A Bug in the Multiobjective Optimizer IBEA: Salutary Lessons for Code Release and a Performance Re-Assessment

Dimo Brockhoff

A Bug in the Multiobjective Optimizer IBEA: Salutary Lessons for Code Release and a Performance Re-Assessment*

Dimo Brockhoff

Inria Lille - Nord Europe, DOLPHIN project-team, 59650 Villeneuve d'Ascq, France
dimo.brockhoff@inria.fr

Abstract. The Indicator-Based Evolutionary Algorithm (IBEA) is one of the first indicator-based multiobjective optimization algorithms and due to its wide availability in several algorithm packages is often used as a reference algorithm when benchmarking multiobjective optimizers. The original publication on IBEA proposes to use two specific variants: one based on the ε-indicator and one based on the hypervolume. Several experimental studies concluded that, surprisingly, the IBEA variant with the ε-indicator performs better than the one with the hypervolume—even if the (unary) hypervolume indicator itself is the quality measure used in the performance assessment. Recently, a small bug has been found in the hypervolume variant of IBEA with large implications on its performance. Here, we not only explain the bug in detail and correct it, but also present the (improved) results of the corrected version. Moreover, and probably even more important for the scientific community, we point out that this bug has been transferred to other than the original software package, discuss how this obscured the bug, and argue in favor of some simple, even obvious guidelines how the optimization community should deal with algorithm source codes, documentation, and the (natural) existence of bugs in the future.

1 Introduction

The Indicator-Based Evolutionary Algorithm (IBEA, [12]) is one of the first proposed indicator-based multiobjective optimization algorithms. Due to its simplicity, good performance, and wide availability in several algorithm packages such as PISA [3], Paradiseo [7], jMetal [5] or the MOEA Framework [6], IBEA is an often-used reference algorithm when benchmarking multiobjective optimizers.

The main idea behind IBEA is to employ in the calculation of a solution's fitness a binary quality indicator, which assigns two solution sets a scalar value indicating their relative quality. The original publication proposes to use two specific IBEA variants: one based on the additive ε-indicator, denoted $\text{IBEA}_{\varepsilon+}$ in the following, and one based on the hypervolume (denoted IBEA_{HD}; more details about the algorithm are provided in the following section). Several experimental studies concluded that, surprisingly, the IBEA variant with the ε-indicator performs better than the one with the hypervolume [2, 10, 1]—even if

* This is an author version of the EMO 2015 paper published by Springer Verlag. The final publication is available at www.springerlink.com.

the (unary) hypervolume indicator itself is the quality measure used in the performance assessment [12]. This led to the fact that most studies using IBEA use the version employing the ε-indicator.

Recently, a small bug has been reported in the hypervolume variant of IBEA in the Paradiseo [7] implementation which turned out to stem from its original PISA implementation [3] and which has some large implications on its performance. In the following, we not only explain the bug in detail and correct it, but also present the (improved) results of the corrected version on the same test problems as in the original publication [12]. As expected, the corrected version outperforms the buggy one with the exceptions of the discrete knapsack and network processor design problems and for a low number of objective functions where the two versions do not differ statistically significantly. On the ZDT6 problem, we furthermore show that the former version was not invariant under permutations of the objective functions while the corrected one is.

Moreover, we have seen that the same bug has been also present in other algorithm packages such as jMetal [5] and the MOEA framework [6]. Hence, we argue during the final part of the paper in favor of independent implementations, thorough testing, and a precise and honest documentation of algorithm packages within our community.

2 IBEA

The general Indicator-Based Evolutionary Algorithm (IBEA) as proposed by Zitzler and Künzli [12] is one of the very first multiobjective optimizers to integrate user preferences in a clear and mathematically sound way. The main contribution of IBEA was to open up a new research area on the design of multiobjective optimization algorithms which employ a so-called *quality indicator* in their (environmental) selection procedure.

Before we describe the original IBEA algorithm in more detail, let us mention that we consider, w.l.o.g., minimization problems here where the Pareto dominance relation $\prec$ is defined between solutions x^1 and x^2 as $x^1 \prec x^2$ if and only if $f_i(x^1) \leq f_i(x^2)$ for all objective functions $f_i : X \to Z$ $(1 \leq i \leq k)$ and $f_i(x^1) < f_i(x^2)$ for at least one objective function. In this case, we also say x^1 dominates x^2. An m-ary quality indicator is furthermore a function $I : \Omega^m \to \mathbb{R}$ that maps m solution sets $X_1, \ldots, X_m$ from the set of all possible solutions $(X_1, \ldots, X_m \in \Omega = 2^X)$ to a real number. Nowadays, mostly *unary quality indicators* such as the standard hypervolume indicator are used in both performance assessment and the definition of solution quality within the environmental selection. Instead, IBEA itself is based on *binary quality indicators* that map *two* solution sets to a real number.

To be more precise, the fitness of a solution x^1 in IBEA's population P is assigned by

$$F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-I(\{x^2\},\{x^1\})/(c \cdot \kappa)}$$

where $\kappa > 0$ is a parameter of the algorithm and $c = \max_{x^1,x^2 \in P} |I(x^1, x^2)|$ is the maximum indicator value between any two population members. This fitness assignment scheme of IBEA has the theoretical property that if a solution x^1 dominates solution x^2 then also $F(x^1) > F(x^2)$ as long as the chosen binary indicator I itself is "dominance preserving"[1] [12]. Note that in the following, we abuse the mathematical notation and write $I(x, y)$ instead of $I(\{x\}, \{y\})$ if x and y are single solutions. Examples of dominance preserving binary indicators are the binary hypervolume and the binary ε-indicator which, for that reason, have been proposed to be used in the original IBEA publication.

The binary (additive) ε-indicator assigns to two solution sets A and B the minimal objective value ε by which all solutions in A have to be improved (along each objective) in order to (weakly) dominate all solutions in B:

$$I_{\varepsilon+}(A, B) = \min_{\varepsilon} \left\{ \forall x^2 \in B \exists x^1 \in A : f_i(x^1) - \varepsilon \leq f_i(x^2) \text{ for } i \in \{1, \ldots, k\} \right\} .$$

The binary ε-indicator is negative if all solutions in B are dominated by at least one solution in A.

The binary hypervolume indicator used in [12], assigns to two solution sets A and B the "volume of the space that is dominated by B but not by A with respect to a predefined reference point" in objective space [12]:

$$I_{HD}(A, B) = \begin{cases} I_H(B) - I_H(A) & \text{if } \forall x^2 \in B \exists x^1 \in A : x^1 \prec x^2 \\ I_H(A + B) - I_H(A) & \text{else} \end{cases} \quad (1)$$

where $I_H(.)$ denotes the standard (unary) hypervolume proposed in [13] and the index "HD" stands for "hypervolume difference". Also $I_{HD}(A, B)$ is negative if all solutions in B are dominated by at least one solution in A. Note also that neither of the two binary indicators is symmetric, i.e., typically $I(A, B) \neq I(B, A)$ holds. The corresponding IBEA variants using the above defined hypervolume and ε-indicator are denoted IBEA_{HD} and $\text{IBEA}_{\varepsilon+}$ respectively here.

Algorithm 1 shows the pseudo code of the entire IBEA procedure (copied and adapted from [12]). It starts with generating α solutions uniformly at random from the search space (Step 1). Then, IBEA follows the standard way of selecting solutions for mating (via a binary tournament with replacement, Step 5), generating new solutions from those selected solutions (via problem dependent crossover and mutation operators, Step 6), and environmental selection where the above described fitness assignment scheme is used (Step 2) to iteratively reduce the population back to the population size α by deleting the solutions with worst fitness successively (Step 3). Important to note is that the described adaptive version of IBEA scales both the objective values before computing the indicator values (Steps 2.1 and 2.2) and the indicator values themselves before to apply the above fitness assignment scheme (Steps 2.3, 2.4, and 3.3). Moreover, the calculation of the fitness is partially updated as soon as one solution

[1] A binary quality indicator I is called *dominance preserving* if for all solutions $x^1, x^2, x^3 \in X$ both $x^1 \prec x^2 \implies I(\{x^1\}, \{x^2\}) < I(\{x^2\}, \{x^1\})$ and $x^1 \prec x^2 \implies I(\{x^3\}, \{x^1\}) \geq I(\{x^3\}, \{x^2\})$ hold.

Algorithm 1 (Adaptive) IBEA as proposed in [12]

Input: α (population size)
N (maximum number of iterations)
κ (fitness scaling factor)

Output: A (Pareto set approximation)

Step 1: Initialization: Generate initial population P of size α at random; set iteration counter $m = 0$

Step 2: Fitness Assignment: First scale objective and indicator values; then use scaled values to assign fitness for each population member $x^1 \in P$:

1. Determine lower ($\underline{b_i} = \min_{x \in P} f_i(x)$) and upper bound ($\overline{b_i} = \max_{x \in P} f_i(x)$) of each objective function
2. Scale each objective to interval $[0, 1]$: $f_i'(x) = (f_i(x) - \underline{b_i})/(\overline{b_i} - \underline{b_i})$
3. Calculate indicator values $I(x^1, x^2)$ using the scaled objective values f_i' and determine the maximum absolute indicator value $c = \max_{x^1, x^2 \in P} |I(x^1, x^2)|$
4. For all $x^1 \in P$ set $F(x^1) = \sum_{x^2 \in P \setminus \{x^1\}} -e^{-I(x^2, x^1)/(c \cdot \kappa)}$

Step 3: Environmental Selection: Iterate the following three steps until the size of population P does not exceed α :

1. Choose an individual $x^* \in P$ with the smallest fitness value, i.e., $F(x^*) \leq F(x)$ for all $x \in P$
2. Remove x^* from the population
3. Update fitness values of all remaining individuals $x \in P$ as $F(x) = F(x) + e^{-I(x^*, x)/(c \cdot \kappa)}$

Step 4: Termination: If $m \geq N$ or another stopping criterion is fulfilled, stop and return the non-dominated solutions in P as A

Step 5: Mating Selection: Perform binary tournament selection with replacement on P in order to fill the temporary mating pool P'

Step 6: Variation: Apply recombination and mutation operators to the mating pool P' and add the resulting offspring to P. Increment iteration counter ($m = m + 1$) and go to Step 2

is deleted during the environmental selection (Step 3.3). Finally, the algorithm terminates when the total number of iterations N are reached or another user-defined stopping criterion is reached (not implemented in the PISA version).

3 The Bug

The reported bug appeared in the hypervolume calculation of IBEA$_{HD}$, more precisely in the recursive "hypervolume by slicing objectives" technique used in the original PISA implementation, see line 33 of the original C code in Fig. 2. It is caused by a typo which misplaces the correct variable "a" by "b"—resulting in wrongly adding the volume of an objective space part to the indicator value $I(a, b)$ that is not dominated by either solution. Figure 1 shows an example where the bug not only miscalculates the binary hypervolume indicator values but also results in a different order of the two solutions with respect to their fitness. Note that the bug results only in erroneous decisions where the point on the left is wrongly preferred while the opposite never happens. Hence, it can be expected

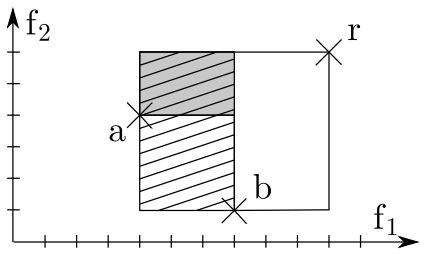

Fig. 1. Illustration of the impact of the bug on the comparison of two solutions with objective vectors a and b. The gray box corresponds to the true hypervolume dominated solely by objective vector a while the striped box shows the actual contribution computed by the original, buggy code. The buggy code considers a better than b while the corrected code considers b better. In both cases, the hypervolume solely dominated by b is computed correctly.

that the correction of this bug has an impact on the search performance of IBEA. It might even explain and counterbalance the impression of previous benchmarking studies that the hypervolume-based IBEA does surprisingly not perform as well as the ε-indicator-based version when the (unary) hypervolume indicator is used as performance measure. In the following, we will thus investigate the effect of the bug fix on the performance of IBEA$_{HD}$ extensively.

4 Concrete Implications for the Performance of IBEA

To investigate the concrete implications of the bug (and its correction) on the performance of IBEA, we rerun the experiments of the original IBEA paper by Zitzler and Künzli [12]. Before we have a closer look on the results, let us note that here, we can show the results for the 2-objective ZDT6 and EXPO2 problems, the 3-objective DTLZ2, DTLZ6, and EXPO3 problems, and on the 4-objective EXPO4 problem mentioned in the original publication—many of which had to be omitted in the original IBEA paper. As much as possible, we tried to use the same problem and algorithm parameters as in the original study.

4.1 Experimental Setup

All experiments were performed in PISA [3] with all modules downloaded from `http://www.tik.ee.ethz.ch/sop/pisa` in their August 2014 version. As problems, we chose the continuous ZDT6, DTLZ2, and DTLZ6 problems with 10, 12, and 22 variables and 2, 3, and 3 objective functions respectively as suggested in the original publications [4, 11]. In addition, we chose the discrete 2-objective 0-1-knapsack problem with 100 items [13] as well as the 2-, 3-, and 4-objective EXPO problem (network processor design) with standard PISA settings [9].

Together with IBEA$_{\varepsilon+}$ and the buggy and corrected IBEA$_{HD}$, we ran the PISA implementations of NSGA-II and SPEA2 as in [12]. All algorithms used a

```
 1 double calcHypInd(ind *p_ind_a, ind *p_ind_b, int d)
 2 /* calculates the hypervolume of that portion of the objective space that
 3    is dominated by individual a but not by individual b */
 4 {
 5    double a, b, r, max;
 6    double volume = 0;
 7
 8    r = rho * (bounds[d - 1].max - bounds[d - 1].min);
 9    max = bounds[d - 1].min + r;
10
11    a = p_ind_a->f[d - 1];
12    if (p_ind_b == NULL)
13       b = max;
14    else
15       b = p_ind_b->f[d - 1];
16
17    if (d == 1)
18    {
19       if (a < b)
20          volume = (b - a) / r;
21       else
22          volume = 0;
23    }
24    else
25    {
26       if (a < b)
27       {
28          volume = calcHypInd(p_ind_a, NULL, d - 1) * (b - a) / r;
29          volume += calcHypInd(p_ind_a, p_ind_b, d - 1) * (max - b) / r;
30       }
31       else
32       {
33          volume = calcHypInd(p_ind_a, p_ind_b, d - 1)
                * (max - a) / r; \\ corrected version
                                 \\ original version: "* (max - b) / r;"
24       }
25    }
26
27    return (volume);
28 }
```

Fig. 2. The source code snippet of the PISA implementation of IBEA and the bug in line 33. For readability, the function name is shortened here.

population size of $\alpha = 100$, a binary tournament mating selection, and were run independently 30 times for 200 iterations each.

Regarding the variation operators, the continuous problems used SBX crossover and polynomial mutation with the PISA parameters set as individual_mutation_probability=1; individual_recombination_probability=1; variable_mutation_probability=0.01; variable_recombination_probability=1; variable_swap_probability=0; eta_mutation=20; eta_recombination=20 and use_symmetric_recombination=1. For the discrete knapsack problem, we used one-bit mutation and one-point-crossover with the PISA parameters mutation_probability=1 and recombination_probability=0.8. For EXPO, we used the standard PISA settings. For IBEA, we furthermore used $\kappa = 0.05$ as suggested in [12] and a reference point of $(2, \ldots, 2)$ for the internal normalized calculations of I_{HD}.

To compare the algorithms, the hypervolume and the additive ε-indicator were recorded every 50 iterations and computed relative to a reference set, obtained by joining all non-dominated solutions at this specific iteration over all algorithm runs. Before computing the indicators, the objective vectors had been normalized such that all non-dominated points at the investigated iteration over all algorithms defined the box $[1, 2]^k$. The reference point for the hypervolume indicator was chosen as $(2.1, \ldots, 2.1)$ as in [12].

4.2 Comparison Between Buggy and Corrected IBEA$_{HD}$

Figures 3 and 4 show the box plots of both the hypervolume and ε-indicator for the four algorithms NSGA-II, SPEA2, IBEA$_{\varepsilon+}$, and IBEA$_{HD}$ after 200 iterations—on the left for the buggy version of IBEA$_{HD}$ and on the right for the corrected version of IBEA$_{HD}$[2]. Figure 5 shows the direct comparison between the buggy IBEA$_{HD}$ and the corrected version on each problem after 50, 100, 150, and 200 iterations. All boxplots are drawn with Matlab and the ends of the notches correspond to "$q_2 - 1.57(q_3 - q_1)/\sqrt{n}$ and $q_2 + 1.57(q_3 - q_1)/\sqrt{n}$, where q_2 is the median, q_1 and q_3 the 25th and 75th percentiles [...], and n is the number of runs", thus indicating statistically significant medians "at the 5% significance level if their intervals do not overlap" (compare the Matlab documentation for further details).

Overall, three main observations can be made: As to the continuous problems, the bug fix has a positive effect on IBEA$_{HD}$ on the 3-objective problems DTLZ2 and DTLZ6 with respect to both indicators. The corrected IBEA$_{HD}$ now results in similar or better indicator values than IBEA$_{\varepsilon+}$. For the 2-objective ZDT6 problem, however, the effect is small and sometimes slightly detrimental (though not statistically significant as the boxplots' notches do overlap). We give a possible explanation for this behavior on ZDT6 in the following section.

[2] The reason for two sets of figures with four algorithms each instead of showing all five algorithms in a single plot is to see the effect of the bug fix directly. Because all results are relative to other algorithms, joining all algorithms alters the box plots (slightly) and thus makes comparisons with the original paper [12] more difficult. For a direct comparison of the buggy and corrected IBEA$_{HD}$, we provide Fig. 5.

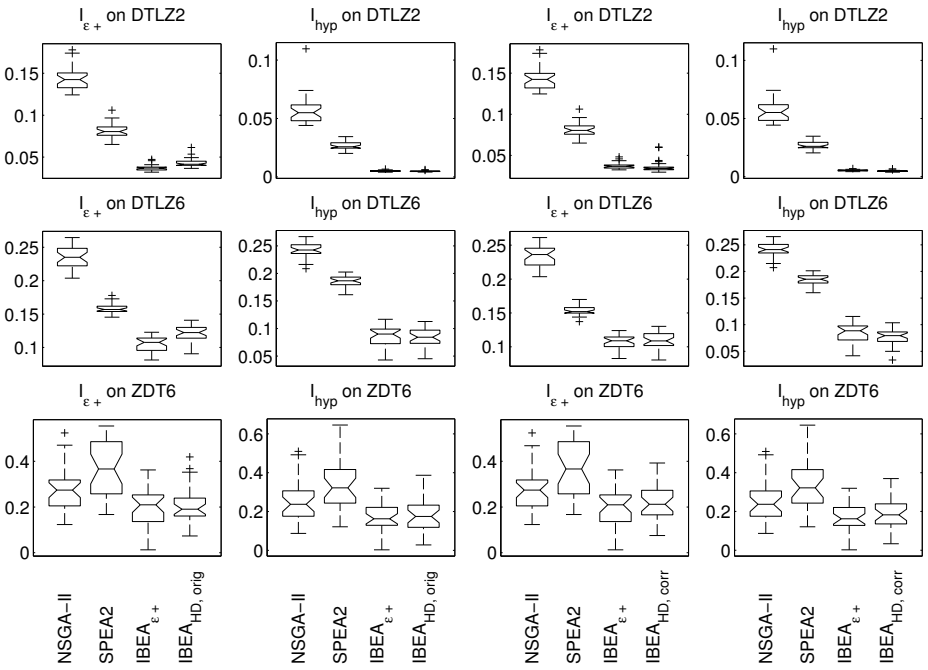

Fig. 3. Boxplots comparing the performance of NSGA-II, SPEA2, IBEA$_{\varepsilon+}$, and IBEA$_{HD}$ for different continuous problems (rows). The left two columns show the results for the original IBEA$_{HD}$ implementation, the two right columns the same results for the corrected IBEA$_{HD}$ version. Columns 1 and 3 show results for the additive ε-indicator; columns 2 and 4 results for the hypervolume indicator. All indicators are computed after 200 generations with respect to the reference set stemming from all algorithms of that plot.

As to the discrete problems, no positive effect of the bug fix can be reported. The results before and after the bug fix are similar and only very few statistically significant differences can be observed when looking at the notches in Fig. 5. The corrected IBEA$_{HD}$ version is not better than IBEA$_{\varepsilon+}$ with respect to the hypervolume indicator except for the 4-objective EXPO problem and a larger number of iterations. Exactly in these cases of the 4-objective EXPO problem, on the other hand, the additive ε-indicator is significantly worse for the corrected IBEA$_{HD}$ version. It seems as if the discreteness of the problems and the comparatively small number of non-dominated solutions in the resulting populations do not allow the (correct) hypervolume fitness to be effective. It can be also noted that especially for the knapsack problem, the variance between runs is quite large, which means that the impact of the bug fix can only be rather small anyway as the results differ much more among the runs than among the algorithms.

Last, we see that the positive effect of the bug fix, at least for the selected problems, becomes larger with an increasing number of objective functions and more pronounced on the continuous problems with more function evaluations. Another fundamental improvement caused by the bug fix will be discussed in the next subsection.

4.3 Invariance With Respect to Objective Permutations

One additional observation, we can make when comparing the buggy and the corrected version of IBEA$_{HD}$, is that the corrected version is, as expected, invariant over a permutation of the objective functions, i.e., the performance is the same when we for example exchange the first and the second objective function. This invariance is a desired property of an optimization algorithm as it generalizes the statements we can make about the performance of the algorithm without actually testing it. The invariance properties of the hypervolume indicator give us theoretically this invariance of IBEA$_{HD}$, but it turns out that the bug in the original PISA implementation resulted in an algorithm that is not invariant. To investigate this, we ran both the buggy and the corrected version of IBEA$_{HD}$ with a population size of 100 for 200 iterations on the ZDT6 problem—this time with an increased number of 100 variables to better see the effect. To be precise, we ran each algorithm independently 30 times on the ZDT6 problem and again, with the same initial random seeds, on the ZDT6 problem where the two objective functions are exchanged (we denote this problem as the "inverted" ZDT6 problem in the figures).

To compare the performance on both problems, we plot the empirical attainment functions [8] where for the "inverted" ZDT6 problem, both objectives are again swapped for comparability. For the corrected IBEA$_{HD}$, due to the same initial random seeds, the 30 runs are exactly the same, while for the buggy IBEA$_{HD}$, we see some differences. Figure 6 shows the comparison of the buggy and the corrected IBEA$_{HD}$ on the original ZDT6 problem (top) and on the "inverted" ZDT6 problem (middle), as well as the results of the buggy IBEA$_{HD}$ on ZDT6 against the buggy IBEA$_{HD}$ results on the "inverted" ZDT6 (bottom)—once again, the results for the corrected version are identical. Not only do the empirical attainment functions differ on the two functions for the buggy IBEA$_{HD}$ (Fig. 6, bottom), especially when looking at the median, but the buggy IBEA$_{HD}$ also significantly outperforms the corrected version on the original ZDT6 problem (gray areas in Fig. 6, top). On the other hand, the results are comparable or even in favor of the corrected version to the left of the Pareto front for the "inverted" ZDT6 (Fig. 6, middle). This let us come to the conclusion that the buggy version of IBEA$_{HD}$ exploits the fact that the original ZDT6 problem can be solved by finding solutions with good first objective and then moving along the axis, which is supported by the fact that the bug favors solutions on the left as mentioned in Sec. 3 (compare Fig. 6, bottom).

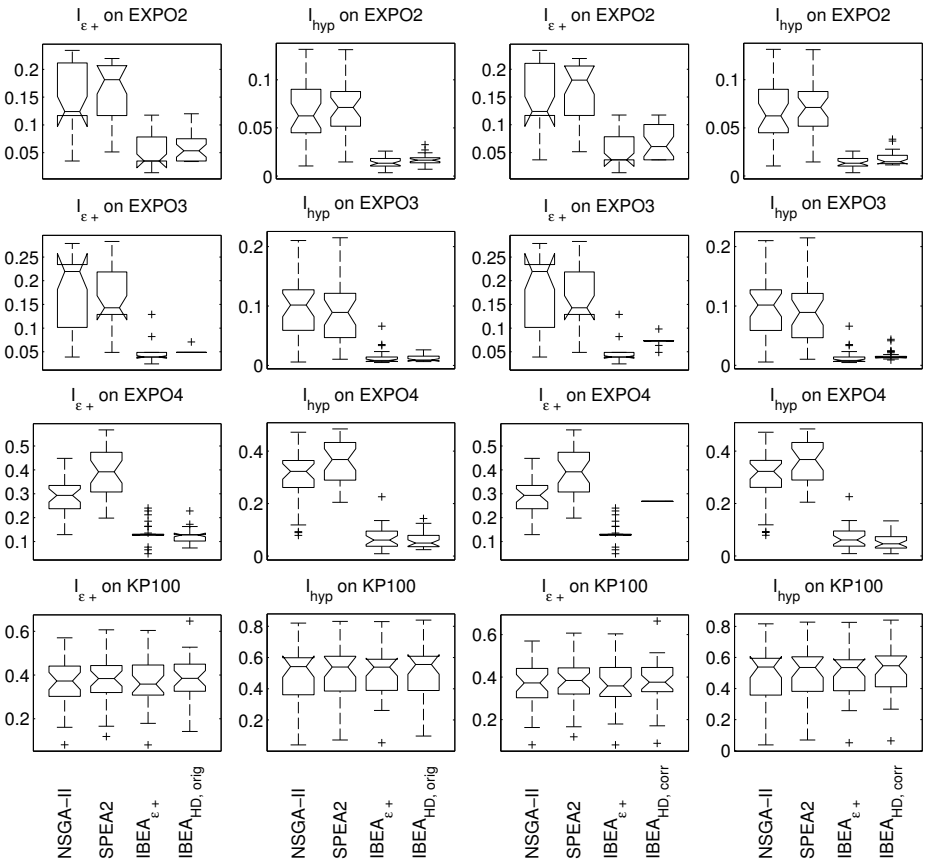

Fig. 4. Boxplots comparing the performance of NSGA-II, SPEA2, IBEA$_{\varepsilon+}$, and IBEA$_{HD}$ for some discrete problems (rows). The left two columns show the results for the original IBEA$_{HD}$ implementation, the two right columns the same results for the corrected IBEA$_{HD}$ version. Columns 1 and 3 show results for the additive ε-indicator; columns 2 and 4 results for the hypervolume indicator. All indicators are computed after 200 generations with respect to the reference set stemming from all algorithms of that plot.

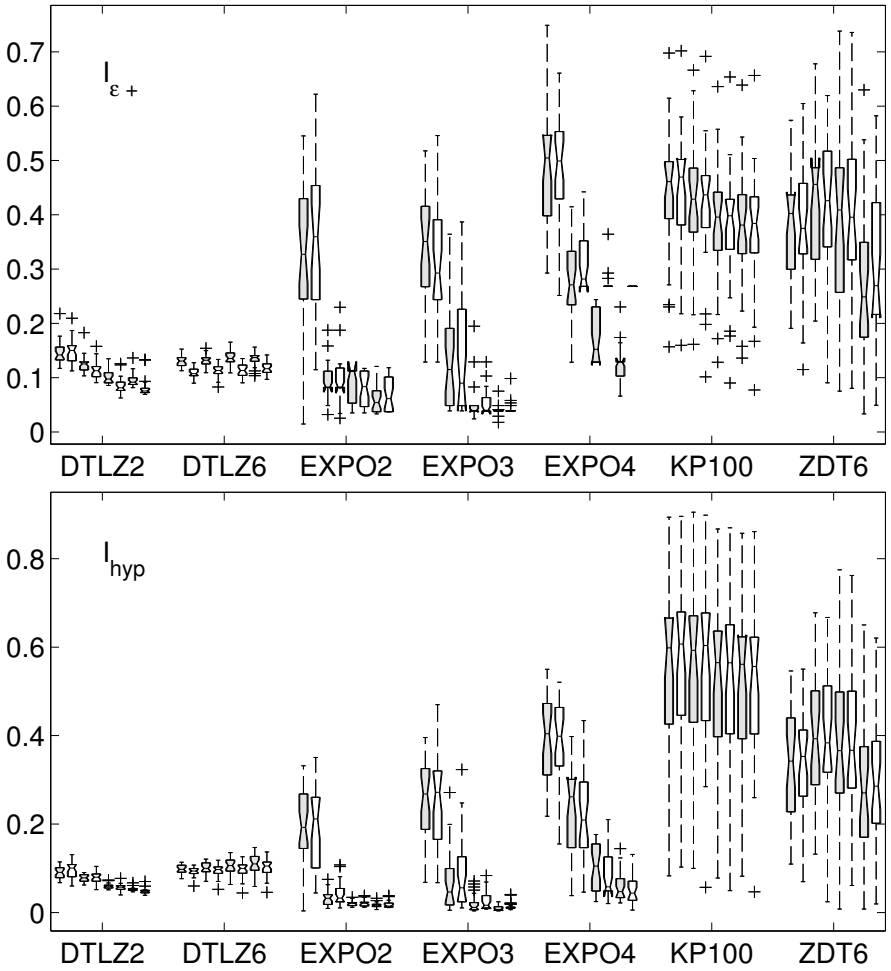

Fig. 5. Comparison via boxplots between the buggy and the corrected IBEA$_{HD}$ version for various problems and number of iterations. For each problem and from left to right, iterations 50, 50, 100, 100, 150, 150, 200, and 200; the left gray box corresponds to the buggy IBEA$_{HD}$ while the right white box corresponds to the corrected version. The top plot shows the additive ε-indicator values over 30 runs, the bottom plot the results for the hypervolume indicator.

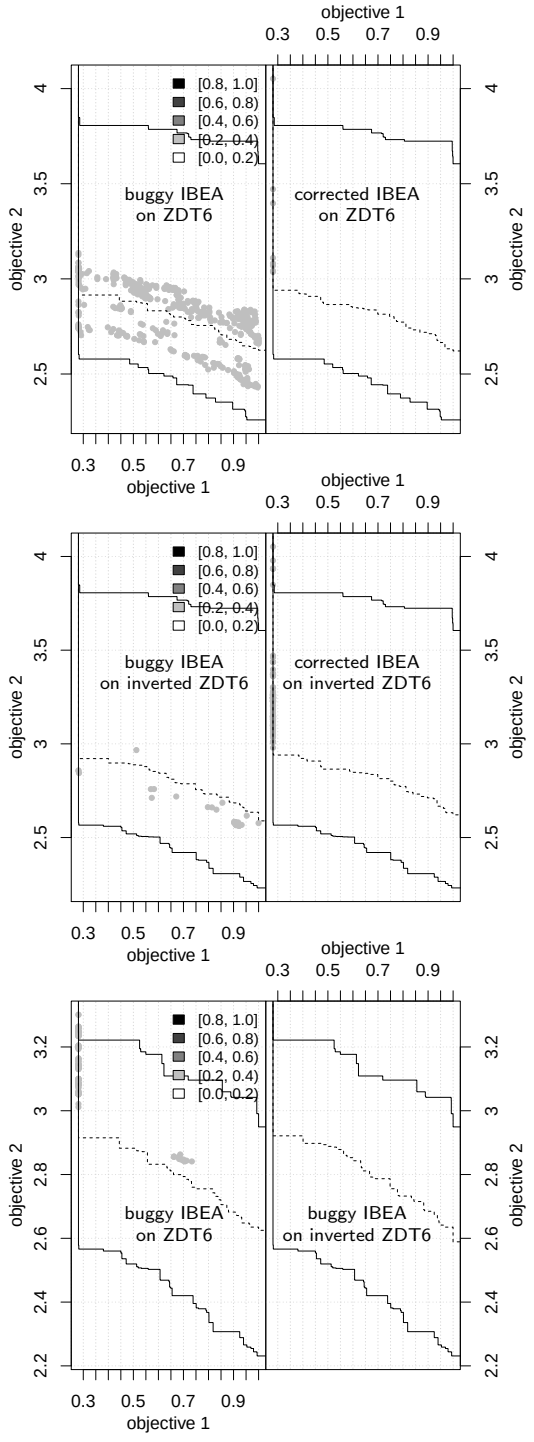

Fig. 6. Empirical attainment function plots after $200{\cdot}D$ function evaluations comparing the buggy and corrected IBEA$_{HD}$ on the ZDT6 problem with 100 variables (top) and on its inverted version (middle, after swapping the objectives again for comparison reasons). On the bottom plot, the results of the buggy IBEA$_{HD}$ on both problems.

5 General Implications on How to Write, Document, and Distribute Algorithm Implementations

Let us end the paper with a look at the broader picture and the way we should deal with algorithm implementations in general. The process of finding, solving, and tracking the bug was actually not optimal from the author's personal point-of-view and occurred as follows. We were made aware about this bug by receiving an e-mail from Yann Semet and his colleagues at Thales who had problems debugging their algorithm which was based on the Paradiseo [7] implementation of IBEA. It quickly turned out that the reason for their suspicious results was the bug reported above. Moreover, it turned out that the bug was also present in the original PISA [3] implementation and that both implementations were actually the same (except for some renaming of functions and variables in Paradiseo). Unfortunately, the Paradiseo code did not mention the original implementation such that it was not directly possible to track the observed bug to its origin.

In the wake of this observation, we checked more carefully other software packages that provide the IBEA algorithm. Though implemented in Java while the PISA implementation of IBEA is in C, also for jMetal [5] the same code snippet of the PISA implementation was used without any reference to the original PISA code. Solely the implementation of IBEA in the MOEA framework [6] mentioned clearly where the code was coming from. It was furthermore easy to report the bug via the corresponding online bug-tracking system—a functionality that the other three frameworks (including PISA itself) either do not offer at the moment or that, in the case of Paradiseo, are not linked from the webpage, such that the bug had to be therefore reported by plain e-mail. Let us mention that, for all above software packages, the developers quickly replied to our bug report and the latest versions of the MOEA Framework (v2.3), jMetal (4.5.1), PISA (from October 13, 2014 on), and the Github repository of Paradiseo already contain the bug fix.

The discovery of the bug within the IBEA implementation in the software packages mentioned and the discovery that several implementations are just copies of the original code without references to it will hopefully have a lasting impact on how source code of optimization algorithms is written and distributed in our community. At least, we should always try to remind ourselves on the following two main aspects:

- Algorithm implementations, even if they are provided in big and well-known algorithm packages, should be always questioned and tested thoroughly. Simple unit tests and even more important independent implementations would have exposed the bug.
- Re-using code can also be beneficial in terms of a broader distribution of an algorithm due to reduced implementation times. But if code is copied, the original code basis should always be mentioned in the code for easier tracking of bugs over different software packages—independent of any (obvious) copyright issues one needs to adhere. To allow for easier tracking in the opposite direction and to distribute bug fixes to other packages, it is fur-

thermore recommended that the original code is regularly checked for bug fixes and the secondary code updated accordingly.

Last, we would like to suggest reporting version numbers of the algorithms used in our papers, such as done frequently for example when the single-objective CMA-ES is used. This will, in case of a bug, make it much simpler to find out whether the reported results are trustworthy or not.

6 Conclusions

The correction of a bug in the hypervolume calculation of the multiobjective optimizer IBEA, uncovered since its first implementation in 2004, showed an important impact in the algorithm's search performance. The buggy implementation is not invariant against permutations of the objective functions and shows worse results especially for continuous problems and when the number of objective functions is high. On the tested discrete problems, the buggy and the corrected IBEA behave similar with the only observed significant worsening for the network processor design problem EXPO when the ε-indicator is used as performance measure. The bug might, furthermore, explain why the IBEA variant employing the ε-indicator was so far more often used in empirical studies than the one using the hypervolume indicator—and thus resulting in the wrong perception of algorithm performances.

Probably even more important than the correction of the bug was the observation that several algorithm frameworks such as Paradiseo and jMetal copied the original PISA code of the (buggy) IBEA without mentioning where the code was coming from. Let us be clear that—as long as no copyright is violated of course—having comparable algorithm implementations with the same performance, in general, is a big plus for comparing and applying algorithms in practice (for example by having platform independent implementations). But without a truly independent implementation of IBEA, it took almost 10 years to find such an important bug as the one discussed. Our community should therefore try to have (at least) *two independent implementations of the main algorithms available* (and *a thorough check that they do the same*). Copying code without referencing to the original source is furthermore detrimental as it is almost impossible to track bug fixes over different software packages. We should therefore also aim at *more visible links between our software packages* and more scientific and technical exchanges among their developers. Furthermore, we should aim at more (and the right) *unit tests* as the simple test described in Sec. 4.3 could have detected the bug earlier. However, also testing cannot detect all bugs: the unit tests in the MOEA Framework for example fully cover the package containing the bug. Since we have to live with the fact that our software will naturally contain bugs, it is therefore even more important that we provide easy ways to report them via *bugtrackers* and to always *mention version numbers* of the algorithms we use in our papers. Addressing the mentioned challenges in the future when it comes to algorithm implementations and distributions will hopefully allow the (multiob-

jective) optimization community to appear even stronger and more trustworthy to the outside.

Acknowledgments. This work was supported by the grant ANR-12-MONU-0009 (NumBBO) of the French National Research Agency. The author would also like to acknowledge the JSPS funded project "Global Research on the Framework of Evolutionary Solution Search to Accelerate Innovation". Special thanks go to Yann Semet and his colleagues from Thales who originally pointed out the bug in the Paradiseo implementation of IBEA. Many thanks also go to Beat Futterknecht, Benny Gächter, David Hadka, Arnaud Liefooghe, Antonio Nebro, and Lothar Thiele for their help with the software packages mentioned in the paper and to Anne Auger, Nikolaus Hansen, Arnaud Liefooghe and the anonymous reviewers for their fruitful comments on the paper.

References

1. Bader, J.: Hypervolume-Based Search For Multiobjective Optimization: Theory and Methods. Ph.D. thesis, ETH Zurich (2010)
2. Basseur, M., Burke, E.K.: Indicator-Based Multi-Objective Local Search. In: Congress on Evolutionary Computation (CEC 2007). pp. 3100–3107 (2007)
3. Bleuler, S., Laumanns, M., Thiele, L., Zitzler, E.: PISA—A Platform and Programming Language Independent Interface for Search Algorithms. In: Evolutionary Multi-Criterion Optimization (EMO 2003). pp. 494–508 (2003)
4. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable Test Problems for Evolutionary Multi-Objective Optimization. In: Evolutionary Multiobjective Optimization: Theoretical Advances and Applications, pp. 105–145 (2005)
5. Durillo, J.J., Nebro, A.J.: jMetal: a Java Framework for Multi-Objective Optimization. Advances in Engineering Software 42, 760–771 (2011)
6. Hadka, D.: MOEA Framework (2014), http://www.moeaframework.org/
7. Liefooghe, A., Basseur, M., Jourdan, L., Talbi, E.G.: ParadisEO-MOEO: A Framework for Evolutionary Multi-objective Optimization. In: Conference on Evolutionary Multi-Criterion Optimization (EMO 2009). pp. 386–400 (2007)
8. López-Ibáñez, M., Paquete, L., Stützle, T.: Exploratory Analysis of Stochastic Local Search Algorithms in Biobjective Optimization. In: Experimental Methods for the Analysis of Optimization Algorithms. pp. 209–222 (2010)
9. Thiele, L., Chakraborty, S., Gries, M., Künzli, S.: Design Space Exploration of Network Processor Architectures. In: Network Processor Design 2002: Design Principles and Practices. Morgan Kaufmann (2002)
10. Wagner, T., Beume, N., Naujoks, B.: Pareto-, Aggregation-, and Indicator-based Methods in Many-objective Optimization. In: Evolutionary Multi-Criterion Optimization (EMO 2007). pp. 742–756 (2007)
11. Zitzler, E., Deb, K., Thiele, L.: Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. Evolutionary Computation 8(2), 173–195 (2000)
12. Zitzler, E., Künzli, S.: Indicator-Based Selection in Multiobjective Search. In: Parallel Problem Solving from Nature (PPSN VIII). pp. 832–842 (2004)
13. Zitzler, E., Thiele, L.: Multiobjective Evolutionary Algorithms: A Comparative Case Study and the Strength Pareto Approach. IEEE Transactions on Evolutionary Computation 3(4), 257–271 (1999)