This is the author's final version of the paper that has been accepted for publication in Evolutionary computation.

Analysis and simplification of the winner of the CEC 2022 optimization competition on single objective bound constrained search

Rafał Biedrzycki rafal.biedrzycki@pw.edu.pl
Warsaw University of Technology, Institute of Computer Science, Warsaw, 00-665, Poland

Abstract

Extending state-of-the-art evolutionary algorithms is a widespread research direction. This trend has resulted in algorithms that give good results but are complex and challenging to analyze. One of these algorithms is EA4Eig — the winner of the CEC 2022 competition on single objective bound constrained search. The algorithm internally uses four optimization algorithms with modified components. This paper presents an analysis of EA4Eig and proposes a simplified version thereof exhibiting better optimization performance. The analysis found that the original source code contains errors that impact the algorithm's rank. The code was corrected, and the CEC 2022 competition ranking was recalculated. The impact of individual EA4Eig components on its performance was empirically analyzed. As a result, the algorithm was simplified by removing two of them. The best remaining component was analyzed further, which made it possible to remove some unnecessary and harmful code. Several versions of the algorithm were created and tested, varying in the degree of simplification. The simplest of them is implemented in 244 lines of C++ code, whereas the original implementation used 716 lines of Matlab code. Further analyses focused on the parameters of the algorithm. The constants hidden in the source code were named and treated as additional configurable parameters that underwent tuning. The ablation analyses showed that two of these hidden parameters had the most significant impact on the improvement achieved by the tuned version. The results of the original and simplified versions were compared on CEC 2022 and BBOB benchmarks. The results confirm that the simplified version is better than the original one on both these benchmarks.

Keywords

algorithms ranking, parameter tuning, benchmarking, EA4Eig algorithm, CEC 2022

1 Introduction

Optimization competitions give an impulse for the development of optimization algorithms. Competition results also facilitate identifying current state-of-the-art algorithms and future research directions. Usually, competitions are connected with conferences that strongly restrict the length of submitted papers. There is no place for a detailed algorithm analysis after discussing related work and presenting and discussing results. Frequently, implementation details are also omitted (Biedrzycki, 2021). At the time of this writing, the most recent resolved competition is the CEC 2022 competition on single objective bound constrained numerical optimization (SOBC) (Kumar et al., 2022). The winner of the competition is named EA4Eig (Bujok and Kolenovsky,

2022). It is a highly complex algorithm as it internally uses four optimization algorithms, namely: CoBiDE (Wang et al., 2014), IDEbd (Bujok and Tvrdík, 2017), CMA-ES (Hansen and Ostermeier, 2001), and jSO (Brest et al., 2017). The high complexity is also visible when looking at the source code. There are 716 nontrivial lines of code in Matlab (as reported by "cloc" tool, without a helper file that iterates over dimensions). From a scientific point of view, it is interesting to see which components and concepts were the most important to its success. This question is still not answered.

It is generally known that some modern algorithms are overcomplicated and can be simplified, e.g., Piotrowski and Napiorkowski (2018) showed that L-SHADE-EPSin could be simplified without performance loss. It might also be the case for EA4Eig. Having a simpler algorithm makes it run faster, but also, more importantly, it facilitates understanding its behavior and identifying successful concepts that should be further developed.

The main contributions of this paper are:

- The examination of the impact of EA4Eig components and parameters on its results. Thanks to this, its code was significantly simplified without quality loss on the CEC 2022 benchmark. The code size was reduced from 716 nontrivial lines of code in Matlab to 244 lines in C++. The code that remains is much easier to understand.
- The examination of the original and simplified versions of the algorithm on 24 functions from the BBOB benchmark in 10 and 40 dimensions. It was found that the simplified versions are better than the original ones when considering both dimensionalities together, but in 40 dimensions alone more gentle simplification is superior.
- Identification and correction of errors in the EA4Eig source code. Some problems that were found just slowed down the computations without disturbing the results, but others affected the rank achieved by the algorithm. The corrected CEC 2022 ranking is provided.
- Parametrization of the algorithm and parameter tuning. Constants that were hidden in the source code were named and treated as additional configurable parameters for tuning. The automatic parameter tuning was performed. The tuned parameter values improved the algorithm's performance.
- Analysis of the impact of the tuned parameters. The two most important parameters were previously hidden in the source code.
- A C++ implementation of EA4Eig with two components (jSO, IDEbd) and the final, simplified implementation provided in the supplementary materials. This faster implementation makes it possible to performing more experiments in a given time. The code could be used for additional analyses or as a base for further development.
- Providing a detailed description of DE, IDEbd, and EA4Eig in one place. This makes it easier to see similarities and differences between these algorithms.

The article is composed in the following way. Section 2 provides a literature survey that focuses on the description of EA4Eig, which is needed to understand the role of its parameters and errors found. Section 3 describes errors found in the EA4Eig source

code and examines the impact of its main components. Section 4 discusses parameter tuning and ablation analyses of the simplified algorithm. Section 5 compares the results of the simplified algorithm versions to the result of its base version on BBOB benchmark. Section 6 summarizes the observations and concludes the paper.

2 Related work

The analyses of EA4Eig will be performed using the CEC 2022 benchmark, including its ranking scheme (Kumar et al., 2022). The algorithm and its important components will be described here to help understand the meaning of its parameters and the errors found. As descriptions are based on the analysis of the official implementations, they will be more detailed in some places than in the original papers.

2.1 CEC 2022 benchmark

The CEC 2022 (Kumar et al., 2022) defined 12 functions that should be optimized in 10 and 20 dimensions. For each function and dimensionality, $n = 30$ independent runs (trials) are performed. For each run, the error level achieved by the best solution is recorded. Values lower than 10^{-8} are treated as 10^{-8}. Achieving them means that the global optimum has been found. For each function-dimensionality pair, all independent runs of all competitors are ordered by the achieved error level or by used function evaluations when optimum has been found.

The CEC 2022 ranking method is based on the Wilcoxon rank-sum test (Wilcoxon, 1945). The worst run achieves a rank of 1. The rank of the best run depends on the number of competitors and independent runs. The score of the algorithm is a sum of the ranks of its trials minus the correction term $n(n+1)/2$. An elaborate explanation of this ranking method is available in (Price et al., 2023).

2.2 Classical DE algorithm

Three of the algorithms used in EA4Eig are based on differential evolution (DE). Therefore, classical DE will be described before going into the details of EA4Eig. The pseudocode of DE is presented in Figure 1. DE maintains a population of μ individuals. The individuals are randomly initialized in the feasible region of the search space. The main loop runs until the objective function evaluation budget (fes_{max}) is exhausted or the optimum is found. In the simplest versions of DE, three individuals are used to create a mutant. First, the individual x_b, called the base individual, is mutated using the difference between two random individuals ($x_{r2} - x_{r3}$), which is multiplied by the scale factor F. The version that uses a random individual as a base is called DE/rand/1, and the version that uses the best individual as a base is called DE/best/1. After mutation, a trial vector is generated by the binomial crossover: for each dimension, a value is copied from the mutant if a random number sampled from the standard uniform distribution is less than the crossover ratio (CR) value. Otherwise, it is copied from the i-th individual, also called the target individual. During the selection, the objective function value of the target and trial individual is compared, and better enters the next generation.

2.3 EA4Eig algorithm

EA4Eig is the winner of the CEC 2022 competition. It internally uses four optimization algorithms, namely: CoBiDE (Wang et al., 2014), IDEbd (Bujok and Tvrdík, 2017), CMA-ES (Hansen and Ostermeier, 2001), and jSO (Brest et al., 2017). Additionally, all DE-based algorithms use the concept of crossover borrowed from CoBiDE, i.e., with a

1: $t = 0$
2: $P_0 = \{x_1, x_2, \ldots, x_\mu\}$ {initial population of μ random individuals}
3: **while** $fes < fes_{max}$ **and** optimum not found **do**
4: **for all** $i \in \{1, 2, ..., \mu\}$ **do**
5: $b = \text{selectBaseIndividualIndex}$ {select index of base individual}
6: $r2, r3 = \text{selectRandomIndexes}\{b \neq r_2 \neq r_3\}$
7: $v_i = x_b + F \cdot (x_{r2} - x_{r3})$ {F is a scale factor}
8: $u_i = \text{binomialCrossover}(v_i, x_i)$
9: **if** q(u_i)≤q(x_i) **then** {q is the objective function}
10: $P_{t+1,i} = u_i$
11: **else**
12: $P_{t+1,i} = x_i$
13: **end if**
14: **end for**
15: $t = t + 1$
16: **end while**

Figure 1: The pseudocode of DE.

probability of 0.4 Eigen crossover (Guo and Yang, 2015) is used; otherwise, binomial crossover is employed. The pseudocode of EA4Eig is presented in Figure 2. The algorithm starts from the random initialization of population P_0 in a feasible region of the search space. The population size μ is initially set to 100. It is reduced linearly during search like in L-SHADE (Tanabe and Fukunaga, 2014). The success history s_h counts the successes of each component algorithm. The initial success counts are set to 2. The success counter for DE-based algorithms increases when the algorithm finds a solution that passes the selection operator. In the case of CMA-ES, a success counter is increased when the algorithm finds a solution that is better than the worst in the current population. The success counters are the input for the roulette wheel selection of the component algorithm to be used in the current iteration. When the probability of selection of any component algorithm drops below p_{min}, all success counters are reset to the default value. For each place in the population, a mutant is generated using the selected algorithm. In the case of CMA-ES, the mutant is compared with currently the worst individual in the population; when it is better, it replaces it. In the case of DE-based algorithms, Eigen crossover is used with probability p_{eig}. Otherwise, binomial crossover is used. After the crossover, classical DE selection is executed. At the end of the iteration, the population size is updated.

As CMA-ES does not store its knowledge in the population, its use in EA4Eig requires further clarification. The CMA-ES iteration starts by determining the average solution based on the better half of the current population. Then, it generates trials in its typical way. The trails and previously calculated mean are used to update its internal parameters. Such coupling of CMA-ES and DE-based algorithms can be problematic as CMA-ES adapts to the current shape of the objective function. When it is not used, other algorithms in the ensemble can move the population to a part of the search space with different curvature.

When Eigen crossover is used, it utilizes CR values generated once at the beginning of the search for each place in the population (μ values). For each index, CR is drawn from the Cauchy distribution with scale parameter $\sigma_C = 0.1$ and location parameter set to 0.1 (with probability 0.5) or to 0.95 otherwise. The probability of using

$t = 0$
$P_0 = \{x_1, x_2, \ldots, x_\mu\}$; {initial population of μ random individuals}
$s_h = \{2, 2, 2, 2\}$; {success history of 4 component algorithms}
$p_{min} = 1/20$; {minimal probability of using component algorithm}
$p_{eig} = 0.4$; {Eigen crossover probability}
while $fes < fes_{max}$ **and** optimum not found **do**
  $h = \text{selectAlgorithmUsingRoulete}(s_h)$;
  if any $\text{probability}(s_h) < p_{min}$ **then**
    $s_h = \{2, 2, 2, 2\}$
  end if
  for all $i \in \{1, 2, ..., \mu\}$ **do**
    $v_i = \text{generateMutantUsingAlg}(h)$;
    if h is CMA-ES **then**
      if q(v_i)<q(x_{worst}) **then** {q is the objective function}
        $s_h(h) = s_h(h) + 1$
        $P_{t+1,worst} = v_i$ {replace the worst individual}
      end if
    else
      if $\text{U}(0, 1) < p_{eig}$ **then**
        $u_i = \text{eigenCrossover}(v_i, x_i)$
      else
        $u_i = \text{binomialCrossover}(v_i, x_i)$
      end if
      if q(u_i)<q(x_i) **then**
        $s_h(h) = s_h(h) + 1$
        $P_{t+1,i} = u_i$
      end if
    end if
  end for
  Update μ
  $t = t + 1$
end while

Figure 2: The pseudocode of EA4Eig.

0.1 as the location will be named within the paper "small Cauchy thres.". In the algorithm, only the best half of the population is used to calculate the Eigenvectors. The number of used individuals is parametrized here for tuning purposes. It is the product of "μ ratio" and population size.

2.4 IDEbd algorithm

IDEbd is used by EA4Eig. It is based on the IDE (Individual-dependent DE) (Tang et al., 2015) algorithm. The IDE search process is divided into two stages: more explorative and more exploitative. The base vector index depends on the current stage of the algorithm. The IDEbd pseudocode as it is implemented in EA4Eig is provided in Figure 3. The algorithm starts from the initialization of many constants. Most of them were not named nor discussed in (Bujok and Kolenovsky, 2022). Naming parameters and program parametrization (the ability to change parameter values without recompilation) are required for parameter tuning, which will be performed in this paper. It is important to notice that the algorithm uses iteration counter t, and maximal iteration number t_{max} is computed using the initial population size μ. The algorithm switches from stage one to stage two when the number of iterations achieves t_{th} (half of the iterations by default). The main loop runs until the objective function evaluation budget (fes_{max}) is exhausted or the optimum is found. At the beginning, the population is sorted in ascending order according to objective function q. Then, the future population is initialized by the current population. The result of the crossover will later replace some individuals. The individuals in the population are divided into superior and inferior according to objective function. The number of individuals considered superior is regulated by ps, which depends on the ratio t/t_{max}. The ps is also used to calculate perturbation probability pd, which decides how many features of x_{r3} will be replaced by a random number taken from the feasible domain (lines 18 and 19 in Figure 3). The success counter is initialized by 0. It is incremented in line 33 when the result of the crossover is not worse than the i-th individual. The success rate threshold srt is set to 0 by default. It is set to 0.1 in the second stage of the search in the source code, but it does not affect the logic of the program as it is used to decide when to switch to the second stage. The inner "for" loop iterates over all individuals. For each individual, four random indexes from the population are selected. The first of them selects the base individual, and the rest select three random individuals. These individuals are required for the mutation performed in lines 26 and 28. At the first stage of the search, the base index is set to the current index. The scale factor F is generated around b/μ using a normal distribution with σ_N set to 0.1 by default. Analogously, CR is generated, but i is used instead of b. If b and $r1$ are taken from the inferior part of the population, $r1$ is randomly drawn from the superior part. Generally, mutants are created according to the equation: $v_i = x_b + F \cdot (x_{r1} - x_b) + F \cdot (x_{r2} - x_{r3})$, but when the algorithm is in the second stage of the search and randomly drawn number is less than 0.5 the x_i is used as a base individual. This behavior is visible in the source code, but the paper (Bujok and Tvrdík, 2017) states that the best individual is used. As for the first stage of the search $b = i$, both mutation schemes (lines 26 and 28) do the same work. IDEbd normally uses binomial crossover, but when used as a component of EA4Eig, it uses the method described in Section 2.3. When the result of the crossover is not worse than the current individual, the success counter is increased, and the new individual takes i-th place in the future population (P_{t+1}). After processing all individuals in the current iteration, the failures counter is increased when there are too few successes and the algorithm is in the first stage. When there are at least $fails_{th}$ iterations without

$t = 0$; $fails = 0$; $t_{max} = fes_{max}/\mu$; $fails_{th} = t_{max}/10$
$ps_{shape} = 5$; $ps_{min} = 0.1$; $pd_{mult} = 0.1$; $t_{th\_div} = 2$; $\sigma_N = 0.1$
$t_{th} = t_{max}/t_{th\_div}$ {first stage of the search threshold}
while $fes < fes_{max}$ **and** optimum not found **do**
  $P_t = \mathrm{sort}(P_t, \mathrm{q})$ {ascending according to objective fun. q}
  $P_{t+1} = P_t$ {initialize future population}
  $ps = ps_{min} + (1 - ps_{min}) \cdot 10^{ps_{shape} \cdot (t/t_{max} - 1)}$ {prop. of the superior}
  $pd = pd_{mult} \cdot ps$ {x_{r3} perturbation prob.}
  $sc = 0$; $srt = 0$ {success counter; success rate threshold}
  for all $i \in \{1, 2, ..., \mu\}$ **do**
    $b, r_1, r_2, r_3 = \mathrm{selectRandomIndexes}(1, \mu)\{b \neq r_1 \neq r_2 \neq r_3\}$
    if $t \leq t_{th}$ **then**
      $b = i$ {base is current at first stage}
    end if
    $F = \mathrm{N}(b/\mu, \sigma_N)$ {use normal distribution}
    $CR = \mathrm{N}(i/\mu, \sigma_N)$
    for all $j \in \{1, 2, ..., D\}$ **do** {D is problem dimensionality}
      if $\mathrm{U}(0, 1) < pd$ **then** {use uniform distribution}
        $x_{r3,j} = l_j + \mathrm{U}(0, 1) \cdot (u_j - l_j)$ {l, u are lower and upper bounds}
      end if
    end for
    if $b > ps \cdot \mu$ **and** $r_1 > ps \cdot \mu$ **then**
      $r_1 = \mathrm{selectRandomIndexes}(1, ps \cdot \mu)$ {r_1 from the superior}
    end if
    if $t > t_{th}$ **and** $\mathrm{U}(0, 1) < 0.5$ **then**
      $v_i = x_i + F \cdot (x_{r1} - x_b) + F \cdot (x_{r2} - x_{r3})$
    else
      $v_i = x_b + F \cdot (x_{r1} - x_b) + F \cdot (x_{r2} - x_{r3})$
    end if
    $u_i = \mathrm{crossover}(x_i, v_i)$
    if $\mathrm{q}(u_i) \leq \mathrm{q}(x_i)$ **then**
      $P_{t+1,i} = u_i$
      $sc = sc + 1$
    end if
  end for
  if $t < t_{th}$ **then**
    if $sc \leq srt \cdot \mu$ **then**
      $fails = fails + 1$
    else
      $fails = 0$
    end if
    if $fails \geq fails_{th}$ **then**
      $t_{th} = t$ {switch to stage 2}
    end if
  end if
  Update μ
  $t = t + 1$
end while

Figure 3: The pseudocode of IDEbd.

Table 1: The CEC 2022 ranking after correction of EA4Eig bound constraint handling.

Rank	Algorithm	CEC points
1	NL-SHADE-LBC (Stanovov et al., 2022)	181311
2	EA4Eig (Bujok and Kolenovsky, 2022)	164370
3	NL-SHADE-RSP-MID (Biedrzycki et al., 2022)	157568
4	MTT-SHADE (Sun et al., 2022b)	148450
5	jSObinexpEig(Kolenovsky and Bujok, 2022)	146589
6	S-LSHADE-DP (Van Cuong et al., 2022)	143821
7	IUMOEAII (Sallam et al., 2022)	142385
8	IMPML-SHADE (Tseng, 2022)	127091
9	NLSOMACLP (Ding et al., 2022)	114572
10	ZOCMAES (Ning et al., 2022)	105830
11	OMCSOMA (Gu et al., 2022)	102039
12	Co-PPSO (Sun et al., 2022a)	95008
13	SPHH-Ensemble (Pillay and Gerber, 2022)	55769

success, the algorithm is switched to stage two of the search. It is worth noticing that the original population size reduction mechanism of IDEbd (Bujok and Tvrdík, 2017) is not used in the implementation of EA4Eig.

3 Analysis and simplification of EA4Eig

Within this section, EA4Eig will be analyzed, corrected, and simplified. The experimental procedure required for the CEC 2022 competition will be used here (Kumar et al., 2022). All 12 CEC 2022 functions in 10 and 20 dimensions will be used. To find the winning algorithm, the CEC 2022 ranking is calculated based on 30 independent runs. To reduce the random influence on the results, each algorithm uses the same seed, which results in an identical starting population for most considered variants of the algorithms (the population starts to differ when some algorithm variant uses a changed population size). The implementation of EA4Eig and raw results of the other competitors are downloaded from the official competition repository (Suganthan, 2024). The CEC ranking points are shown for each experiment to make it possible to discern small and significant differences between algorithms. The score achieved by an algorithm can be interpreted as the number of its wins when all of its trials are compared to all trials from all other algorithms. The ranking points depend on the number of competitors, so they cannot be compared between different tables.

3.1 Correcting EA4Eig bound constraint handling

During the analysis of the code of EA4Eig, it was found that the algorithm sometimes asks for the value of the objective function outside the bounds of the search space. The benchmark code does not notice that and answers. To be precise, the error was in the decision path in which both jSO and Eigen crossover were used. EA4Eig was corrected. Values after Eigen crossover are repaired in the same way as the ones after binomial crossover. Its source code and codes of other versions considered further in this paper are available on (Biedrzycki, 2024b). The CEC 2022 competition ranking was recalculated using the corrected algorithm. The results are shown in Table 1. It can be observed that after eliminating the error, EA4Eig moved to second place. Still, it is good enough to deserve further analysis.

Table 2: Analysis of the influence of EA4Eig components. The CEC points are shown for a given combination of enabled components, where "only Eig." means that only Eigen crossover is used, and "no Eig." means that it was disabled. When not specified, the algorithm uses unmodified crossover. The default configuration is typeset in bold.

rank	IDEbd	jSO	CMA-ES	CoBiDE	cross.	CEC points
1	IDEbd	jSO				221912
2	IDEbd	jSO	CMA-ES			213481
3		jSO	CMA-ES			198029
4	IDEbd	jSO		CoBiDE		194288
5	IDEbd	jSO	CMA-ES	CoBiDE	only Eig.	177274
6	IDEbd		CMA-ES			176901
7	IDEbd					176867
8		jSO	CMA-ES	CoBiDE		174197
9	IDEbd			CoBiDE		172961
10		jSO				171041
11			CMA-ES	CoBiDE		170337
12			CMA-ES			166675
13	IDEbd		CMA-ES	CoBiDE		165514
14		jSO		CoBiDE		159748
15	**IDEbd**	**jSO**	**CMA-ES**	**CoBiDE**		159090
16	IDEbd	jSO	CMA-ES	CoBiDE	no Eig.	142080
17				CoBiDE		97200

3.2 The influence of the components

As was discussed earlier, EA4Eig internally uses four optimization algorithms. The influence of all these components and the influence of replacing the default crossover by the binomial one and by Eigen crossover ($p_{eig} = 1$) are examined. The results are collected in Table 2. Each row shows the rank and the number of CEC points collected by a specified combination of enabled components. Where not specified, the default crossover is used. The name typeset in bold is the default configuration. According to the CEC ranking, the version that uses only IDEbd and jSO is the best. The IDEbd alone is the seventh, and only jSO is the tenth. The default configuration took 15th place. Changing it by turning off the Eigenvector type of crossover degrades performance. Using only Eigenvector crossover improves the ranking. The worst component is CoBiDE. It is the last in the ranking when used alone. Turning it off is enough to advance to the second position in the table. The winner also does not use it. These results show that EA4Eig could be simplified by turning off its weakest components, and this even improves its performance. It is worth noting that each component algorithm has been partially modified to fit into a common framework. Therefore, the weak result of CoBiDE turned into a component does not mean that it is a weak optimization algorithm when used standalone in a form as its authors intended.

Due to computing efficiency reasons, the code of EA4Eig with two components, namely IDEbd and jSO, was rewritten from Matlab to C++. To avoid misinterpretations and the introduction of new errors, it was ensured that both versions produced identical results. To make this possible, during validation, the default pseudorandom number generators in both languages were replaced by custom ones that used the same pseudorandom numbers previously stored in a file.

3.3 Correcting and simplifying IDEbd

As IDEbd is used in the top configurations, it was analyzed further. It was noticed that to adapt its behavior, the algorithm uses knowledge of how many generations were left till the end of the budget. The maximal number of generations is calculated at the initialization phase of the search using the initial population size. During the search, the population size decreases, which results in observing a larger number of generations than previously computed maximum. Therefore, the current number of iterations t will achieve t_{max} before exploiting the budget expressed in available calls to the objective function. What is more, t can exceed t_{max}. This problem influences the algorithm's behavior as t/t_{max} is used to calculate ps and pd. When IDEbd was used with an ensemble of other algorithms, the problem was not noticeable, as other algorithms used the budget of objective function evaluations but did not increase the IDEbd iteration counter. The error is corrected in the following way. When the current iteration number exceeds the maximum, the maximum is set to the current number. This version will be called "corr. t_{max}".

In the version named "corr. $r1$ selection", two additional changes were made. The most important change is the correction of an error. When $r1$ is randomly drawn from the superior individuals, the code suggests the intention of its creators to exclude indexes used before. However, $r1$ is drawn from 1 to the size of the set of allowable indexes. In the corrected version, previously used indexes are excluded. The second difference is the removal of population sorting at the end of the iteration and after population size reduction. It logically does not change anything, but additional sorting slows the computation.

The EA4Eig mutation is complicated as it uses two thresholds, a random number, and two equations. Therefore, another proposed modification simplifies it. In the version called "simplified mut.", lines 25, 26, 27, and 29 from Figure 3 were removed, and only line 28 was used for the mutation.

Due to the aforementioned discrepancy between the description and the source code (where x_i is used instead of x_{best} in line 26 from Figure 3), the version that agrees with the paper was also examined. This version is named "best as base".

In the version called "without μ adapt." the population size adaptation was disabled, and the version "IDEbd μ adapt." uses the original IDEbd's population size adaptation instead of EA4Eig's.

The results of the corrections and modifications of IDEbd, together with the results of IDE and the best configuration from Table 2 (typeset in bold), are presented in Table 3. It can be observed that after correction of the number of iterations ("IDEbd, corr. t_{max}"), IDEbd is better than the IDEbd and jSO pair. The correction of $r1$ selection improved results further. Making the code closer to the intended version by correcting errors helped the algorithm. Additional simplification of the mutation ("simplified mut.") gave the best result in the table. On the other hand, disabling population size adaptation ("without μ adapt.") or using the adaptation from IDEbd strongly deteriorated results. The population size reduction from L-SHADE positively affects the performance of the algorithm. Using the best solution as a base in half of the search deteriorates results. This might be the reason why x_i was used in EA4Eig instead of the best individual. Further simplification of IDEbd to pure IDE also deteriorated performance.

The best version from Table 3 underwent several further simplifications. One of them (named $F = 0.8$) used constant $F = 0.8$ instead of randomly drawn F. The version "without μ adapt." turns off population size adaptation. The subsequent sim-

Table 3: The influence of corrections and basic modifications of IDEbd. The name typeset in bold is the best configuration from Table 2.

Algorithm	CEC points
IDEbd, corr. t_{max}, corr. $r1$ selection, simplified mut.	91238
IDEbd, corr. t_{max}, corr. $r1$ selection	89040
IDEbd, corr. t_{max}	86930
IDEbd+jSO	73561
IDEbd, corr. t_{max}, corr. $r1$ selection, best as base	71229
IDEbd, corr. t_{max}, corr. $r1$ selection, without μ adapt.	66874
IDEbd, corr. t_{max}, corr. $r1$ selection, IDEbd μ adapt.	65355
IDE	60575

Table 4: The results of modified IDEbd. The name typeset in bold is the best configuration from Table 3.

Algorithm	CEC points
no rand. feat. in x_{r3}	123553
simplified mut.	117326
$r1$ from superior	116255
only 1 stage mut.	105263
no Eig. cross.	99749
only 2 stage mut.	99336
without μ adapt.	99278
random $r1$	82800
$F = 0.8$	65394
only Eig. cross.	63047

plification ("no rand. feat. in x_{r3}") turns off additional perturbation of x_{r3} (line 19 from Figure 3). The version "$r1$ from superior" always draws $r1$ from the superior part of the population. Yet another version ("only 1 stage mut.") always assumes $b = i$, not only in the first stage of the search. On the contrary, in "only 2 stage mut." b is always randomly selected (like in the second stage of the algorithm). In the last examined modification, $r1$ is always randomly drawn, as in DE/rand. This modification is called "random $r1$". The results of the experiments are provided in Table 4. It can be observed that the best is the version without randomization in x_{r3} ("no rand. feat. in x_{r3}"). Not using Eigen crossover deteriorated results, but using only this mechanism was the last in the table. Therefore, the concept of random switching between binomial and Eigen crossover is good. The random selection of the scale factor is also much better than the typical constant value. The IDEbd idea of x_{r1} selection is better than selection from the superior and much better than random selection. Analogously, the x_b selection scheme is better than using x_i as x_b and better than using a random individual like in DE/rand/1. All other simplifications deteriorated results.

4 Parameter tuning

The source code of EA4Eig revealed that all parameters are set to "nice" values like 100, 0.5, and 0.1. This suggests that the algorithm was not tuned. It is interesting to examine whether parameters were set correctly for the CEC 2022 benchmark and which parameters are the most important. The search for the optimal set of parameters

Table 5: Parameter types, search ranges, default and tuned values.

Parameter	type	range	default	tuned
μ	i	20 – 400	100	76
p_{eig}	r	0 – 1	0.4	0.18
σ_N	r	0.05 – 0.3	0.1	0.26
μ ratio for Eig.	r	0 – 1	0.5	0.29
$fails_{th\_div}$	i	5 – 20	10	20
ps_{shape}	r	1 – 6	5	4.14
μ_{min}	i	4 – 50	10	19
t_{th_div}	r	1 – 4	2	2.31
ps_{min}	r	0.05 – 0.2	0.1	0.08
small Cauchy thres.	r	0 – 1	0.5	0.49
σ_C	r	0.05 – 0.3	0.1	0.19

Table 6: The influence of the tuning on the best algorithms. The name of best version from Table 4 is typeset in bold.

Algorithm	CEC points
simplified mut., no rand. feat. in x_{r3}, tuned	67449
IDEbd, tuned	57403
simplified mut., no rand. feat. in x_{r3}	56288
jSO, tuned	52283
IDEbd	49559
jSO	41020

was performed using the Irace (López-Ibáñez et al., 2016) package. Irace's budget was set to 5000 experiments. The whole benchmark (12 functions in 10 and 20 dimensions) is treated as an instance of Irace. The result seen by Irace comes from 5 independent runs of the algorithm under tuning. As competing algorithms are required to calculate CEC scores, NL-SHADE-RSP (Stanovov et al., 2021), SADE (Qin et al., 2009), RB_IPOP (Biedrzycki, 2017), and EA4Eig with IDEbd and jSO were used.

The parameter names of the best algorithm from Table 4, search ranges, default, and tuned parameter values are provided in Table 5. The tuned values in most cases differ significantly from the default ones, e.g., σ_N is nearly three times larger, and p_{eig} is more than two times smaller.

In addition to tuning the best configuration found so far, IDEbd and jSO were also tuned. These algorithms have more parameters than the simplified one. Therefore, they might be more susceptible to tuning. The results are provided in Table 6. It can be observed that all tuned versions are better than their default configurations. The tuned version of the simplified algorithm is the best in the table.

To find out which tuned parameter contributed the most to the superiority of the tuned algorithm, an ablation analysis (Fawcett and Hoos, 2016) is performed. The input for the ablation is two configurations: the default and tuned. The ablation starts from the default configuration; then, it generates a sequence of configurations where one parameter is taken from the target and the rest from the source. The best parameter is found and fixed, and the process repeats for the rest of the parameters. To find the winning parameter, each configuration is run 30 times, and its CEC points are calculated using NL-SHADE-RSP, SADE, RB_IPOP, and EA4Eig with IDEbd and jSO as

Table 7: The results of the ablation analysis of parameters of the best method.

parameter	CEC points	Δ points
default	72842	
σ_N	74377	1535
p_{eig}	74986	609
small Cauchy thres.	75163	177
t_{th_div}	75279	116
$fails_{th\_div}$	75279	0
ps_{shape}	75279	0
ps_{min}	74944	-335
σ_C	75072	128
μ_{min}	74799	-273
μ ratio for Eig.	74440	-359
μ	73821	-619

competitors.

The results of ablation analysis for the simplified algorithm are provided in Table 7. It can be observed that the tuning of σ_N gave the largest improvement. It seems that larger variability in random numbers generated for F and CR positively affects the performance of the algorithm. The influence of the rest of the parameters is marginal. Some tuned values (like μ) deteriorated results. This is possible because the tuning budget is limited due to high computational demand. From the algorithm simplification point of view, parameters $fails_{th\_div}$ and ps_{shape} are most interesting as they do not influence the number of gained points. When we look closer, the shape of the curves (line 7 in Figure 3) for both ps_{shape} values are identical for more than half of the iterations. Therefore, the results obtained should not be surprising. Considering $fails_{th\_div}$, when it is set to 0.1, it will change the algorithm's behavior when 10 percent of the consecutive iterations will not be able to find at least as good trial as the parent. It is implausible. Moreover, this mechanism is active only in the first half of the iterations. Therefore, the related code can be superfluous. To verify this hypothesis, the code responsible for detecting consecutive failures was removed (lines 9, 33, 36-45 in Figure 3), and the experiments were repeated. The results are identical to those with this component enabled.

5 Testing the simplified version

The results presented above compared different algorithm versions using the CEC 2022 ranking. It is worth examining whether a simplified algorithm is better than its more complicated versions according to other quality measures. Recently, Biedrzycki (2024a) proposed to compare algorithms using the weighted combination of three numbers. The first is the percentage of trials that found the global optimum (from 30 independent runs on all 12 functions and two considered dimensionalities). The second is the percentage of thresholds found. There are 51 thresholds spanned equidistant in a logarithmic scale between 10^3 and 10^{-8}. The third is the percentage of objective function evaluations that were not used, as the search is stopped when the global optimum is found. The results of the experiments are provided in Table 8. It can be observed that all results are similar. The tuned simplified method is the best in the table. Its untuned version finds the global optimum slightly more frequently than EA4Eig. It is the best

Table 8: The comparison of the original method and its two simplifications. The suffix 'tuned' means that the parameters of the algorithm were tuned. The data contains the percentage of trials that found the global optimum, the percentage of thresholds achieved by all trials, and the percentage of budget left.

Algorithm	% of optimum found	% of achieved thresholds	% of budget left
simplified mut., no rand. feat. in x_{r3}, tuned	38	49	26
EA4EigIDEbd, tuned	38	49	25
simplified mut., no rand. feat. in x_{r3}	37	48	27
EA4Eig	37	48	24

Table 9: CEC 2022 points obtained by EA4Eig and the simplified method for dimensionality: 2, 3, 5, 10, 20 and 40.

Algorithm	2 D	3 D	5 D	10 D	20 D	40 D	All
simp., no rand. feat. in x_{r3}	16716	15859	17749	16568	12762	10159	89813
EA4Eig	4884	5741	3851	5032	8838	11441	39787

in the table regarding speed. The results in the table confirm that the simplified version is not worse than the full version.

5.1 Testing on alternative benchmark

The CEC 2022 benchmark was used before because EA4Eig won the CEC 2022 competition. This section verifies the performance of simplified versions of EA4Eig on a different set of benchmark functions. As the EA4Eig is written in Matlab and the simplifications are in C++, the benchmark should support both programming languages. Such conditions are met by COCO (Hansen et al., 2021). The COCO includes a black-box optimization benchmarking (BBOB) test suite with 24 noiseless, single-objective functions. These functions are used here in 2, 3, 5, 10, 20, and 40 dimensions to examine how the performance of the considered algorithms scales with dimensionality. This set of dimensionalities was used in BBOB. The experimental setup used here is similar to the one used in Section 3. The search is bound by $\langle -5, 5\rangle^D$. The budget is set to $10000 \cdot D$. For each function, 30 independent runs on its first instance are performed, and the error level achieved by the best solution is recorded. Values below 10^{-8} are treated as 10^{-8}. All statistical tests and related rankings are calculated by the software provided by García et al. (2010).

The comparison of the simplified and original version using CEC 2022 points is provided in Table 9, and the percentage measure is provided in Table 10. It can be observed that the simplified method is the best when considering the combined scores in all considered dimensionalities. Both the performance measures agree that the simplified method is much better in dimensionalities 2, 3, 5, and 10, but degraded performance is visible in 40 D, especially when considering the percentage measure. Therefore, it is worth going back in the simplification chain to check at what point the quality dropped for 40 D problems. Table 11 compares the original method and its four simplifications discussed in Section 3.3. Besides the percentage measure, the table contains the results of well-established ranking methods and statistical tests recommended

Table 10: The comparison of the simplified method and EA4Eig using the percentage measure where each triplet contains the percentage of trials that found the global optimum, the percentage of thresholds achieved by all trials, and the percentage of budget left. The name of the simplified method (simp., no rand. feat. in xr3) was abbreviated to "simp.".

Alg.	2 D	3 D	5 D	10 D	20 D	40 D	All
simp.	93 96 68	87 93 61	78 87 53	51 70 36	20 45 15	15 38 13	57 71 41
EA4Eig	48 70 34	43 61 31	41 58 27	37 53 23	37 54 22	25 44 12	38 57 25

Table 11: The comparison of EA4Eig and its four simplifications using percentage of trials when the global optimum was found and rankings: Friedman, Aligned Friedman, and Quade on BBOB functions in 40 dimensions.

Algorithm	Friedman	Aligned F.	Quade	% opt.
IDEbd+jSO	2.42	41.92	2.31	31
IDEbd, corr. t_{max}, corr. $r1$ selection	2.77	57.60	2.75	26
EA4Eig	2.81	59.52	2.82	25
IDEbd,. . ., corr. $r1$ selection, simp. m.	3.23	68.19	3.14	22
IDEbd,. . ., simp. m., no rand. feat. in x_{r3}	3.77	75.27	3.98	15
p-value	3.5E-2	6.1E-4	1.9E-2	–

by Derrac et al. (2011) for comparing evolutionary algorithms, i.e., Friedman, Aligned Friedman, and Quade. All ranking methods agreed that two simplified versions are better than the original. The best in 40 D is the combination of IDEbd and jSO, which used only half of the original components. Even IDEbd with corrections discussed in Section 3 ranks better than the original. It follows that removing additional randomness introduced to x_{r3} and used in switching mutation operator helps in 2, 3, 5, and 10 D, but it degrades performance in 40 D. All statistical tests agreed that at least one pair of algorithms has significantly different performance. For further analysis, taking IDEbd+jSO as a control method, Holm-corrected p-values were calculated. The results are provided in Table 12. Assuming a level of significance $\alpha = 0.05$, the Quade test could not reject the null hypothesis. The corrected p-value from the Friedman test identified the strongest simplification as inferior to IDEbd+jSO. The results of aligned Friedman additionally identified the second most substantial simplification as a worse method. The rest of the comparisons were far from rejecting the null hypothesis. Therefore, the simplified version named IDEbd+jSO is not worse than the original. The first step of IDEbd simplifications (IDEbd, corr. t_{max}, corr. $r1$ selection) gave results similar to the best one.

Table 12: Adjusted p-values (Holm) taking IDEbd+jSO as a control method on BBOB functions in 40 dimensions.

Algorithm	Friedman	Aligned F.	Quade
IDEbd,. . ., simp. m., no rand. feat. in x_{r3}	0.01	0.004	0.10
IDEbd,. . ., corr. $r1$ selection, simp. m.	0.23	0.027	0.78
EA4Eig	0.77	0.169	0.98
IDEbd, corr. t_{max}, corr. $r1$ selection	0.77	0.169	0.98

The results on the BBOB benchmark (considering all dimensionalities together) confirmed that the strongest simplification is the best. However, this method's advantage decreases with the increase in dimensionality. After focusing on 40 dimensions, the results of the four ranking methods consistently showed a decrease in the quality of the most simplified method. Still, two proposed simplifications gave better results than the original. The most complicated of these two uses only about half of the original set of internal components. The second one uses about one-quarter of the components.

6 Conclusions

This paper analyzed the EA4Eig algorithm which won the CEC 2022 SOBC competition. It was shown which of its four main components are the most important and which only deteriorated results and can be disabled. During the analysis, some errors in the source code were found. Some of them improved the results of the method, and some deteriorated it. It was also found that the algorithm can be simplified without quality loss. Several simplified versions of the algorithm were examined, varying in the degree of simplification. It was shown that the version that is more complicated than the IDE algorithm and simpler than IDEbd (with crossover from CoBiDE and population size reduction from L-SHADE) is the best. Due to computational efficiency reasons, the code of EA4Eig with two components, namely IDEbd and jSO, was rewritten from Matlab to C++. The simplification is visible in the number of nontrivial lines of code. The original algorithm has 716 lines in Matlab, and the final simplified version only has 244 lines in C++. The code is available online (Biedrzycki, 2024b). The parameters of the simplified method, jSO, and IDEbd, were tuned. All tuned versions were better than their default counterparts. The tuned version of the simplified algorithm was the best. The ablation analysis of the tuned parameters showed that σ_N and p_{eig}, whose values were hidden in the original code, contributed the most to the success of the tuned version. The superiority of the simplified version on the CEC 2022 benchmark was confirmed using quality measures based on the percentage of the optimal solutions found and by the CEC 2022 ranking. The simplified version is also better than the original one on BBOB when considering all dimensionalities together, but in 40 dimensions alone a more gentle simplification is superior.

Programming is inextricably linked with making errors that stay hidden in the source code. It is very hard to eliminate them just by code review when the code is large and complicated, like in the case of EA4Eig. Finding these errors was possible by rewriting the program into another language and trying to obtain precisely the same results as the original. Even during the rewrite process, it was impossible to eliminate all code that did not affect the results. The last fragments of unnecessary code were found by a critical analysis of the parameters impact on the algorithm's results.

The implementations of contemporary metaheuristics tend to be overcomplicated and can frequently be simplified without quality loss, gaining readability and computational speed. As the community is overflood by numerous versions of algorithms, their analysis, simplification, and bug removal are more important than the development of yet another algorithm.

Future work will determine how the simplified versions performs on other benchmarks and real-world applications.

References

Biedrzycki, R. (2017). A version of IPOP-CMA-ES algorithm with midpoint for CEC 2017 single objective bound constrained problems. In *IEEE Congress on Evolutionary*

Computation (CEC), pages 1489–1494.

Biedrzycki, R. (2021). Comparison with state-of-the-art: Traps and pitfalls. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 863–870.

Biedrzycki, R. (2024a). Revisiting CEC 2022 ranking: a new ranking method and influence of parameter tuning. *Swarm and Evolutionary Computation*, 89:101623.

Biedrzycki, R. (2024b). Supplementary materials for the paper: "Analysis and simplification of the winner of the CEC 2022 optimization competition on single objective bound constrained search". `https://staff.elka.pw.edu.pl/%7Erbiedrzy/publ/EA4EigSimpl.zip`. (Accessed: 1 June 2024).

Biedrzycki, R., Arabas, J., and Warchulski, E. (2022). A version of NL-SHADE-RSP algorithm with midpoint for CEC 2022 single objective bound constrained problems. In *IEEE Congress on Evolutionary Computation (CEC)*.

Brest, J., Maučec, M. S., and Bošković, B. (2017). Single objective real-parameter optimization: Algorithm jSO. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1311–1318.

Bujok, P. and Kolenovsky, P. (2022). Eigen crossover in cooperative model of evolutionary algorithms applied to CEC 2022 single objective numerical optimisation. In *IEEE Congress on Evolutionary Computation (CEC)*, page 1–8. IEEE Press.

Bujok, P. and Tvrdík, J. (2017). Enhanced individual-dependent differential evolution with population size adaptation. In *IEEE Congress on Evolutionary Computation (CEC)*, page 1358–1365. IEEE Press.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

Ding, H., Gu, Y., Wu, H., and Zhou, J. (2022). NL-SOMA-CLP for real parameter single objective bound constrained optimization. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 5–6, New York, NY, USA. Association for Computing Machinery.

Fawcett, C. and Hoos, H. H. (2016). Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458.

García, S., Fernández, A., Luengo, J., and Herrera, F. (2010). Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. *Information Sciences*, 180(10):2044–2064. Special Issue on Intelligent Distributed Information Systems.

Gu, Y., Ding, H., Wu, H., and Zhou, J. (2022). Opposite learning and multi-migrating strategy-based self-organizing migrating algorithm with the convergence monitoring mechanism. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 7–8, New York, NY, USA. Association for Computing Machinery.

Guo, S.-M. and Yang, C.-C. (2015). Enhancing differential evolution utilizing eigenvector-based crossover operator. *IEEE Transactions on Evolutionary Computation*, 19(1):31–49.

Hansen, N., Auger, A., Ros, R., Mersmann, O., Tušar, T., and Brockhoff, D. (2021). COCO: A platform for comparing continuous optimizers in a black-box setting. *Optimization Methods and Software*, 36:114–144.

Hansen, N. and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Kolenovsky, P. and Bujok, P. (2022). An adaptive variant of jSO with multiple crossover strategies employing Eigen transformation. In *IEEE Congress on Evolutionary Computation (CEC)*.

Kumar, A., Price, K. V., Mohamed, A. W., Hadi, A. A., and Suganthan, P. (2022). Problem definitions and evaluation criteria for the CEC 2022 special session and competition on single objective bound constrained numerical optimization. Technical report, Nanyang Technol. Univ., Singapore.

López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., and Birattari, M. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Ning, Y., Jian, D., Wu, H., and Zhou, J. (2022). Zeroth-order covariance matrix adaptation evolution strategy for single objective bound constrained numerical optimization competition. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 9–10, New York, NY, USA. Association for Computing Machinery.

Pillay, N. and Gerber, M. (2022). GECCO 2022 - competition on single objective bound constrained numerical optimization - submission. Technical report, Department of Computer Science, University of Pretoria Gauteng, South Africa.

Piotrowski, A. P. and Napiorkowski, J. J. (2018). Some metaheuristics should be simplified. *Information Sciences*, 427:32–62.

Price, K. V., Kumar, A., and Suganthan, P. (2023). Trial-based dominance for comparing both the speed and accuracy of stochastic optimizers with standard non-parametric tests. *Swarm and Evolutionary Computation*, 78:101287.

Qin, A. K., Huang, V. L., and Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Trans. Evol. Comput.*, 13(2):398–417.

Sallam, K. M., Abdel-Basset, M., El-Abd, M., and Wagdy, A. (2022). IMODEII: an improved IMODE algorithm based on the reinforcement learning. In *IEEE Congress on Evolutionary Computation (CEC)*.

Stanovov, V., Akhmedova, S., and Semenkin, E. (2021). NL-SHADE-RSP algorithm with adaptive archive and selective pressure for CEC 2021 numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 809–816.

Stanovov, V., Akhmedova, S., and Semenkin, E. (2022). NL-SHADE-LBC algorithm with linear parameter adaptation bias change for CEC 2022 numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC)*.

Suganthan, P. (2024). Repository for CEC 2022 special session and competition on single objective bound constrained numerical optimization. https://github.com/P-N-Suganthan/2022-SO-BO/tree/main. (Accessed: 1 October 2023).

Sun, B., Li, W., and Huang, Y. (2022a). Performance of composite PPSO on single objective bound constrained numerical optimization problems of CEC 2022. In *IEEE Congress on Evolutionary Computation (CEC)*.

Sun, B., Sun, Y., and Li, W. (2022b). Multiple topology SHADE with tolerance-based composite framework for CEC2022 single objective bound constrained numerical optimization. In *IEEE Congress on Evolutionary Computation (CEC)*.

Tanabe, R. and Fukunaga, A. S. (2014). Improving the search performance of SHADE using linear population size reduction. In *IEEE Congress on Evolutionary Computation (CEC)*, pages 1658–1665.

Tang, L., Dong, Y., and Liu, J. (2015). Differential evolution with an individual-dependent mechanism. *IEEE Transactions on Evolutionary Computation*, 19(4):560–574.

Tseng, T.-R. (2022). Improvement of multi-population ML-SHADE. Technical report, Submission to GECCO 2022 - Competition on Single Objective Bound Constrained Numerical Optimization.

Van Cuong, L., Bao, N. N., Phuong, N. K., and Binh, H. T. T. (2022). Dynamic perturbation for population diversity management in differential evolution. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, page 391–394, New York, NY, USA. Association for Computing Machinery.

Wang, Y., Li, H.-X., Huang, T., and Li, L. (2014). Differential evolution based on covariance matrix learning and bimodal distribution parameter setting. *Appl. Soft Comput.*, 18(C):232–247.

Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6):80–83.