Algorithm Results on the New COCO Test Problems

Technical Report IJS-DP 12992

Delovno poročilo IJS-DP 12992

Tea Tušar
Computational Intelligence Group
Department of Intelligent Systems
Jožef Stefan Institute
Jamova cesta 39, SI-1000 Ljubljana, Slovenia
tea.tusar@ijs.si

December 2019

Abstract

Recently, a new suite of bi-objective mixed-integer test problems (`bbob-biobj-mixint`) was introduced and integrated into the COCO platform (`github.com/numbbo/coco/`). However, no algorithm data was provided yet for this suite. This report presents a new algorithm for solving mixed-integer problems called BBDEMO (Black Box Differential Evolution for Multiobjective Optimization). It extends the Black Box Differential Evolution algorithm to multi-objective problems. We show the performance of BBDEMO and a baseline random search on the problems from the `bbob-biobj-mixint` suite. The results of both algorithms are available to be included in future benchmarking studies using the `bbob-biobj-mixint` suite.

Contents

1 Introduction

The development of new optimization algorithms requires testing, or *benchmarking*, them on a number of test problems. In the field of multi-objective optimization this is almost exclusively done on two problem suites, DTLZ [1] and WFG [2]. The specialization to these two suites of test problems has been so severe that the state-of-the-art multi-objective optimization algorithms are in all likelihood overfitting to these problems. This is particularly concerning since the DTLZ and WFG problems possess some artificial properties that are useful when designing test suites, but are not likely to occur in real-world problems. Therefore, we cannot expect algorithms that perform well on such problems to also perform well on real-world problems. Furthermore, both suites consist of only continuous problems, but we know that many real-world optimization problems are in fact mixed-integer (contain continuous as well as integer variables). The lack of test suites with mixed-integer problems is hindering research in this direction.

In addition, the comparison methodology most often used in contemporary benchmark studies does not provide a lot of useful information. Although the problems from the DTLZ and WFG test suites are scalable, they are only rarely used with more than the 'default' problem dimension. The algorithms are compared after an arbitrary large number of function evaluations and their convergence graphs are only seldom shown. All this is steering the research efforts of the community away from being useful in efficiently solving the real-world multi-objective optimization problems.

Our previous work was focused on introducing new, real-world-inspired problem suites and integrating them in the COCO (Comparing Continuous Optimizers, `https://github.com/numbbo/coco`) platform [3]. COCO makes it easy to inspect the results of multiple algorithms on separate functions, groups of functions with similar properties and even all functions of the suite by aggregating algorithm results. Automatically produced plots for different dimensions showing the convergence of the algorithms facilitate understanding of the algorithm performance and learning lessons from test problems that can then be employed to solving real-world problems of various properties and dimensions. While two of the newly proposed suites (`rw-top-trumps-biobj` and `rw-mario-gan-biobj` [4]) implement real-world problems and are therefore a welcome contribution to the research community, the biggest niche in multi-objective optimization benchmarking is targeted by the `bbob-biobj-mixint` suite [5] that consists of scalable mixed-integer bi-objective problems. The paper [5] presented single- and bi-objective suites of mixed-integer problems, but did not provide any algorithm data for the bi-objective ones. This is rectified by the present report.

In this work we introduce a new algorithm for solving multi-objective optimization problems called BBDEMO (Black Box Differential Evolution for Multiobjective Optimization) that was designed with mixed-integer problems in mind. We show its results on the `bbob-biobj-mixint` suite using the random search algorithm as baseline. In addition to presenting BBDEMO, the main purpose of this study is to provide algorithm data for this problem suite and share it with the research community to facilitate future benchmark studies.

The remainder of this report is structured as follows. After some preliminary information on multi-objective optimization problems and related work in Section 2, Section 3 details the BBDEMO algorithm. Then, Section 4 shortly summarizes the properties of the latest suites of bi-objective test problems in COCO. The experiments using the `bbob-biobj-mixint` suite are described in Section 5, while their results are presented in Section 6. The paper ends with concluding remarks in Section 7.

2 Preliminaries

We first give some background information about multi-objective optimization problems, the dominance relation and hypervolume indicator. Then we place the BBDEMO algorithm in the context of related work.

2.1 Multi-objective optimization problems

An m-objective mixed-integer problem can be formally defined as

$$
\begin{aligned}
& \min\left(f_1(x), \ldots, f_m(x)\right) & \\
\text{where} \quad & f_j : \mathbb{Z}^k \times \mathbb{R}^{(n-k)} \to \mathbb{R} & j = 1, \ldots, m \\
& f_j : x \mapsto f_j(x) & j = 1, \ldots, m \\
& x_i \in [x_i^{\min}, x_i^{\max}] \cap \mathbb{Z} & i = 1, \ldots, k \\
& x_i \in D_i \subseteq \mathbb{R}^{(n-k)} & i = k+1, \ldots, n
\end{aligned}
$$

The problem entails minimization of the given function(s) where the first k variables are integer and the last $n - k$ continuous. If $k = 0$, the problem is continuous and if $k = n$, it is integer. A continuous variable i can be box-constrained, i.e., $D_i = [x_i^{\min}, x_i^{\max}]$, or (partially) unconstrained. If $m = 1$, the optimization problem is single-objective and if $m \geq 2$, it is multi-objective.

Solutions to a multi-objective optimization problem can be compared using the *dominance relation*. We say that solution x *dominates* solution y, $x \prec y$, if x is not worse than y on any objective and better on at least one. If neither $x \prec y$ nor $y \prec x$, the solutions are *mutually non-dominating*. If a solution has no dominating solutions, it is *non-dominated*. The set of all non-dominating solutions is called the *Pareto set*. Its image in the objective space is the *Pareto front*.

Quality indicators can be used to assess and compare sets of solutions to multi-objective optimization problems. Among the many available ones, the *hypervolume indicator* [6] is the only one that is strictly Pareto-compliant [7]. For a given set of points $\mathcal{A}$ and a reference point, it gives the volume of the portion of the objective space that is dominated by $\mathcal{A}$ and limited by the reference point. Given the set $\mathcal{A}$, the *hypervolume improvement* of solution a equals to the increase of hypervolume if a is added to $\mathcal{A}$.

2.2 Related work

Differential Evolution (DE) [8] is one of the most well-known single-objective algorithms. It became very popular due to its simplicity and efficiency and has been the subject of many studies and extensions since its first introduction. One of the prominent extensions that adapted DE to solve multi-objective optimization problems is Differential Evolution for Multiobjective Optimization (DEMO) [9], which combines the search mechanism of DE with the environmental selection of NSGA-II [10]. However, DE also has known drawbacks, for example, its performance is not invariant under an orthogonal rotation. In order to rectify this (and some other issues of the original algorithm), one of the original authors of DE recently presented a new, simplified version of DE called Black Box Differential Evolution (BBDE) [11], whose performance is invariant under a number of transformations, including an orthogonal rotation.

The main idea behind the BBDEMO algorithm is to extend BBDE to multi-objective optimization in a similar way in which DE was extended to DEMO. However, instead of requiring an elaborate environmental selection, the problem of minimizing two objectives is transformed to the single-objective problem of maximizing the *uncrowded hypervolume improvement*, an extension of the hypervolume improvement presented in [12].

3 The BBDEMO algorithm

The BBDEMO algorithm is conceived as a multi-objective variant of the BBDE algorithm [11]. It generates solutions similarly to the single-objective BBDE algorithm and compares them using the uncrowded hypervolume improvement [12]. To avoid premature convergence to a local optimum, perturbation of solutions is performed whenever a lack of diversity in the population is detected. The BBDEMO algorithm is outlined in Algorithm 1, while the separate procedures for offspring creation, repair and perturbation as well as for updating the parent population are given in Algorithms 2, 3, 4 and 5, respectively.

Algorithm 1: BBDEMO

Input: Population size, stopping criterion, perturbation parameters
Output: Pareto set approximation
Create initial population of random solutions $\mathcal{P}$
Evaluate all solutions in $\mathcal{P}$
Create initial archive of non-dominated solutions $\mathcal{A}$
while *stopping criterion not met* **do**
    $b \leftarrow$ the best solution from $\mathcal{P}$
    Initialize candidate population $\mathcal{C} \leftarrow \{\}$
    forall *parents* $p \in \mathcal{P}$ **do**
        Create candidate c from parent p and best solution b (see Algorithm 2)
        Repair the candidate c if necessary (see Algorithm 3)
        Add c to $\mathcal{C}$
    end forall
    Perturb the candidates $c \in \mathcal{C}$ (see Algorithm 4)
    Evaluate all candidates $c \in \mathcal{C}$
    Update the parent population $\mathcal{P}$ and archive $\mathcal{A}$ (see Algorithm 5)
end while
return $\mathcal{P}$

The BBDEMO algorithm starts by filling the initial parent population $\mathcal{P}$ with random solutions. Then, the solutions are evaluated and added to the archive of non-dominated solutions $\mathcal{A}$. Further steps are performed until a stopping criterion is met, which usually depends on to the number of performed evaluations. In each of the steps, a population of candidates $\mathcal{C}$ is first created from the parent population $\mathcal{P}$. It is then perturbed and evaluated. Finally, the parent population and the archive are updated with the candidates that outperform their parents. When the algorithms concludes, it returns the last parent population $\mathcal{P}$.

The creation of candidates is detailed in Algorithm 2. It is a variant of the BBDE/target-to-best strategy [13], where the candidate is computed as a linear combination of the parent p, the best solution in the population b and two different random solutions p_1 and p_2. There are no additional constraints on the solutions p_1 and p_2, which means one (or both) of them could equal the parent and/or the best solution. The factors F_1 and F_2 used in the linear combination are sampled from the log-normal distribution with $\exp(\mathcal{N}(0.1, 0.5))$. This parameter setting favors larger values compared to the 'default' log-normal distribution $\exp(\mathcal{N}(0, 1))$.

The candidate creation ignores the bounds of the search space, therefore it can (and often does) happen that the candidate falls out of these bounds. The repair procedure that is normally used in DE simply crops all the values that exceed the bounds to those bounds. However, such a repair disturbs the rotational invariance of the candidate creation [13]. To amend this issue, BBDEMO uses the repair procedure presented in Algorithm 3, which makes sure that the direction of the candidate (from the perspective of the parent) remains intact after the repair. The proce-

Algorithm 2: Candidate creation

Input: Population $\mathcal{P}$, parent p, best solution b
Output: Candidate solution

$p_1, p_2 \leftarrow$ two different random solutions from $\mathcal{P}$
$F_1, F_2 \leftarrow$ two random values sampled from $X_F \sim \exp(\mathcal{N}(0.1, 0.5))$
$c \leftarrow p + F_1(b - p) + F_2(p_1 - p_2)$
return c

dure first collects the coefficients that would crop the separate candidate values to the bounds of the search space. If the candidate is entirely contained in the search space, nothing needs to be done and the procedure ends there. However, if at least one of the coefficients differs from 1, the repair is triggered and the candidate is scaled using the parent as the origin and the minimal value of the repair coefficients for the scaling factor.

Algorithm 3: Candidate repair

Input: Candidate c, parent p
Output: Repaired candidate

Initialize repair vector $r \leftarrow \mathbf{1}^n$
forall *variables* $i \in \{1, \ldots, n\}$ *where* $c_i \neq p_i$ **do**
    if $c_i < x_i^{\min}$ **then**
        $r_i \leftarrow \frac{p_i - x_i^{\min}}{p_i - c_i}$
    else if $c_i > x_i^{\max}$ **then**
        $r_i \leftarrow \frac{x_i^{\max} - p_i}{c_i - p_i}$
    end if
end forall
if $r \neq \mathbf{1}^n$ **then**
    $c \leftarrow p + (c - p) \min_i r_i$
end if
return c

One known issue of DE-like candidate creation is that the solutions often lose diversity (all solutions in the population eventually share the same value of one or more variables). Because candidates are created by linear combination of existing solutions, once this happens, the diversity cannot be recovered. This is exacerbated by the BBDE approach, which gets rid of the mutation step as it is rotationally invariant, and further by the target-to-best strategy which accelerates the convergence to the best solution in the population. Therefore, BBDEMO includes a perturbation procedure (that could also be regarded as mutation), with the aim of adding diversity to the population. At the beginning of this procedure (see Algorithm 4), the need to perform perturbation is checked. This means that for each of the variables, we compute the standard deviation of the variable values in the candidate population. If the standard deviation is smaller than some threshold, the variable is stored to a list of variables that require perturbation. If after going through all of the variables this list is empty, the procedure stops without preforming any perturbation. If, on the other hand, this list is non-empty, the perturbation is applied to the variables with lacking diversity using two helper values. The first is a random value sampled from the normal distribution $\mathcal{N}(0, 1)$, while the second one depends on whether the variable is an integer one or a continuous one. The two cases are handled separately to assure that integer variables (which are also most prone to diversity loss) actually get perturbed. If at the end of the

Algorithm 4: Perturbation of the candidate population

Input: Candidate population $\mathcal{C}$, perturbation parameters $\bar{s}$, d_{int}, d_{cont}
Output: Perturbed candidate population

Initialize set of variables to be perturbed $\mathcal{I} \leftarrow \{\}$
forall *variables* $i \in \{1, \ldots, n\}$ **do**
    $\mathcal{V} \leftarrow$ values of the i-th variable for all $c \in \mathcal{C}$
    $s \leftarrow$ standard deviation of $\mathcal{V}$
    if $s \leq \bar{s}$ **then**
        Add i to $\mathcal{I}$
    end if
end forall
if $\mathcal{I} \neq \{\}$ **then**
    forall $c \in \mathcal{C}$ **do**
        forall *variables* $i \in \mathcal{I}$ **do**
            $r \leftarrow$ random value sampled from $X_r \sim \mathcal{N}(0, 1)$
            if i *is an integer variable* **then**
                $c_i \leftarrow c_i + r d_{\text{int}}$
            else
                $c_i \leftarrow c_i + r d_{\text{cont}}(x_i^{\max} - x_i^{\min})$
            end if
        end forall
        Repair the candidate c if necessary
    end forall
end if
return c

perturbation a candidate falls out of the bounds of the search space, it is repaired.

After all the candidates are evaluated, the parent population needs to be updated. As depicted in Algorithm 5, this is done by comparing each candidate c with its parent p and determining which of them is better. A helper variable u is used to store whether the candidate should replace its parent. First, the candidate and parent are compared with regard to the dominance relation. If the candidate dominates the parent, u is set to 1. If the parent dominates the candidate, u is given the value of 0. If neither of the two relations holds, the parent and candidate are compared with regard to how much they would contribute to the archive. We use the measure known as the uncrowded hypervolume improvement [12] denoted here by h to assess their contribution. If a solution is not dominated by the archive, its h value equals the 'usual' hypervolume improvement. If, on the other hand, the solution is dominated by the archive, h amounts to the negative distance to the archive.

The comparison with the uncrowded hypervolume improvement occurs as follows. First, a copy of the archive called $\mathcal{A}'$ is created and used to compute the uncrowded hypervolume improvement for the parent p. If $h_{\mathcal{A}'}(p)$ is positive, this means that the parent is included in the archive. In order to make the comparison with the candidate fair, we remove the parent from the temporary archive. Next, we compute the uncrowded hypervolume improvement for the candidate with regard to the temporary archive $\mathcal{A}'$. The variable u takes on the value 1 if the candidate's uncrowded hypervolume improvement is greater than that of the parent and 0 otherwise. If at the end of the procedure u equals 1, the candidate is better than the parent and replaces it. It is also added to the original archive $\mathcal{A}$ (note that such an addition is effective only if the candidate is not dominated by the archive). Because at the end the true archive is updated, the next candidate-parent pair will be compared with regard to this new archive. This ensures

Algorithm 5: Update of the parent population and the archive

Input: Parent population $\mathcal{P}$, candidate population $\mathcal{C}$, archive $\mathcal{A}$
Output: Updated parent population

1 **forall** *pairs* $(p \in \mathcal{P}, c \in \mathcal{C})$ **do**
2 **if** $c \prec p$ **then**
3 $u \leftarrow 1$
4 **else if** $p \prec c$ **then**
5 $u \leftarrow 0$
6 **else**
7 $\mathcal{A}' \leftarrow$ a copy of the archive $\mathcal{A}$
8 $h_{\mathcal{A}'}(p) \leftarrow$ the uncrowded hypervolume improvement of p with regard to $\mathcal{A}'$
9 **if** $h_{\mathcal{A}'}(p) > 0$ **then**
10 $\mathcal{A}' \leftarrow \mathcal{A}'$ without the parent p
11 **end if**
12 $h_{\mathcal{A}'}(c) \leftarrow$ the uncrowded hypervolume improvement of c with regard to $\mathcal{A}'$
13 **if** $h_{\mathcal{A}'}(c) > h_{\mathcal{A}'}(p)$ **then**
14 $u \leftarrow 1$
15 **else**
16 $u \leftarrow 0$
17 **end if**
18 **end if**
19 **if** $u = 1$ **then**
20 $p \leftarrow c$
21 Add c to $\mathcal{A}$
22 **end if**
23 **end forall**

that each comparison is performed with maximal information about the state of the archive.

The updating procedure of BBDEMO seems needlessly complex. Note that the steps in lines 2–6 could be removed without affecting its outcome. However, dominance comparisons between two solutions are computationally cheaper than handling the archive and computing the uncrowded hypervolume improvement, therefore we added these additional steps to make the algorithm more efficient. Similarly holds for the check performed in line 9.

The implementation of the BBDEMO algorithm is available in Python from `https://dis.ijs.si/tea/research.htm`.

4 The new COCO bi-objective test suites

In the last few years, the COCO platform has been extended to include the following new bi-objective test suites[1]:

- The `bbob-biobj-ext` [14] suite extends the `bbob-biobj` suite by adding combinations of single-objective functions previously missing from the suite. It contains 92 continuous functions in six dimensions (2, 3, 5, 10, 20, 40) and 15 instances.
- The `bbob-biobj-mixint` [5] suite consists of mixed-integer problems created by partially discretizing the continuous problems of the `bbob-biobj-ext` suite. It also contains 92 func-

[1] COCO has also recently acquired a number of new single-objective suites, but we omit them here since this report is focused on bi-objective problems.

tions in six dimensions and 15 instances, however the dimensions are higher (5, 10, 20, 40, 80, 160) since 4/5 of the variables are integer.

- The `rw-mario-gan-biobj` [4] suite contains real-world problems of finding playable levels for the Mario platform game. It incorporates ten continuous functions in four dimensions (10, 20, 30, 40) and seven instances. Two of the functions are computed directly, while the evaluation of the remaining eight functions requires simulations of game play and therefore takes a longer time to evaluate.
- The `rw-top-trumps-biobj` [4] suite comprises integer real-world problems of fine-tuning the values on the cards of a Top Trumps deck. It contains three functions in four dimensions (88, 128, 168, 208) and five instances. The first function is computed directly, while the remaining two are calculated based on the simulation results.

The first two suites are available in the COCO platform at `https://github.com/numbbo/coco`, while the last two can be accessed from `https://github.com/ttusar/coco/tree/gbea`.

In the remainder of this report, we limit ourselves to the `bbob-biobj-mixint` suite.

5 Experimental setup

The experiments were performed on the first five instances of all 92 functions in four dimensions of the `bbob-biobj-mixint` test suite. The two algorithms were run for a fixed budget of $2000n$ function evaluations where $n \in \{5, 10, 20, 40\}$ is the problem dimension.

While random search has no additional parameters, the parameters of BBDEMO were set as follows:

- Population size $= 20$,
- Diversity threshold $\bar{s} = 10^{-3}$,
- Perturbation parameter for integer variables $d_{\text{int}} = 3$,
- Perturbation parameter for continuous variables $d_{\text{cont}} = 10^{-2}$.

6 Results

We first show aggregated results in the form of ECDF (Empirical Cumulative Distribution Function) plots as produced by COCO and then view the results on some of the selected separate problems in more detail.

6.1 Aggregated results

Figure 1 shows the aggregated results over all functions and the first five instances for each of the four dimensions.

We can see that at the beginning, BBDEMO performs similarly to random search, which is understandable, since its first population is created randomly. Then, as the search progresses, BBDEMO moves away from random search. The difference between the two algorithms increases with increasing dimension, due to random search performing poorly in higher dimensions. We also see that the chosen budget of $2000n$ function evaluations is not necessarily enough for BBDEMO to converge, especially on 40-D problems.

We know the optima of each objective function separately (the two extreme points), but not the combined Pareto fronts, i.e., the true optima for the problems in the `bbob-biobj-mixint`

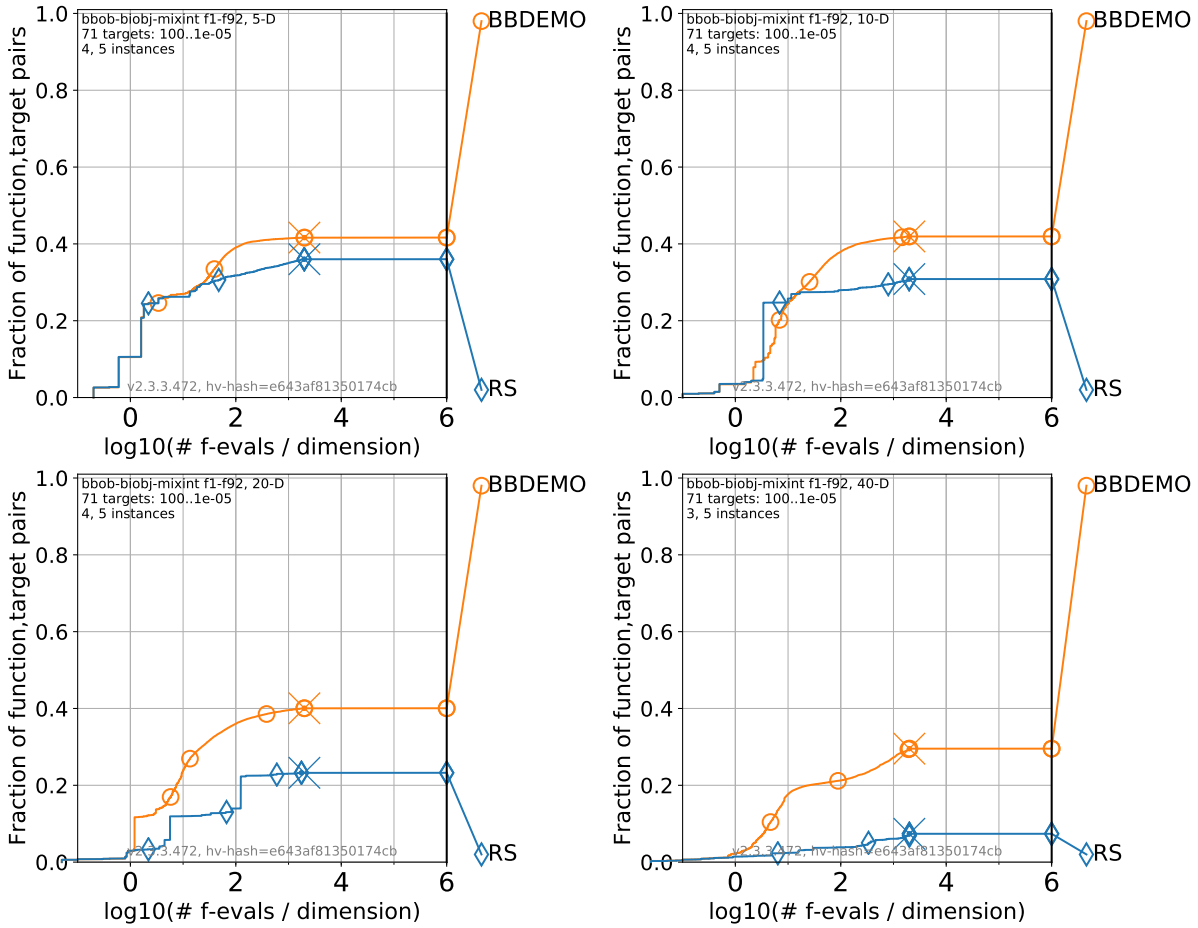

Figure 1: Empirical cumulative distribution of the number of objective function evaluations divided by dimension for 72 targets with target precision in $\{10^{-5}, 10^{-4.9}, 10^{-4.8}, \ldots, 10^{-1.9}, 10^{2}\}$ for all functions in dimensions $n \in \{5, 10, 20, 40\}$.

test suite are unknown. Therefore, these plots show the achievement of performance targets relative to the unachievable optimum of reaching the ideal point. Since the true optima could be quite different from one function to the next (and sometime this holds for different instances of the same function), it is hard to compare among themselves these performances over different functions.

6.2 Separate results

In order to learn more about the performance of BBDEMO, we look at the Pareto front approximations achieved by both algorithms for some selected separate functions.

Figure 2 shows the performance of BBDEMO and random search in the objective space for the first instance of function f_1 (the double sphere function). We can see that BBDEMO converged quite well in lower dimensions (the front is smooth and we know that this function has no local optima that could trap an optimization algorithm, so it must have converged to the Pareto front). We also see how the distance between the fronts attained by the two algorithms increases with increased problem dimension.

In some of the 5-D functions, random search is able to get very close to the front discovered by BBDEMO. An example of this is shown in Figure 3, where we see instances 1 and 2 of the f_{11} (Separate ellipsoid/Separate ellipsoid) function. On a few of the harder functions, for example f_{87} (101-peaks Gallagher/21-peaks Gallagher) presented in Figure 4, the fronts by random search

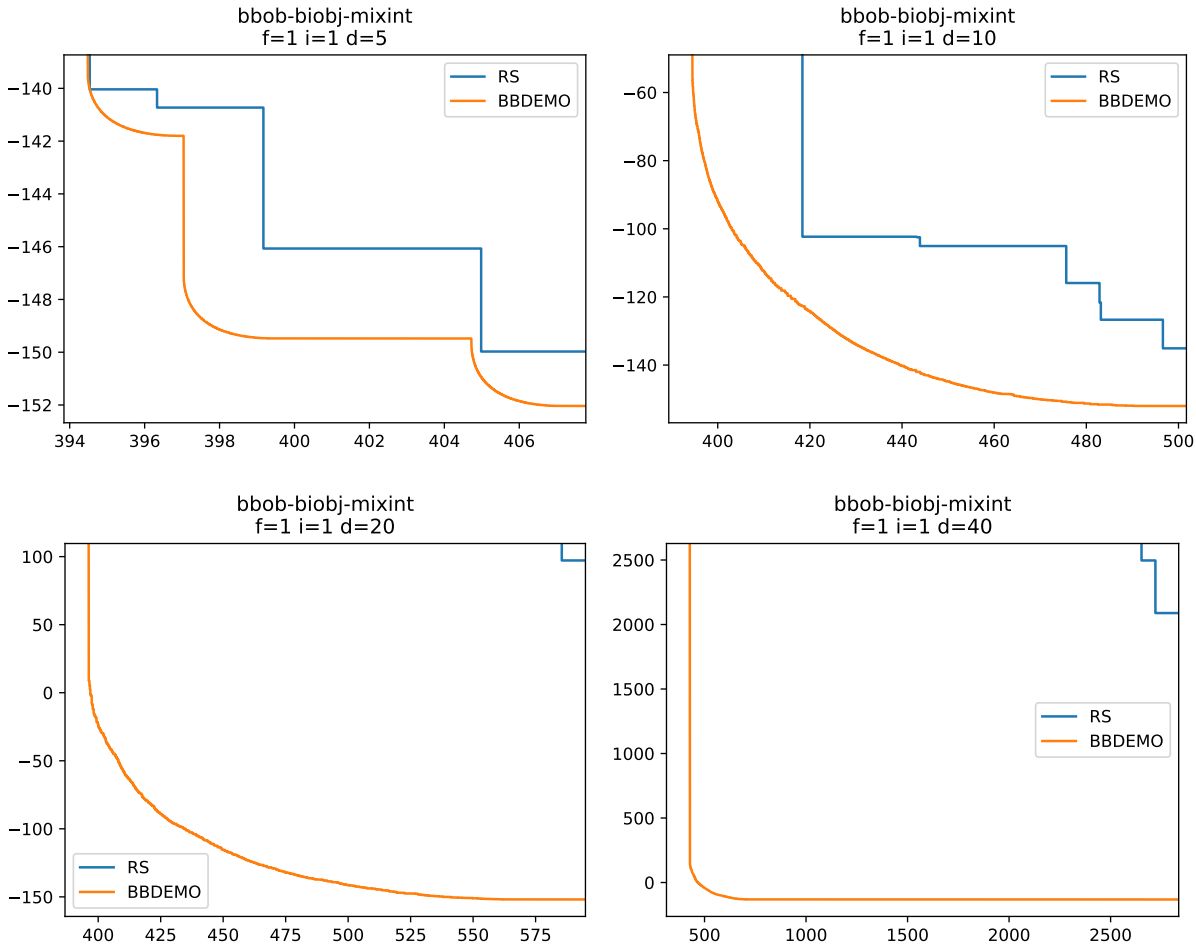

Figure 2: Pareto front approximations in objective space for the first instance of the Sphere/ Sphere function ($f_1 1$) in dimensions $n \in \{5, 10, 20, 40\}$.

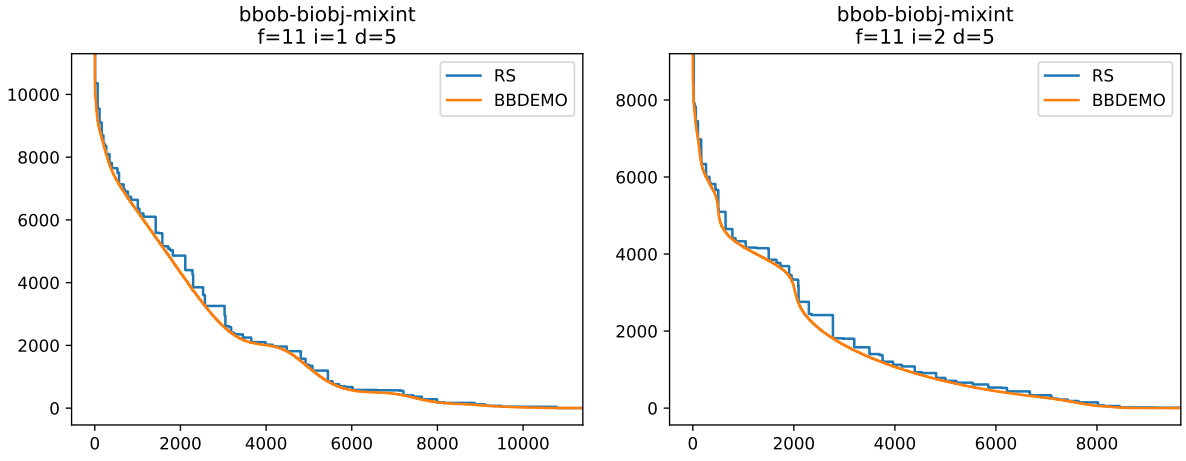

Figure 3: Pareto front approximations in objective space for the first two instances of the Sphere/ Sphere function (f_1) in 5-D.

and BBDEMO are non-dominated. This rarely happens in dimensions higher than five.

In dimensions larger than ten, the results achieved by the two algorithms differ so much that

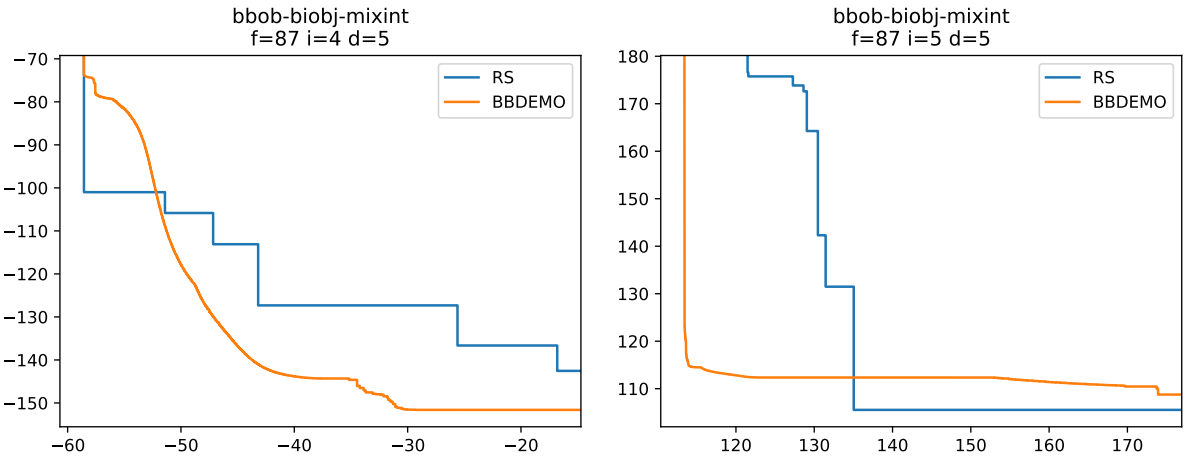

Figure 4: Pareto front approximations in objective space for instances four and five of the 101-peaks Gallagher/21-peaks Gallagher function (f_{87}) in 5-D.

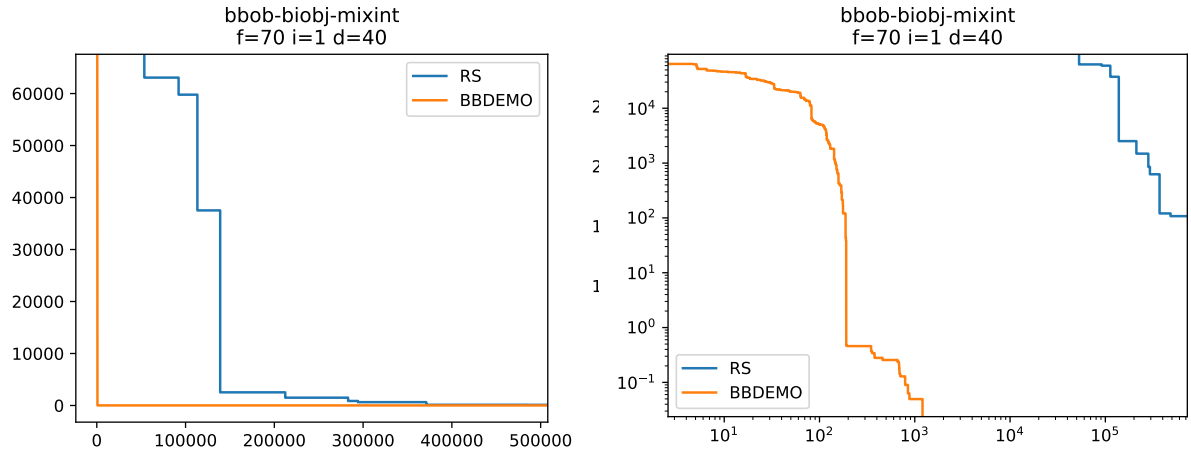

Figure 5: Pareto front approximations in objective space for the first instance of the Ellipsoid/Discus function (f_{70}) in 40-D. On the left hand side, the unscaled objective space and on the right hand side, differences to the best points achieved in logarithmic scale.

we need to use a logarithmic scale in the objective space (showing the difference from the best point achieved) in order to be able to zoom in on BBDEMO's performance. See Figure 5 for the performance by the two algorithms on the first instance of f_{70} (Ellipsoid/Discus) in 40-D. Based on the front ruggedness we can speculate that BBDEMO has not yet converged to the Pareto front on this problem.

7 Conclusions

We have introduced the BBDEMO algorithm that generates solutions similarly to the single-objective BBDE algorithm and compares them using the uncrowded hypervolume improvement. We have shown its results on the recently proposed `bbob-biobj-mixint` suite of mixed-integer test functions using random search as a baseline. Since at present a good approximation of the Pareto sets and fronts for these problems is still unknown, we cannot know how well BBDEMO is doing. These first results can nevertheless be very useful for future benchmarking studies.

Acknowledgment

The author acknowledges the financial support from the Slovenian Research Agency (project No. Z2-8177).

References

[1] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multiobjective optimization," in *Evolutionary Multiobjective Optimization*, ser. Advanced Information and Knowledge Processing. Springer, 2005, pp. 105–145.

[2] S. Huband, P. Hingston, L. Barone, and R. L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 477–506, 2006.

[3] N. Hansen, A. Auger, O. Mersmann, T. Tusar, and D. Brockhoff, "COCO: A platform for comparing continuous optimizers in a black-box setting," *CoRR*, vol. abs/1603.08785, 2016.

[4] V. Volz, B. Naujoks, P. Kerschke, and T. Tušar, "Single- and multi-objective game-benchmark for evolutionary algorithms," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. ACM, 2019, pp. 647–655.

[5] T. Tušar, D. Brockhoff, and N. Hansen, "Mixed-integer benchmark problems for single- and bi-objective optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. ACM, 2019, pp. 718–726.

[6] E. Zitzler and L. Thiele, "Multiobjective optimization using evolutionary algorithms - A comparative case study," in *Parallel Problem Solving from Nature - PPSN V, 5th International Conference, Amsterdam, The Netherlands, September 27-30, 1998, Proceedings*, ser. Lecture Notes in Computer Science, vol. 1498. Springer, 1998, pp. 292–304.

[7] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, and V. G. da Fonseca, "Performance assessment of multiobjective optimizers: an analysis and review," *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 117–132, 2003.

[8] R. Storn and K. V. Price, "Differential evolution - A simple and efficient heuristic for global optimization over continuous spaces," *Journal of Global Optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[9] T. Robič and B. Filipič, "DEMO: differential evolution for multiobjective optimization," in *Evolutionary Multi-Criterion Optimization, Third International Conference, EMO 2005, Guanajuato, Mexico, March 9-11, 2005, Proceedings*, ser. Lecture Notes in Computer Science, vol. 3410. Springer, 2005, pp. 520–533.

[10] K. Deb, S. Agrawal, A. Pratap, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 2, pp. 182–197, 2002.

[11] K. V. Price, "How symmetry constrains evolutionary optimizers," in *2017 IEEE Congress on Evolutionary Computation, CEC 2017, Donostia, San Sebastián, Spain, June 5-8, 2017*. IEEE, 2017, pp. 1712–1719.

[12] C. Touré, N. Hansen, A. Auger, and D. Brockhoff, "Uncrowded hypervolume improvement: COMO-CMA-ES and the sofomore framework," in *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2019, Prague, Czech Republic, July 13-17, 2019*. ACM, 2019, pp. 638–646.

[13] A. Vodopija, T. Tušar, and B. Filipič, "Comparing black-box differential evolution and classic differential evolution," in *Proceedings of the Genetic and Evolutionary Computation Conference Companion, GECCO 2018, Kyoto, Japan, July 15-19, 2018*. ACM, 2018, pp. 1537–1544.

[14] D. Brockhoff, T. Tušar, N. Hansen, and A. Auger, "Using well-understood single-objective functions in multiobjective black-box optimization test suites," *CoRR*, vol. abs/1604.00359v3, 2019.