Considerations in the Incremental Hypervolume Algorithm of the WFG

Raquel Hernández Gómez[1], Jesús Guillermo Falcón-Cardona[2][0000−0001−5991−6927], and Carlos A. Coello Coello[1][0000−0002−8435−680X]⋆

[1] CINVESTAV-IPN
Department of Computer Science
Evolutionary Computation Group
Av. IPN No. 2508
Col. San Pedro Zacatenco
Mexico City, Mexico 07360
ccoello@cs.cinvestav.mx
[2] Tecnológico de Monterrey
School of Engineering and Science
Computer Science Department
Av. Eugenio Garza Sada 2501
Col. Tecnológico
Monterrey, Nuevo León, Mexico 64700
jfalcon@tec.mx

Abstract. The hypervolume indicator (HV) has been subject of a lot of research in the last few years, mainly because its maximization yields near-optimal approximations of the Pareto optimal front of a multi-objective optimization problem. This feature has been exploited by several evolutionary optimizers, in spite of the considerable growth in computational cost that it is involved in the computation of HV as we increase the number of objectives. Some years ago, the Walking Fish Group (WFG) implemented a new version of the incremental hypervolume algorithm, named IWFG 1.01. This implementation is the fastest reported to date for determining the solution that contributes the least to the HV of a non-dominated set. Nevertheless, this new version has gone mostly unnoticed by the research community. We believe that this is due to an error in the source code provided by the authors of this algorithm, which appears when coupling it to a multi-objective evolutionary algorithm. In this paper, we describe this error, and we propose a solution to fix it. Moreover, we illustrate the significant gains in performance produced by IWFG 1.01 in many-objective optimization problems (i.e., problems having three or more objectives), when integrated into the S-Metric Selection Evolutionary Multi-Objective Algorithm (SMS-EMOA).

Keywords: Hypervolume indicator · multi-objective optimization · selection mechanism

⋆ The third author acknowledges support from CONACyT project no. 2016-01-1920 (*Investigación en Fronteras de la Ciencia 2016*)

1 Introduction

The hypervolume indicator (HV) [15], also known as the *Lebesgue measure* or S-metric, is one of the most preferred quality indicators (QIs) for comparing multi-objective optimizers. In a single value, HV captures convergence to the Pareto optimal front as well as spread along the objective space. HV and its variants are the only unary QIs that are known to be Pareto compliant [16] and it has been proved that maximizing HV is equivalent to reaching the Pareto optimal set [10]. For these reasons, several multi-objective evolutionary algorithms (MOEAs) have incorporated HV in their survival selection mechanism [9].

Although the computational cost of calculating the exact HV is exponential with the scaling of the objectives [13], the Walking Fish Group (WFG) (`http://www.wfg.csse.uwa.edu.au/hypervolume`) has proposed clever implementations where, in practice, the real performance is unrelated to this worst case complexity [5, 14, 13]. Of our particular interest is the Incremental Hypervolume Algorithm (IWFG) [5, 14], designed for determining which point in a set contributes least to HV. This algorithm uses several ideas to provide a substantial speed up. The most recent implementation is the IWFG 1.01 [5], which was released in November 2015. This version reported outstanding performance for even large fronts, being significantly faster than previous approaches in many-objective optimization problems [3]. However, this important version has gone unnoticed by the research community. Popular frameworks of evolutionary multi-objective optimization, such as jMetal (`http://jmetal.github.io/jMetal`) and MOEA Framework (`http://moeaframework.org`) do not have this update. We believe that this omission is because of the occurrence of an error, which is triggered when integrating IWFG 1.01 into an MOEA.

The main contributions of this paper are the isolation, replication and description of this error, as well as an easy-to-implement solution to it. Furthermore, we show the potential gains in speed up of IWFG 1.01 when coupled to the S-Metric Selection Evolutionary Multi-Objective Algorithm (SMS-EMOA) [1] (a hypervolume-based algorithm) on some test problems of the Deb-Thiele-Laumanns-Zitzler (DTLZ) test suite [8]. The rest of this paper is organized as follows. Sect. 2 defines the concepts and notation used in multi-objective optimization. Sect. 3 outlines the IWFG 1.01 algorithm. Sect. 4 provides our contribution. Sect. 5 presents the validation of our proposed solution. Sect. 6 contains our conclusions.

2 Background

We are interested in solving Multi-Objective Optimization Problems (MOPs) of the form:

$$\text{Minimize} \quad \mathbf{f}(\mathbf{x}) := (f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_m(\mathbf{x})) \tag{1}$$

$$\text{subject to} \quad \mathbf{x} \in X, \tag{2}$$

where $X \subset \mathbb{R}^n$ is the feasible region in decision space, and $Z \subset \mathbb{R}^m$ is in the objective space. Each decision vector $\mathbf{x} \in X$ is related to an objective vector $\mathbf{f}(\mathbf{x}) \in Z$. Since objectives might be in conflict with one another, it is not possible to compare two solutions $\mathbf{x}, \mathbf{y} \in X$ in a straightforward manner as in single-objective optimization. As an alternative, the *Pareto dominance relation* must be applied. It is said that $\mathbf{x}$ "dominates" $\mathbf{y}$, if it stands:

$$\begin{aligned} \mathbf{x} \prec \mathbf{y} \Leftrightarrow & \left(\forall i \in \{1, \ldots, m\}, f_i(\mathbf{x}) \leq f_i(\mathbf{y})\right) \wedge \\ & \left(\exists j \in \{1, \ldots, m\}, f_j(\mathbf{x}) < f_j(\mathbf{y})\right). \end{aligned}$$

The non-dominated solutions of a set $A \subseteq X$ are defined as:

$$\mathrm{NDS}(A) := \{\mathbf{a} \in A \colon \nexists \mathbf{a}' \in A, \mathbf{a}' \prec \mathbf{a}\}. \tag{3}$$

The solution to an MOP consists of finding the optimal set of non-dominated decision vectors in all X, which cannot be improved in any objective without worsening at least another objective. This set is known as the *Pareto Optimal Set*, and its image is named the *Pareto Optimal Front*.

The hypervolume indicator [15] determines the size of the portion of the objective space that is dominated by a set A of non-dominated solutions, collectively and bounded by a reference point $\mathbf{z} \in \mathbb{R}^m$, defined as:

$$HV(A; \mathbf{z}) = \Lambda\left(\bigcup_{\mathbf{a} \in A} \{\mathbf{x} \mid \mathbf{a} \prec \mathbf{x} \prec \mathbf{z}\}\right), \tag{4}$$

where Λ denotes the Lebesgue measure in $\mathbb{R}^m$, and $\mathbf{z}$ should be dominated by all members of A.

Another concept is the *inclusive hypervolume* of a solution $\mathbf{p}$ which is used to denote the size of the part of the objective space dominated by $\mathbf{p}$ alone, that is:

$$IncHV(\mathbf{p}; \mathbf{z}) := HV(\{\mathbf{p}\}; \mathbf{z}). \tag{5}$$

The *hypervolume contribution* or *exclusive hypervolume* of a solution $\mathbf{p}$ relative to a set A (denoted as $ExcHV(\mathbf{p}, A; \mathbf{z})$) is the size of the part of the objective space that is dominated by $\mathbf{p}$ but is not dominated by any element of A. Hence, $ExcHV(\mathbf{p}, A; \mathbf{z}) < IncHV(\mathbf{p}; \mathbf{z})$. The exclusive hypervolume is defined as:

$$ExcHV(\mathbf{p}, A; \mathbf{z}) := HV(A \cup \{\mathbf{p}\}; \mathbf{z}) - HV(A; \mathbf{z}). \tag{6}$$

In this work, we focus on SMS-EMOA since it is one of the most important HV-based MOEAs [9]. SMS-EMOA is a steady-state MOEA that employs the Pareto dominance relation as its main selection criterion and a density estimator based on the exclusive hypervolume. At each iteration, a single solution is created and added to a temporary population which is divided into layers, using the non-dominated sorting algorithm [7]. If the last layer (the worst one according to the Pareto dominance) has more than one solution, the one having the minimum

Algorithm 1 Naive method to determine the solution with the lowest exclusive hypervolume.

Input: $A \subset Z$ set of solutions, reference point $\mathbf{z}$
Output: Solution $\mathbf{s}$ that contributes the least to $HV(A; \mathbf{z})$

```
1: t ← HV(A; z)
2: m ← ∞
3: for all a ∈ A do
4:    c ← t − HV(A \ {a}; z)
5:    if c < m then
6:       m ← c
7:       s ← a
8:    end if
9: end for
10: return s
```

exclusive hypervolume is deleted. The identification of the minimum exclusive hypervolume value is the core idea behind SMS-EMOA. Software frameworks for evolutionary multi-objective optimization, such as jMetal and MOEA framework implement this step using a naïve approach that iteratively calculates the hypervolume indicator of $|P| - 1$ individuals, as shown in Algorithm 1.

3 IWFG 1.01 Algorithm

In Algorithm 2, we reproduce the pseudocode of IWFG 1.01 [5]. This improved version of Algorithm 1 consists of two phases: the slicing process (lines 1 to 8), and the full computation of the exclusive hypervolume of the least-contributing solution (lines 9 to 14). Here, *head* and *tail* are list functions.[3] In the first phase, *Rank heuristic* imposes the order in which objectives will be processed (in a worsening sequence). The overall HV is then processed in "slices" made by cuts along the corresponding objective. Those slices related to a solution $\mathbf{a}$ are stored in the list $S[\mathbf{a}]$ (line 3). The elements of this list are assumed to be ordered by size from the largest to the smallest. In lines 4 and 5, the biggest slice of a solution $\mathbf{a}$ is successively divided by making $k-1$ cuts along the remaining objectives. These sub-slices are reinserted into the list $S[\mathbf{a}]$. In line 7, for each solution, the partial exclusive hypervolume relative to the biggest slice is determined. In the second phase, a greedy approach is adopted, named "best-first" queuing mechanism. The idea is to process at each iteration the solution $\mathbf{s}$ with the smallest partial HV until its list of slices has been completely processed. Moreover, instead of using the expression (6) for calculating the exclusive hypervolume, IWFG 1.01 uses a more efficient mechanism [2]:

$$ExcHV(\mathbf{p}, A; \mathbf{z}) := HV(\{\mathbf{p}\}; \mathbf{z}) - HV(\mathrm{NDS}(B); \mathbf{z}), \tag{7}$$

where

$$B := \{\mathrm{limit}(\mathbf{p}, \mathbf{a}) \mid \mathbf{a} \in A\}, \tag{8}$$

[3] For instance, $\mathrm{head}([a, b, c, d]) := a$ and $\mathrm{tail}([a, b, c, d]) := [b, c, d]$.

Algorithm 2 Incremental Hypervolume IWFG 1.01

Input: $A \subset Z$ set of solutions, reference point $\mathbf{z}$, depth $k \in \mathbb{N}$
Output: Solution $\mathbf{s}$ that contributes the least to $HV(A; \mathbf{z})$

```
1: for all a in A do
2:    Sort the objectives of a according to Rank heuristic, using the binary search
3:    S[a] ← the slices for a at the top level m
4:    for d = 1 to k − 1 do
5:       S[a] ← slice(head(S[a]), m − d) ∪ tail(S[a])
6:    end for
7:    p[a] ← ExcHV(a, head(S[a]); z)
8: end for
9: s ← arg min_{a∈A} p[a]
10: while S[s] ≠ [ ] do
11:    p[s] ← p[s] + ExcHV(s, head(S[s]); z)
12:    S[s] ← tail(S[s])
13:    s ← arg min_{a∈A} p[a]
14: end while
15: return s
```

$$\begin{aligned}&\text{limit}(< p_1, \ldots, p_m >, < a_1, \ldots, a_m >)\\ &\quad := < \text{worse}(p_1, a_1), \ldots, \text{worse}(p_m, a_m) > .\end{aligned}$$

In Figure 1, we illustrate the two different ways to compute the exclusive hypervolume. In this case, expression (7) calculates the HV of only two solutions, whereas expression (6) considers six solutions. This computational effort reduction is because of the filtering of non-dominated solutions. With these ideas, the IWFG 1.01 algorithm determines the exclusive hypervolume in a fraction of the time needed to process the total HV [5].

4 Contribution

In Figure 2(a), we reproduce a common error of Algorithm IWFG 1.01 using data produced by an MOEA. The program receives as input the file "sample.dat", which contains one front delimited by #, and a reference point with five objectives. As can be noticed, the component throws a fatal error causing an abnormal termination. In this case, the segmentation fault is raised by hardware, which has memory protection, notifying the operating system that the program `iwfg` attempts to access a memory location that is not allowed.

In order to track the source of this error, we relied on the debugging tools Valgrind (`http://valgrind.org`) and gdb (`https://www.gnu.org/software/gdb`). We found that the problem lies in the `binarySearch` function of Figure 3 since it does not contemplate the situation of identical solutions, as it is the case for the objective vector $(0.51, 0.46, 0.73, 0.00, 0.00)$ from our example in Figure 2(a). In evolutionary multi-objective optimization, these copies are known as *indifferent solutions* [4, p. 244], and are occasionally present in a population when variation operators are not applied, so the offspring become clones

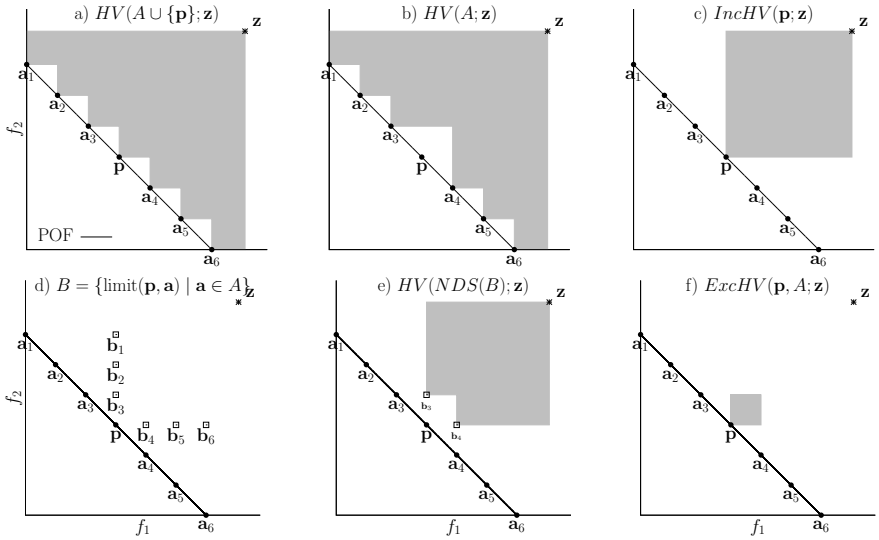

Fig. 1. Steps for the calculation of the exclusive hypervolume of a solution $\mathbf{p}$ using the naive way $HV(A \cup \{\mathbf{p}\}; \mathbf{z}) - HV(A; \mathbf{z})$ and the efficient way $IncHV(\mathbf{p}; \mathbf{z}) - HV(NDS(B); \mathbf{z})$, where $A = \{\mathbf{a}_1, \mathbf{a}_2, \mathbf{a}_3, \mathbf{a}_4, \mathbf{a}_5, \mathbf{a}_6\}$ and $B = \{\mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3, \mathbf{b}_4, \mathbf{b}_5, \mathbf{b}_6\}$.

(a)

```
> cat sample.dat
#
0.00 0.00 0.00 0.00 1.00
0.51 0.46 0.73 0.00 0.00
0.47 0.43 0.46 0.45 0.42
0.00 0.47 0.54 0.70 0.00
0.51 0.46 0.73 0.00 0.00
0.00 0.81 0.00 0.58 0.00
0.93 0.00 0.36 0.00 0.00
#
> ./iwfg sample.dat 1.1 1.1
    1.1 1.1 1.1
Segmentation fault (core
    dumped)
```

(b)

```
> ./iwfg sample.dat 1.1 1.1
    1.1 1.1 1.1
mehv(1) = 0.00
Smallest: 0.51 0.46 0.73 0.00
    0.00
Total time = 0.00 (s)
```

Fig. 2. Output of the IWFG 1.01 component using the functions (a) BinarySearch and (b) ourBinarySearch.

of the parents. According to expression (3), indifferent solutions are considered non-dominated to each other, so the requirement of the IWFG 1.0 to accept only fronts with non-dominated solutions is still fulfilled. The purpose of the function `int binarySearch(POINT p,int d)` is to locate the index `i` at which the solution `p` resides in the array of memory addresses `fsorted[d].points[i]`, assuming that elements are already sorted by the given objective `d` from the highest to the lowest value. The ordering relation is achieved by the function `int`

```
1 int binarySearch(POINT p, int d) {
2   int min = 0;
3   int max = fsorted[d].nPoints-1;
4   gorder = torder[d];
5
6   while(min <= max) {
7     int mid = (max+min)/2;
8     if(p.objectives ==
9        fsorted[d].points[mid].objectives)
10      return mid;
11    else if(greaterorder(&p,
12            &fsorted[d].points[mid]) ==-1)
13      max = mid - 1;
14    else
15      min = mid + 1;
16  }
17  return -1;
18 }
```

Fig. 3. Source code (in ANSI C) of the original binarySearch function.

`greaterorder(&p,&q)`, which numerically compares two solutions. This function returns `-1` if the d^{th} objective of `p` is greater than the d^{th} objective of `q`. In the opposite case, it returns `1`, and if they have the same value, the remainder objectives are inspected in the same way using the order imposed by the Rank heuristic. In the case of indifferent solutions, `greaterorder` returns `0`.

During the search, the error originates when the first occurrence of a repeated solution does not match with the memory address of `p`. Thus, the binary search focuses on the upper half of the array in lieu of examining adjacent elements. So, if the solution is not found in this half, the function returns `-1`, which is an invalid index. It is important to mention that the error happens only from three objectives onwards since for two objectives the exclusive hypervolume is computed. The `binarySearch` function is invoked by the Rank heuristic, which stores the returned misinformation. The slicing process accesses the indices, at which point the fault occurs.

One possible solution to this issue is to include the case when `greaterorder` recognizes two identical solutions. In Figure 4, we present the source code of our proposed correction, named `ourBinarySearch`. Once a duplicated objective vector is found, in lines 17 to 32, adjacent memory locations are inspected until there is a match with the address of `p`. In Figure 2(b), we show the right output of our previous example using the proposed function. It is worth noticing that one of the duplicated solutions is suggested for removal.

The computational complexity of `binarySearch` is $O(m \lg |P|)$, where m represents the number of objectives and $|P|$ is the number of non-dominated solutions in the front. For `ourBinarySearch` is $O(m(\lg |P|+k))$, where k denotes the number of indifferent solutions. Here, the worst case occurs when all elements are repeated, so the computational complexity is $O(m|P|)$. However, this is very unlikely, and in the average case $k << |P|$. Thus, the complexity of our proposed function remains the same as the original one.

```
1int ourBinarySearch(POINT p, int d) {
2  int i, r;
3  int min = 0;
4  int max = fsorted[d].nPoints-1;
5  gorder = torder[d];
6
7  while(min <= max) {
8    int mid = (max+min)/2;
9
10   if(p.objectives ==
11      fsorted[d].points[mid].objectives)
12     return mid;
13   else if((r = greaterorder(&p, &fsorted[d].points[mid])) == -1)
14     max = mid-1;
15   else if(r == 1)
16     min = mid+1;
17   else { /* (r = 0) duplicated solutions */
18     /* check solutions on the left of mid */
19     i = mid - 1;
20     while(i >= min && greaterorder(&p, &fsorted[d].points[i]) == 0) {
21       if(p.objectives == fsorted[d].points[i].objectives)
22         return i;
23       i--;
24     }
25     /* check solutions on the right of mid */
26     i = mid + 1;
27     while(i <= max && greaterorder(&p, &fsorted[d].points[i]) == 0) {
28       if(p.objectives == fsorted[d].points[i].objectives)
29         return i;
30       i++;
31     }
32   }
33 }
34 return -1;
35}
```

Fig. 4. Source code (in ANSI C) of our proposed binarySearch function.

The source code of the IWFG 1.0 algorithm, as well as other algorithms, is available at `http://computacion.cs.cinvestav.mx/~rhernandez`, being the IWFG modules thread-safe. EMO Project runs on Unix-based systems, and it is implemented in ANSI C, MPI (Message Passing Interface) and Gnuplot (`http:/www.gnuplot.info`). As shown in Figure 5, EMO Project is constituted by three main parts: the applications, the EMO library, and the parallelization layer. In addition, there are two special actors: the common user and the developer. The former can invoke predefined applications, while the second can define new problems, implement algorithms and create more applications. The applications consist of a set of command-line programs, which start with the prefix "`emo_`", and their purpose is to perform essential operations in evolutionary multi-objective optimization. The EMO library is composed of a set of built-in functions and structured data types, whereas the parallelization layer allows, among other functionalities, the simultaneous execution of different MOEA calls over a set of available processors using a Round-robin scheme. Interested readers are referred to [11] for further details.

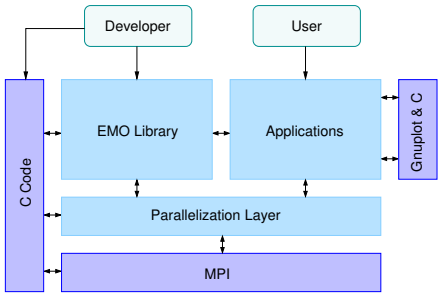

Fig. 5. Architecture of the Evolutionary Multi-objective Optimization Project (EMO Project).

5 Experimental Study

We compared the performance of IWFG 1.01 versus the variant that calculates the HV using the method described in [13], and determines the contributions with Algorithm 1. This latter version, commonly applied in most frameworks, is denoted here as IWFG 1.00. Both versions were coupled to SMS-EMOA. Additionally, we considered in our experiments NSGA-III [6], which was designed to deal with many-objective optimization problems. In the survival selection mechanism of this optimizer, the non-dominated sorting procedure [7] is combined with a niching strategy that requires a set of well-spread reference points in such a way that the population is normalized and associated with the lines passing through the reference points and the origin. Those individuals having the closest perpendicular distance to isolated lines are chosen for the next generation.

As test problems, we adopted the DTLZ1, DTLZ2, and DTLZ7 instances [8] with the number of decision variables of $m+4$, $m+9$, and $m+19$, respectively. The variation operators were Polynomial-based mutation and Simulated Binary Crossover (SBX). For the mutation operator, its probability and distribution index were set to $1/n$ and 20, respectively. For the crossover operator, these parameters varied according to the number of objectives: for two objectives we adopted 0.9 and 20, whereas for higher dimensionality we adopted 1.0 and 30. In all cases, the population size was set to 100 individuals. The set of reference points for NSGA-III was generated using the Uniform Design method [12] having the same cardinality as the population. The maximum number of evaluations (1×10^3) was set to 40, 60, 70, 80, 80, 90 for 2 to 7 objectives, respectively.

For the performance assessment, we relied on the hypervolume indicator using the reference point $(2, 2, \ldots)$ for DTLZ1,2 and $(2, 2, \ldots, 2m+1)$ for DTLZ7. In all experiments, we performed ten independent runs. Besides, we applied two Wilcoxon rank sum tests to the mean hypervolume indicator values in order to determine: A) if the distributions of both variants of SMS-EMOA were identical or different (two-tailed test), and B) if SMS-EMOA IWFG 1.01 performed better than NSGA-III (one-tailed test). Both statistical tests were contemplated at the confidence interval of 99%. Finally, executions have been done over the

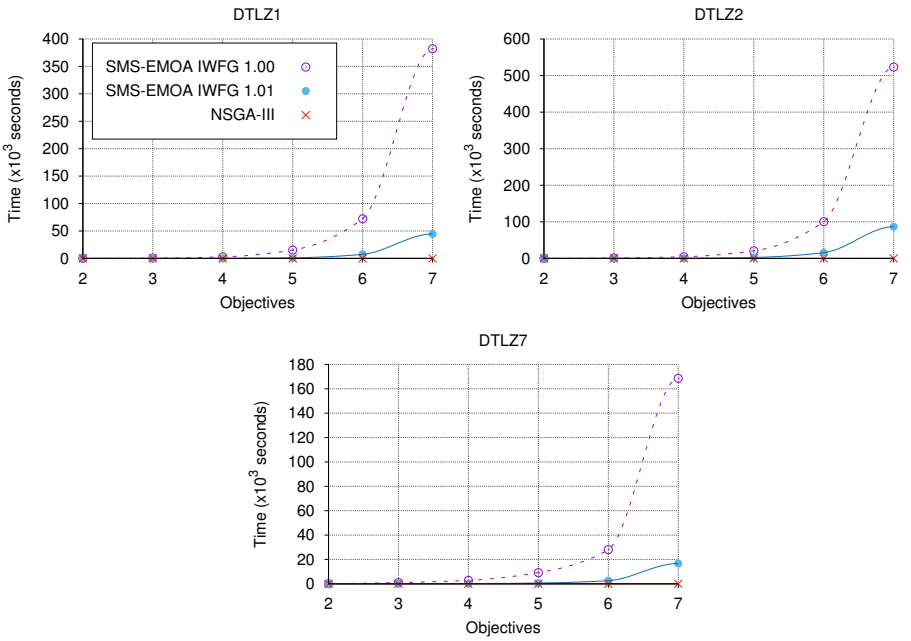

Fig. 6. Average execution time of the MOEAs on some instances of the DTLZ benchmark.

GNU/Linux Xiuhcoatl Cluster of 72 nodes with 252 GB of RAM and InfiniBand interconnection network. Each node is a 32-core AMD Opteron(TM) Processor 6274 1.36 GHz. Algorithms were implemented in the C language, compiled with gcc 4.4.7 -O3 and parallelized with Open MPI version 3.0.0 using the command `emo_task` [11].

In Figure 6, we show the average execution time of all optimizers. As expected, the fastest algorithm was NSGA-III since its computational complexity of $\mathcal{O}(|P|^2m + m^3)$ is much lower than that of HV-based MOEAs. In the second place was SMS-EMOA IWFG 1.01, which spent much less computational time than SMS-EMOA IWFG 1.00. Here, it is worth mentioning that time reduction becomes more significant as the number of objectives increases. Regarding the quality of the solutions in terms of the hypervolume indicator (see Table 1) both versions of SMS-EMOA produced slightly different Pareto-front approximations even though we used the same random seeds. This occurs because, during the survival selection process, several individuals may have the same HV contribution. Thus, the choice of the methods depends on the way in which the population is sorted. In spite of this, there is no significant difference in quality.

Although SMS-EMOA IWFG 1.01 is not the fastest state-of-the-art algorithm, it achieved results which are significantly better than those generated by NSGA-III, in all cases (see Table 1). The only exception is DTLZ7 with four objectives in which there was a tie.

Table 1. Median and standard deviation of the hypervolume indicator. If the p-value of test A is greater than 0.01, then it means that the two samples of SMS-EMOA are equivalent (denoted by =). If the p-value of test B is less or equal than 0.01, then it means that SMS-EMOA IWFG 1.01 performs significantly better than NSGA-III (indicated by ↑).

m	**SMS-EMOA IWFG 1.00**		**SMS-EMOA IWFG 1.01**		**NSGA-III**		p-**value**	
							test A	**test B**
				DTLZ1				
2	3.873652e+0	1.8e-4	3.873610e+0	1.5e-4	3.865911e+0	3.2e-4	9.1e-1 =	9.1e-5 ↑
3	7.974010e+0	4.8e-5	7.974043e+0	8.3e-5	7.937169e+0	1.0e-3	7.1e-1 =	9.1e-5 ↑
4	1.599436e+1	3.1e-5	1.599437e+1	1.2e-5	1.590821e+1	2.9e-3	2.4e-1 =	8.0e-5 ↑
5	3.199857e+1	6.1e-5	3.199859e+1	5.1e-5	3.184074e+1	8.5e-3	4.9e-1 =	9.0e-5 ↑
6	6.399957e+1	2.1e-5	6.399956e+1	2.5e-5	6.368232e+1	4.4e-2	5.2e-1 =	9.0e-5 ↑
7	1.279999e+2	4.2e-5	1.279999e+2	4.8e-5	1.273449e+2	7.5e-2	6.5e-1 =	6.4e-5 ↑
				DTLZ2				
2	3.211015e+0	1.9e-5	3.211003e+0	2.6e-5	3.200341e+0	2.4e-4	6.5e-1 =	9.1e-5 ↑
3	7.427029e+0	5.4e-5	7.427018e+0	5.8e-5	7.317104e+0	5.8e-3	8.5e-1 =	9.1e-5 ↑
4	1.558050e+1	7.7e-5	1.558050e+1	7.7e-5	1.518218e+1	1.8e-2	1.0e+0 =	9.0e-5 ↑
5	3.168567e+1	7.6e-5	3.168567e+1	7.6e-5	3.067499e+1	3.5e-2	1.0e+0 =	9.0e-5 ↑
6	6.375871e+1	1.1e-4	6.375871e+1	1.1e-4	6.145779e+1	9.7e-2	1.0e+0 =	9.1e-5 ↑
7	1.278103e+2	1.4e-4	1.278103e+2	1.4e-4	1.215988e+2	9.0e-1	1.0e+0 =	8.4e-5 ↑
				DTLZ7				
2	4.418220e+0	6.5e-6	4.418217e+0	4.4e-6	4.403340e+0	2.6e-3	1.9e-1 =	8.9e-5 ↑
3	1.357868e+1	1.4e-4	1.357868e+1	1.4e-4	1.312882e+1	6.2e-2	5.2e-1 =	9.1e-5 ↑
4	3.014482e+1	4.2e+0	3.014535e+1	4.2e+0	3.194008e+1	1.4e+0	6.9e-1 =	2.9e-1
5	7.769183e+1	5.8e+0	7.769183e+1	5.8e+0	6.765668e+1	2.2e+0	5.2e-1 =	9.1e-5 ↑
6	2.057445e+2	1.0e+1	1.867199e+2	1.0e+1	1.374160e+2	6.3e+0	2.7e-1 =	9.1e-5 ↑
7	4.257511e+2	3.3e+1	4.264531e+2	2.7e+1	2.601411e+2	1.6e+1	7.2e-1 =	9.1e-5 ↑

6 Conclusions

Recently, an optimized version of the incremental hypervolume algorithm of the Walking Fish Group (IWFG) was proposed. This algorithm determines the solution that contributes the least to the HV of a non-dominated set. However, its use has been limited due to a bug in its implementation. We observed that this error occurs during the slicing process, specifically in the function `binarySearch`, where duplicated solutions are not considered for problems with more than two objectives. In this paper, we have proposed a corrected version of such function, which has an average-case complexity of $O(m \lg |P|)$, where m denotes the number of objectives and $|P|$ the population size. Clearly, there are other possible solutions to this issue, such as to remove duplicated solutions before calculating the incremental hypervolume. However, we have presented the one that we believe is the easiest to update in the component while keeping a low computational cost. The potential performance of the IWFG component should be exploited by the many-objective community since it can achieve high-quality solutions at an affordable computational cost.

References

1. Beume, N., Naujoks, B., Emmerich, M.: SMS-EMOA: Multiobjective Selection based on Dominated Hypervolume. European Journal of Operational Research **181**(3), 1653–1669 (16 September 2007)
2. Bradstreet, L., While, L., Barone, L.: A New Way of Calculating Exact Exclusive Hypervolumes. Tech. Rep. UWA-CSSE-09-002, The University of Western Australia, School of Computer Science and Software Engineering (2009)
3. Bradstreet, L., While, L., Barone, L.: A Fast Incremental Hypervolume Algorithm. IEEE Transactions on Evolutionary Computation **12**(6), 714–723 (December 2008)
4. Coello Coello, C.A., Lamont, G.B., Van Veldhuizen, D.A.: Evolutionary Algorithms for Solving Multi-Objective Problems. Springer, New York, second edn. (September 2007), iSBN 978-0-387-33254-3
5. Cox, W., While, L.: Improving the IWFG Algorithm for Calculating Incremental Hypervolume. In: 2016 IEEE Congress on Evolutionary Computation (CEC). pp. 3969–3976 (July 2016)
6. Deb, K., Jain, H.: An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. IEEE Transactions on Evolutionary Computation **18**(4), 577–601 (Aug 2014)
7. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA–II. IEEE Transactions on Evolutionary Computation **6**(2), 182–197 (April 2002)
8. Deb, K., Thiele, L., Laumanns, M., Zitzler, E.: Scalable Test Problems for Evolutionary Multiobjective Optimization. In: Abraham, A., Jain, L., Goldberg, R. (eds.) Evolutionary Multiobjective Optimization, pp. 105–145. Advanced Information and Knowledge Processing, Springer London (2005)
9. Falcón-Cardona, J.G., Coello, C.A.C.: Indicator-Based Multi-Objective Evolutionary Algorithms: A Comprehensive Survey. ACM Comput. Surv. **53**(2) (2020). https://doi.org/10.1145/3376916, `https://doi.org/10.1145/3376916`
10. Fleischer, M.: The Measure of Pareto Optima Applications to Multi-objective Metaheuristics. In: Fonseca, C., Fleming, P., Zitzler, E., Thiele, L., Deb, K. (eds.) Evolutionary Multi-Criterion Optimization, Lecture Notes in Computer Science, vol. 2632, pp. 519–533. Springer Berlin Heidelberg (2003)
11. Hernández Gómez, R.: Parallel Hyper-Heuristics for Multi-Objective Optimization. Ph.D. thesis, CINVESTAV-IPN, Mexico City, Mexico (March 2018)
12. Tan, Y., Jiao, Y., Li, H., Wang, X.: MOEA/D plus Uniform Design: A New Version of MOEA/D for Optimization Problems with Many Objectives. Computers & Operations Research **40**(6), 1648–1660 (June 2013)
13. While, L., Bradstreet, L., Barone, L.: A Fast Way of Calculating Exact Hypervolumes. IEEE Transactions on Evolutionary Computation **16**(1), 86–95 (Feb 2012)
14. While, L., Bradstreet, L.: Applying the WFG Algorithm to Calculate Incremental Hypervolumes. In: 2012 IEEE Congress on Evolutionary Computation (CEC'2012). pp. 489–496. IEEE Press, Brisbane, Australia (June 10-15 2012)
15. Zitzler, E.: Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications. Ph.D. thesis, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland (November 1999)
16. Zitzler, E., Thiele, L., Laumanns, M., Fonseca, C.M., da Fonseca, V.G.: Performance Assessment of Multiobjective Optimizers: An Analysis and Review. IEEE Transactions on Evolutionary Computation **7**(2), 117–132 (April 2003)