Conclusions

In the present paper we have tested 30 metaheuristics on four benchmark sets with classical and tenfold extended number of function calls. We have analyzed how the extra computational time affects the final performance of various algorithms, and the final ranking of metaheuristics. The four major questions of the study formulated in the Introduction may be answered as follows:

a. how much improvement may be obtained when the computational budget is increased by an order of magnitude? In other words, we ask whether the gain would be worth the effort.

The benefits from a tenfold extended search are highly diversified and depend on both the problem and the algorithm, which confirms former Kazikova et al. [12] findings. However, according to our results, the properties of the problem are more important than the features of the algorithm. On numerous problems the extended number of function calls is entirely wasted by all metaheuristics; on others some, or even almost all algorithms may highly improve. Hence, the answer cannot be generalized, and would mainly depend on the problem that is to be solved.

b. to what extent would the increased number of function calls affect the ranking of algorithms?

The tenfold extended number of allowed function calls modestly affects the ranking of algorithms. The vast majority of algorithms do not change their ranking positions by more than a few locations. Only some algorithms, especially those recently introduced, may significantly improve their ranking positions. Nonetheless, the same algorithms occupy the highest ranking positions after classical and extended computational budgets.

c. what kind of algorithms achieve larger improvements when the search is longer: those that already performed well, or those that performed poorly under the classical budget? Those that are adaptive, or those with fixed structure? Is the performance of various algorithms more diversified, or more similar when the search is ten times longer?

Weaker algorithms that do not lose diversity benefit most frequently from the extended search. Those algorithms that lose diversity quickly are simply unable to take advantage of additional time. For those metaheuristics that performed best after the classical number of function calls, further improvements are hard to obtain (as they have already reached very good solutions); they may also gain improvement during the extended search, but on limited number of problems.

Algorithms introduced in recent years that were initially tested on problems for which a number of function calls was high, benefit more from the extended search than older algorithms that were tested in initial papers on problems with lower computational budgets.

Algorithms with fixed structures often perform poorly, both for the classical and the extended number of function calls. Many adaptive algorithms are able to benefit highly from the extended search, however others cannot. Some adaptive procedures were probably developed for specific kinds of problems and computational budgets, and they turned out not sufficiently flexible to be efficiently applied to other problems, or larger computational budgets.

When the search is extended, the rate of statistically significant differences between performances of various algorithms is lower for benchmarks based on mathematical functions, but higher for real-world problems. Hence, after the extended search the performances of different algorithms become more diversified for real-world problems, but more similar for mathematical functions.

d. on what kind of problems the largest improvements will be observed: unimodal or multimodal, artificial benchmarks or real-world problems?

The benefits from the extended search are highly problem-dependent. The benefits are larger for real-world problems than for mathematical benchmark functions.

The improvements obtained by many algorithms on unimodal or simple multimodal mathematical functions are diversified, and depend both on the specific features of the particular problem, and on the algorithm itself. The benefits from the extended search for solving difficult composition functions are often marginal for all algorithms. When the mathematical function is simple, the best algorithms can solve it within the classical computational budget, but some among poorer metaheuristics may improve noticeably when the search is longer. If finding the optima of the mathematical function is very difficult, almost no metaheuristic would improve in any serious manner during the extended search. The largest benefits from extended search are observed for moderately difficult functions.

At least for some real-world problems the situation is much different, and even the best algorithms may highly improve when the computational budget is tenfold extended. The diversity, and the problem-dependence of the results show that future studies should be more focused on the specific problems, rather than on the benchmark-averaged statistics that hide the diversity of the results.

Author contributions

AEP and APP designed the research; APP performed the research; APP and JNN analyzed data and results; APP, JNN, and AEP wrote the paper.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

The authors do not have permission to share data.

Acknowledgments

This work was supported within statutory activities No 3841/E-41/S/2022 of the Ministry of Science and Higher Education of Poland.

Appendix A. Supplementary data

Supplementary data to this article can be found online at https://doi.org/10.1016/j.ins.2023.119766.

References

[1] J. del Ser, E. Osaba, D. Molina, X.S. Yang, S. Salcedo-Sanz, D. Camacho, S. Das, P.N. Suganthan, C.A. Coello Coello, F. Herrera, Bio-inspired computation: Where we stand and what's next, Swarm Evol. Comput. 48 (2019) 220–250.
[2] K. Varelas, O.A. El Hara, D. Brockhoff, N. Hansen, D.M. Nguyen, T. Tusar, A. Auger, Benchmarking large-scale continuous optimizers: The bbob-largescale testbed, a COCO software guide and beyond, Appl. Soft Comput. 97 (2020), 106737.
[3] N.H. Awad, M.Z. Ali, J.J. Liang, B.Y. Qu, P.N. Suganthan, Problem Definitions and Evaluation Criteria for the CEC 2017 Special Session and Competition on Single Objective Real-Parameter Numerical Optimization, Nanyang Technological University, Singapore, 2016. Technical Report.
[4] K. Sörensen, Metaheuristics—the metaphor exposed, Int. Trans. Oper. Res. 22 (1) (2015) 3–18.
[5] C. Huang, Y. Li, X. Yao, A Survey of Automatic Parameter Tuning Methods for Metaheuristics, IEEE Trans. Evol. Comput. 24 (2) (2020) 201–216.
[6] P. Posik, W. Huyer, L. Pal, A Comparison of Global Search Algorithms for Continuous Black-Box Optimization, Evol. Comput. 20 (4) (2012) 509–541.
[7] Y.D. Sergeev, D.E. Kvasov, M.S. Mukhametzhanov, On the efficiency of nature-inspired metaheuristics in expensive global optimization with limited budget, Sci. Rep. 8 (2018) 453.
[8] K.V. Price, N.H. Awad, M.Z. Ali, P.N. Suganthan, The 2019 100-digit challenge on real-parameter, single-objective optimization: Analysis of results, Nanyang Technological University, Singapore, Tech. Rep., 2019 http://www.ntu.edu.sg/home/epnsugan.
[9] A.P. Piotrowski, M.J. Napiorkowski, J.J. Napiorkowski, P.M. Rowinski, Swarm Intelligence and Evolutionary Algorithms: Performance versus speed, Inf. Sci. 384 (2017) 34–85.
[10] S. Das, P.N. Suganthan, Problem Definitions and Evaluation Criteria For CEC 2011 Competition On Testing Evolutionary Algorithms On Real World Optimization Problems, Jadavpur Univ, Nanyang Technol. Univ., Kolkata, India, 2010.
[11] A.E. Ezugwu, O.J. Adeleke, A.A. Akinyelu, S. Viriri, A conceptual comparison of several metaheuristic algorithms on continuous optimisation problems, Neural Comput. Applic. 32 (10) (2020) 6207–6251.
[12] A. Kazikova, M. Pluhacek, R. Senkerik, How does the number of objective function evaluations impact our understanding of metaheuristics behavior? IEEE Access 9 (2021) 44032–44048.
[13] Q. Chen, B. Liu, Q. Zhang, J. Liang, P.N. Suganthan, B. Qu, Problem definitions and evaluation criteria for CEC 2015 special session on bound constrained single-objective computationally expensive numerical optimization, Comput. Intell. Lab., Zhengzhou Univ., Zhengzhou, China, Nanyang Technol. Univ., Singapore, Tech. Rep., 2014.
[14] 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, 2013.
[15] C.T. Yue, K.V. Price, P.N. Suganthan, J.J. Liang, M.Z. Ali, B.Y. Qu, N.H. Awad, P.P. Biswas, Problem Definitions and Evaluation Criteria for the CEC 2020 Special Session and Competition on Single Objective Bound Constrained Numerical Optimization. Technical Report, Zhengzhou University, China and Nanyang Technological University, Singapore, 2019.
[16] A.H. Halim, I. Ismail, S. Das, Performance assessment of the metaheuristic optimization algorithms: an exhaustive review, Artif. Intell. Rev. 54 (3) (2021) 2323–2409.
[17] A.P. Piotrowski, J.J. Napiorkowski, A.E. Piotrowska, Population size in Particle Swarm Optimization, Swarm Evol. Comput. 58 (2020), 100718.
[18] S. Garcia, F. Herrera, An extension on "Statistical comparisons of classifiers over multiple data sets" for all pairwise comparisons, J. Mach. Learn. Res. 9 (2008) 2677–2694.
[19] Z.H. Zhan, Z.J. Wang, H. Jin, J. Zhang, Adaptive distributed differential evolution, IEEE Trans. Cybern. 50 (11) (2020) 4633–4647.
[20] A.W. Mohamed, A.A. Hadi, A.K. Mohamed, N.H. Awad, Evaluating the Performance of Adaptive Gaining- Sharing Knowledge Based Algorithm on CEC 2020 Benchmark Problems, in: 2020 IEEE Congress on Evolutionary Computation (CEC), Glasgow, UK, https://doi.org/10.1109/CEC48606.2020.9185901.
[21] W. Zhao, L. Wang, S. Mirjalili, Artificial hummingbird algorithm: A new bio-inspired optimizer with its engineering applications, Comput. Methods Appl. Mech. Eng. 388 (2022), 114194.
[22] W.N. Chen, J. Zhang, Y. Lin, N. Chen, Z.H. Zhan, H.S.H. Chung, Y. Li, Y.H. Shi, Particle Swarm Optimization with an aging leader and challengers, IEEE Trans. Evol. Comput. 17 (2) (2013) 241–258.
[23] W. Zhu, Y. Tang, J.A. Fang, W.B. Zhang, Adaptive population tuning scheme for Differential Evolution, Inf. Sci. 223 (2013) 164–191.
[24] Z.H. Cai, W.Y. Gong, C.X. Ling, H. Zhang, A clustering-based Differential Evolution for global optimization, Appl. Soft Comput. 11 (1) (2011) 1363–1379.
[25] J.S. Pan, N. Liu, S.C. Chu, A Hybrid Differential Evolution Algorithm and Its Application in Unmanned Combat Aerial Vehicle Path Planning, IEEE Access 8 (2020) 17691–17712.
[26] J.J. Liang, A.K. Qin, P.N. Suganthan, S. Baskar, Comprehensive learning particle swarm optimizer for global optimization of multimodal functions, IEEE Trans. Evol. Comput. 10 (3) (2006) 281–295.
[27] Y. Wang, H.X. Li, T.W. Huang, L. Li, Differential Evolution based on covariance matrix learning and bimodal distribution parameter setting, Appl. Soft Comput. 18 (2014) 232–247.
[28] Z. Meng, Y. Zhong, C. Yang, CS-DE: Cooperative Strategy based Differential Evolution with population diversity enhancement, Inf. Sci. 577 (2021) 663–696.
[29] Z.Q. Zhang, X.X. Zhu, Y.H. Wang, M.C. Zhou, Dual-environmental particle swarm optimizer in noisy and noise-free environments, IEEE Trans. Cybern. 49 (6) (2019) 2011–2021.
[30] Z. Meng, C. Yang, X. Li, Y. Chen, Di-DE: Depth Information-Based Differential Evolution With Adaptive Parameter Control for Numerical Optimization, IEEE Access 8 (2020) 40809–40827.
[31] A.A. Hadi, A.W. Mohamed, K.M., Jambi, Single-Objective Real-Parameter Optimization: Enhanced LSHADE-SPACMA Algorithm, in: Heuristics for Optimization and Learning, Stud. Comput. Intell. 906 (2021) 103-121.
[32] G. Wu, X. Shen, H. Li, H. Chen, A. Lin, P.N. Suganthan, Ensemble of differential evolution variants, Inf. Sci. 423 (2018) 172–186.
[33] N. Lynn, P.N. Suganthan, Ensemble particle swarm optimizer, Appl. Soft Comput. 55 (2017) 533–548.
[34] S. Aras, E. Gedikli, H.T. Kahraman, A novel stochastic fractal search algorithm with fitness-distance balance for global numerical optimization, Swarm Evol. Comput. 61 (2021), 100821.
[35] Z. Meng, J.S. Pan, HARD-DE: Hierarchical ARchive Based Mutation Strategy With Depth Information of Evolution for the Enhancement of Differential Evolution on Numerical Optimization, IEEE Access 7 (2019) 12832–12854.
[36] Z. Meng, C. Yang, Hip-DE: Historical population based mutation strategy in differential evolution with parameter adaptive mechanism, Inf. Sci. 562 (2021) 44–77.
[37] G. Khademi, H. Mohammadi, D. Simon, Hybrid invasive weed/biogeography-based optimization, Eng. Appl. Artif. Intel. 64 (2017) 213–231.
[38] G. Zhang, Y. Shi, Hybrid Sampling Evolution Strategy for Solving Single Objective Bound Constrained Problems, in: 2018 IEEE Congress on Evolutionary Computation (CEC), Rio de Janeiro, Brazil, https://doi.org/10.1109/CEC.2018.8477908.
[39] R. Polakova, J. Tvrdik, P. Bujok, Differential evolution with adaptive mechanism of population size according to current population diversity, Swarm Evol. Comput. 50 (2019), 100519.
[40] R. Tanabe, A. Fukunaga, Improving the search performance of SHADE using linear population size reduction, in: Proc. IEEE Congress on Evolutionary Computation, Bejing, China, 2014, pp. 1658–1665.
[41] A.P. Piotrowski, J.J. Napiorkowski, Some metaheuristics should be simplified, Inf. Sci. 427 (2018) 32–62.
[42] S. Biswas, D. Saha, S. De, A.D. Cobb, S. Das, B.A. Jalaian, Improving Differential Evolution through Bayesian Hyperparameter Optimization, in: 2021 IEEE Congress on Evolutionary Computation (CEC), Kraków, Poland, https://doi.org/10.1109/10.1109/CEC45853.2021.9504792.
[43] A. Ghosh, S. Das, A.K. Das, R. Senkerik, A. Viktorin, I. Zelinka, A.D. Masegosa, Using spatial neighborhoods for parameter adaptation: An improved success history based differential evolution, Swarm Evol. Comput. 71 (2022), 101057.

[44] A. Kumar, P.P. Biswas, P.N. Suganthan, Differential evolution with orthogonal array-based initialization and a novel selection strategy, Swarm Evol. Comput. 68 (2022), 101010.
[45] Z. Meng, Y. Zhong, G. Mao, Y. Liang, PSO-sono: A novel PSO variant for single-objective numerical optimization, Inf. Sci. 586 (2022) 176–191.
[46] Q.B. Diep, I. Zelinka, S. Das, R. Senkerik, SOMA T3A for solving the 100-digit challenge, in: Proceedings of the 2019 Swarm, Evolutionary and Memetic Computing Conference, Maribor, Slovenia, 2019.
[47] X. Xia, L. Gui, F. Yu, H. Wu, B. Wei, Y.L. Zhang, Z.H. Zhan, Triple Archives Particle Swarm Optimization, IEEE Trans. Cybern. 50 (12) (2020) 4862–4875.
[48] X. Sun, L. Jiang, Y. Shen, H. Kang, Q. Chen, Success History-Based Adaptive Differential Evolution Using Turning-Based Mutation, Mathematics 8 (2020) 1565.
[49] E. Osaba, E. Villar-Rodriguez, J. Del Ser, A.J. Nebro, D. Molina, A. LaTorre, P.N. Suganthan, C.A. Coello Coello, F. Herrera, A Tutorial On the design, experimentation and application of metaheuristic algorithms to real-World optimization problems, Swarm Evol. Comput. 64 (2021), 100888.
[50] K. Eggensperger, M. Lindauer, F. Hutter, Pitfalls and Best Practices in Algorithm Configuration, J. Artif. Intell. Res. 64 (2019) 861–893.