Improving Multi-Objective Evolutionary Algorithms using Grammatical Evolution

Amín V. Bernabé Rodríguez[a], Braulio I. Alejo-Cerezo[b], Carlos A. Coello Coello[a]

[a] *CINVESTAV-IPN, Computer Science Department, Mexico City, 07360, Mexico*
[b] *UPIITA-IPN, Mexico City, 07340, Mexico*

Abstract

Multi-objective evolutionary algorithms (MOEAs) have become an effective choice to solve multi-objective optimization problems (MOPs). However, it is well known that Pareto dominance-based MOEAs struggle in MOPs with four or more objective functions due to a lack of selection pressure in high dimensional spaces. The main choices for dealing with such problems are decomposition-based and indicator-based MOEAs. In this work, we propose the use of Grammatical Evolution (an evolutionary computation search technique) to generate functions that can improve decomposition-based and indicator-based MOEAs. Namely, we propose a methodology to generate new scalarizing functions, which are known to have a great impact in the performance of decomposition-based MOEAs and in some indicator-based MOEAs. Additionally, we propose another methodology to generate hypervolume approximations, since the hypervolume is a popular performance indicator used not only in indicator-based MOEAs but also to assess performance of MOEAs. Using our first methodology, we generate two new scalarizing functions and provide their corresponding experimental validation to show that they exhibit a competitive behavior when compared against some well-known scalarizing functions such as ASF, PBI and the Tchebycheff scalarizing function. Using our second methodology, we produce 4 different hypervolume approximations and compare their performance against the Monte Carlo method and against two other state-of-the-art hypervolume ap-

Email addresses: amin.bernabe@cinvestav.mx (Amín V. Bernabé Rodríguez), balejoc1400@alumno.ipn.mx (Braulio I. Alejo-Cerezo), ccoello@cs.cinvestav.mx (Carlos A. Coello Coello)

proximations. The experimental results show that our functions exhibit a good compromise in terms of quality and execution time.

Keywords: Grammatical evolution, Genetic programming, Evolutionary algorithms, Multi-objective optimization

---

1. Introduction

Multi-objective optimization problems (MOPs) are those in which two or more objective functions, usually in conflict, require to be simultaneously optimized. Many real-world problems are indeed MOPs [1, 2]. A MOP is formally defined as follows:

$$\text{minimize } \vec{f}(\vec{x}) := [f_1(\vec{x}), f_2(\vec{x}), \ldots, f_k(\vec{x})] \tag{1}$$

subject to:

$$g_i(\vec{x}) \leq 0 \quad i = 1, 2, \ldots, p \tag{2}$$

$$h_i(\vec{x}) = 0 \quad i = 1, 2, \ldots, q \tag{3}$$

where $\vec{x} = [x_1, x_2, \ldots, x_n]^T$ is the vector of decision variables, $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$, $i = 1, ..., k$ are the objective functions and $g_i, h_j : \mathbb{R}^n \rightarrow \mathbb{R}$, $i = 1, ..., p$, $j = 1, ..., q$ are the constraint functions of the problem.

Given that the objective functions are in conflict, there is no single solution to a MOP. Instead of that, we attempt to generate a set of solutions that represent the best possible trade-offs among the objectives. In order to characterize such solutions, Pareto dominance is commonly used.

Definition 1. Given two vectors $\vec{x}, \vec{y} \in \mathbb{R}^k$, we say that $\vec{x} \leq \vec{y}$ if $x_i \leq y_i$ for $i = 1, ..., k$, and that $\vec{x}$ **dominates** $\vec{y}$ (denoted by $\vec{x} \prec \vec{y}$) if $\vec{x} \leq \vec{y}$ and $\vec{x} \neq \vec{y}$.

Definition 2. We say that a vector of decision variables $\vec{x} \in \mathcal{X} \subset \mathbb{R}^n$ is **non-dominated** with respect to $\mathcal{X}$, if there does not exist another $\vec{x}' \in \mathcal{X}$ such that $\vec{f}(\vec{x}') \prec \vec{f}(\vec{x})$.

Definition 3. We say that a vector of decision variables $\vec{x}^* \in \mathcal{F} \subset \mathbb{R}^n$ ($\mathcal{F}$ is the feasible region) is **Pareto-optimal** if it is non-dominated with respect to $\mathcal{F}$.

Definition 4. The **Pareto Optimal Set** $\mathcal{P}^*$ is defined by:

$$\mathcal{P}^* = \{\vec{x} \in \mathcal{F} | \vec{x} \text{ is Pareto-optimal}\}$$

Definition 5. The **Pareto Front** $\mathcal{PF}^*$ is defined by:

$$\mathcal{PF}^* = \{\vec{f}(\vec{x}) \in \mathbb{R}^k | \vec{x} \in \mathcal{P}^*\}$$

Then, the goal of solving a given MOP is to find the Pareto optimal set ($\mathcal{P}^*$) from the feasible region ($\mathcal{F}$), defined by (2) and (3).

Some of the most popular techniques used to solve MOPs are the Multi-objective Evolutionary Algorithms (MOEAs). These are population-based methods, which allow to obtain a set of solutions in a single execution. This represents a clear advantage with respect to traditional mathematical programming techniques, which usually produce a single solution per execution. MOEAs are also more general in the sense that they require little domain-specific information and do not impose requirements on the objective functions (e.g., they don't need to be differentiable nor being defined in algebraic form [3]).

There is a wide variety of MOEAs in the specialized literature, but they can be broadly classified into 3 categories: Pareto dominance-based, decomposition-based and indicator-based MOEAs [4]. In this work, we present the generation of new elements which could improve the performance of MOEAs in the last two of these categories. Because of that, we will briefly discuss next both decomposition-based and indicator-based MOEAs.

Decomposition-based MOEAs work by transforming a MOP into two or more single-objective optimization problems, which are solved simultaneously using neighborhood search [5]. One of the advantages of this technique is that they are not easily affected by selection pressure issues [6], which makes them particularly useful in MOPs with many objectives (4 or more). These MOEAs use scalarizing functions to aggregate the multiple objective functions into a single function. There are multiple scalarizing functions with different properties, such as the optimality of the solutions found (weak/strong Pareto optimality), as well as with different requirements (reference points, utopian/Nadir objective vector, aspiration levels or additional classifications) [7].

The performance of a decomposition-based MOEA is closely related to both, the scalarizing function adopted, which can determine the type of solutions found [8], as well as the weight vectors used, since they strongly determine the distribution of such solutions [9]. In this work, we propose

a methodology to produce new scalarizing functions to improve the performance of these MOEAs.[1] It is important to mention that here, we extend the methodology originally presented in [11], by using a different (improved) implementation, as well as by including a comprehensive series of experiments and a new application which is described next.

Indicator-based MOEAs adopt performance indicators to measure the quality of a given individual with respect to the rest of the population and uses this information to assign its fitness value. The individual with the worst contribution is usually deleted from the population and replaced in the next generation [12]. One of the most widely used indicators in these algorithms is the hypervolume. This is due to the fact that the hypervolume is Pareto compliant, which means that it preserves the order imposed by the Pareto dominance relation [13]. Additionally, it provides both convergence and (to some extent) diversity information, since a set with a better coverage of the Pareto front will have a better hypervolume value. Nonetheless, the main drawback of the hypervolume is that its computational cost increases exponentially with the number of objectives. This has motivated a lot of research aiming to find more efficient ways of calculating it [14, 15] or ways of approximating it [16, 17, 18] to reduce its computational time. In this paper, we propose the use of our methodology based on genetic programming to produce hypervolume approximations with the same goal of decreasing its computational time.

In order to generate new scalarizing functions as well as hypervolume approximations, we use a variant of Genetic Programming (GP), which is a well-known evolutionary computation technique that allows us to evolve computer programs to solve a given problem. This is achieved by genetically breeding a population of computer programs and applying genetic operators iteratively until a certain termination criteria is met [19].

Grammatical Evolution (GE) is a grammar-based form of GP, which utilizes a different genotype-phenotype mapping. Similar to GP, it also starts by breeding a population of computer programs which are potential solutions to a given optimization problem, which is encoded in a fitness function. Using this fitness function, all individuals in the population are evaluated and are assigned a corresponding fitness value, which allows a selection mechanism

[1] It is worth mentioning that indicator-based MOEAs based on $R2$ also use scalarizing functions [10].

to compare them and to choose which of them should be propagated across the next generation of individuals. Then, they are recombined accordingly using genetic operators, such as crossover and mutation, to create an offspring population. This process is repeated until the termination criteria is met. However, whereas GP usually manipulates a tree data structure, GE genotypes usually consist of integer or binary lists [20]. Consequently, one of the main advantages of GE is that it can be easily applied to different problem domains, and all we need to adapt are two components: (1) a grammar to define the syntax of potential solutions, usually given in a Backus-Naur form, and (2) a fitness function to evaluate such solutions[21].

Our main motivation in this work is to propose an implementation that allows the automatic generation of components that could potentially improve the performance of MOEAs. Thus, we chose a Python-coded GE implementation that can be easily modified not only to generate the components that we propose in this work, but also to generate different elements that can be used in different MOEAs. Hence, we can summarize the contributions of this work as follows:

- We propose a GE implementation to generate new scalarizing functions in a relatively simple manner and with a great capability to adapt to different training setups.
- Using the implementation mentioned in the previous point, we generated two new scalarizing functions and we performed their corresponding experimental validation. Our experiments show that these two scalarizing functions are able to outperform other scalarizing functions such as the achievement scalarizing function, the Tchebycheff scalarizing function and penalty-boundary intersection.
- We propose another variant of the GE implementation which allows to produce new hypervolume approximations.
- Using the implementation indicated in the previous point, we describe the methodology that we used to generate two different hypervolume approximations for data in 2, 3, 4 and 5-dimensional spaces, producing a total of 8 different hypervolume approximations.

The remainder of this paper is organized as follows. In Section 2, we present some previous related work. In Section 3 we present the methodolo-

gies used to automatically generate new scalarizing functions and hypervolume approximations. Then, in Section 4 we provide the experimental setup that we use to generate our scalarizing functions and hypervolume approximations, along with their definition. Next, we validate them experimentally in Section 5. Finally, we present our conclusions and some possible paths for future work in Section 6.

2. Previous Related Work

There are several works involving the use of GP to automatically generate components that improve different Artificial Intelligence algorithms. For instance, a GP based system (EvoCK) [22] has been combined with a Machine Learning (ML) algorithm specialized in planning (Hamlet) [23], giving rise to Hamlet-EvoCK [24], which alleviates some of the handicaps of the original approach. One of these handicaps is that the ML technique can refine its behavior when being presented an appropriate set of examples. However, when the example-space is large, it deteriorates its performance due to the computational time required. Then, in Hamlet-EvoCK, the authors propose to use the GP system to generate examples with certain characteristics, such as simplicity, while also evolving the population of examples based on a fitness function that measures the efficiency of such examples. Another example is the use of GP to automatically develop Artificial Neural Networks (ANNs), which can be done by evolution of the weights, by evolution of the architectures or by evolution of the learning rules. In [25] and [26] two proposals are made to achieve this automatic generation of ANNs requiring minimal to no human intervention and obtaining either average or better results than other automatic ANN generation techniques. More recently, a GP hyperheuristic was proposed in [27] to evolve scheduling heuristics for dynamic flexible job-scheduling problems. The generation of these heuristics is performed only with the selected features determined by the GP, since feature selection is a key part of this process.

In the context of automatic generation of scalarizing functions, to the best of the authors' knowledge, the only related work is the one presented in [11]. In that work, a hybrid implementation is presented combining EllenGP [28], which is a GP implementation with local search used to generate scalarizing functions, and MOMBI-II [29], which is a MOEA adopted to assess the performance of the scalarizing functions generated. The search process of these new scalarizing functions involves coding the new functions into MOMBI-II

and use it to solve a given MOP. In this work, we use a very similar methodology, but we extend the number of training MOPs solved in the search process to 2 instead of only 1, along with some other modifications such as the addition of a threshold to reduce the number of function evaluations invested in scalarizing functions which obtain bad results in a sample MOP. The increase in the number of training MOPs is proposed to favor the generation of scalarizing functions with a good performance in MOPs different from the one used in the training process.

Regarding the generation of hypervolume approximations using a similar technique, a GP-based methodology was recently proposed to approximate, in an efficient way, the hypervolume value and the hypervolume contribution (used for indicator-based MOEAs) in 3, 4 and 5 dimensional spaces [30]. Given a set of training data, they adopted statistical analysis to obtain certain statistical features used as the variables given to the GP. In this work, we present a different variant to generate hypervolume approximations. Instead of obtaining information from the population as a whole, and building approximation functions with such information, we consider point-wise information. This means that we evaluate the approximation functions iteratively with each of the points in the data. This increases the computational time $\mathcal{O}(n)$, where n is the number of points in the data, but also improves the quality of the approximation.

3. Proposed implementation

In this work, we decided to use `PonyGE2`, which is a grammatical evolution implementation in Python [31]. Being a population-based evolutionary algorithm, GE iterates during a search loop in which individuals are evaluated, selected, recombined and mutated until a certain (given) criterion is satisfied. In our experiments, this termination criterion is a maximum number of generations.

In this section, we describe the two main components needed to implement `PonyGE2`, which are the fitness function to guide the search process and the grammar to generate the individuals. In the following subsections we describe each of these elements for the generation of new scalarizing functions as well as for the generation of the hypervolume approximations.

3.1. Scalarizing functions

The GE implementation that we adopted handles the creation and reproduction of individuals that encode certain functions defined by a given grammar. Therefore, our main contribution is the methodology used to define how well each of these automatically created functions works as a scalarizing function. To do this, we adopted two components: a MOEA that uses scalarizing functions (MOMBI-II) and a performance indicator (hypervolume) to measure the solutions generated. An overview of this process is shown in Fig 1.

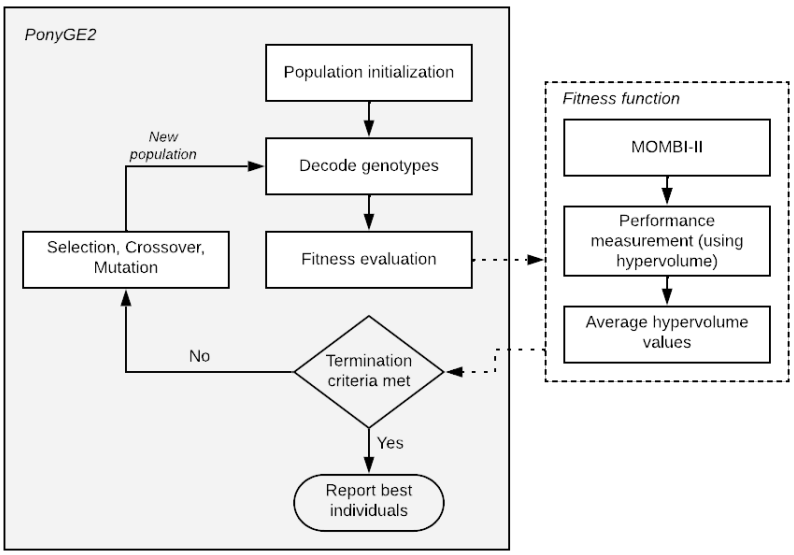


Figure 1: Diagram of grammatical evolution used to generate scalarizing functions.

MOMBI-II is a metaheuristic developed to solve many-objective optimization problems [29]. We adopted MOMBI-II in our implementation because it is a competitive algorithm which has shown a good performance when compared against state-of-the-art MOEAs in benchmark problems with up to 10 objectives. Additionally, it uses scalarizing functions, since it is based on the $R2$ indicator. By default, it uses the Achievement Scalarizing Function (ASF) [32] which is defined as follows:

$$ASF(\vec{f}(\vec{x}), \vec{w}) := \max_{i \in 1,\ldots,m} \left(\frac{f_i(\vec{x})}{w_i} \right) \tag{4}$$

where $\vec{f}(\vec{x})$ is the image of $\vec{x}$ in objective space and $\vec{w} \in \mathbb{R}^m$ is a weight vector. However, other scalarizing functions can be used, such as the Tchebycheff (TCH) scalarizing function [33], defined as follows:

$$TCH(\vec{f}(\vec{x}), \vec{w}) := \max_{i \in 1,\ldots,m} (w_i |f_i(\vec{x})|) \,. \tag{5}$$

Another commonly used scalarizing function in decomposition-based MOEAs is the Penalty Boundary Intersection (PBI) [34], defined as follows:

$$PBI(\vec{f}(\vec{x}), \vec{w}) := \theta d_2 - d_1 \tag{6}$$

where $d_1 := \left| \vec{f}(\vec{x}) \cdot {}^{\vec{w}}/_{\|\vec{w}\|} \right| \quad , \quad d_2 := \left\| \vec{f}(\vec{x}) - d_1 {}^{\vec{w}}/_{\|\vec{w}\|} \right\|$ and θ is a penalty parameter, usually set by default as $\theta = 5$.

We implemented each of the new scalarizing functions generated by `PonyGE2`, replacing ASF. Then, we used this modified version of MOMBI-II to solve multiple MOPs. Finally, we used the hypervolume to assess the quality of the Pareto fronts obtained. The average hypervolume values obtained were used to set the fitness of the new function.

The fitness function that we propose to guide the search process of `PonyGE2` requires the following information:

- The parameters needed to define the training MOP. This includes the MOP, the number of objectives, the number of executions and the maximum number of function evaluations. For example, we could define that the training MOP is DTLZ4, with 2 objectives, considering 30 executions with a maximum number of 50,000 function evaluations each.

- The maximum hypervolume value for each of the MOPs defined, along with the reference point used to obtain such hypervolume value. This value is used to normalize hypervolume results. Following the previous example, the maximum hypervolume for DTLZ4 with 2 objectives can be set to 0.210 considering the reference point [1,1].

These parameters are enough for GE to generate new scalarizing functions. However, we included a step where we solve the training MOP with a smaller number of function evaluations and a lower number of executions to improve the search speed, because the use of the hypervolume can make the process

computationally expensive. Thus, we need the following additional parameters:

- The parameters needed to define the sample MOP, which is the same as the training MOP, but with a lower number of function evaluations and a lower number of executions as well. This sample MOP is used to determine if the current function is worth performing more evaluations. Once again, following the previous example, we define the sample MOP to be DTLZ4, with 2 objectives, but considering 15 executions with a maximum number of 10,000 function evaluations.

- The maximum hypervolume value corresponding to the sample MOP. We need this since a lower number of function evaluations results in a different maximum hypervolume value.

- A hypervolume threshold that defines the percentage of the real maximum hypervolume that should be attained by the executions performed using the sample MOP in order to perform the full executions of the main training MOP. In case this threshold is not attained, no more function evaluations are spent in the current individual.

In Algorithm 1, we show the fitness function in detail, given the decoded phenotype of the individual (i.e., the scalarizing function) as well as the aforementioned parameters. We start by initializing variables (lines 1-2). Then, we use MOMBI-II with the new scalarizing function to solve the sample MOP. Then, we obtain the hypervolume of the corresponding Pareto front and we store this value. This process is repeated (lines 3-7) to obtain an average hypervolume (line 8). If the average hypervolume is below some percentage of the real hypervolume (defined by the threshold parameter) we normalize the average hypervolume (line 10) and then we penalize this value by dividing it by the number of MOPs (line 11). This penalty is simply intended to significantly decrease the fitness of individuals that do not reach the threshold in order to avoid the propagation of such individuals in the population. We chose to divide the current fitness by the number of MOPs as it is a simple way of decreasing the fitness while still keeping a remainder of the original fitness in case that most of the population contains bad individuals, which may occur in the first generations of the execution. In the event that the average hypervolume is greater than the desired threshold, we proceed to evaluate each of the desired MOPs using the new scalarizing

function (lines 15-19). For each MOP, after averaging its hypervolume (line 20) and normalizing it (line 21), we cumulatively store this value to be the individual's final fitness (line 22).

Algorithm 1: Fitness function used to generate scalarizing functions

Input : phenotype, $M\vec{O}P = \{mop_1, \ldots, mop_j\}$, $H\vec{V} = \{hv_1, \ldots, hv_j\}$, $\vec{N} = \{n_1, \ldots, n_j\}, sample\_mop, sample\_hv, sample\_n, threshold$;
Output: fitness;

```
1  fitness ← 0;
2  avg_hv ← 0;
3  for i ← 0 to sample_n do
4  |   PF_i ←MOMBI2(sample_mop, phenotype);
5  |   aux_hv ← compute hypervolume value of PF_i;
6  |   avg_hv ← avg_hv + aux_hv;
7  end
8  avg_hv ← avg_hv/sample_n;
9  if avg_hv < threshold * sample_hv then
10 |   fitness ← avg_hv/sample_hv;
11 |   fitness ← fitness/size(MOP);
12 else
13 |   foreach mop ∈ MOP do
14 |   |   avg_hv ← 0;
15 |   |   for i ← 0 to n_j do
16 |   |   |   PF_i ←MOMBI2(mop_i, phenotype);
17 |   |   |   aux_hv ← compute hypervolume value of PF_i;
18 |   |   |   avg_hv ← avg_hv + aux_hv;
19 |   |   end
20 |   |   avg_hv ← avg_hv/n_i;
21 |   |   avg_hv ← avg_hv/hv_i ;
22 |   |   fitness ← fitness + avg_hv;
23 |   end
24 end
25 return fitness;
```

When incorporating the new scalarizing functions in MOMBI-II we used

the max operator in the following way. Given the decoded phenotype to be $SF(f_i(\vec{x}), w_i)$, where $\vec{f}(\vec{x})$ is the objective functions vector and $\vec{w}$ is the weight vector, we obtain the final Grammatical Evolution Scalarizing Function (GE_SF) as follows:

$$GE\_SF(\vec{f}(\vec{x}), \vec{w}) := \max_{i \in 1,\ldots,m} (SF(f_i(\vec{x}), w_i)). \tag{7}$$

The grammar used to generate the phenotypes consists of basic arithmetic operations and the square root, as shown below. We deliberately chose not to include more complex functions such as trigonometric functions, because some previous experiments showed that such functions generate really specific scalarizing functions which are not able to generalize a good performance in problems different to the ones used in the training.

$$\begin{aligned} \langle e \rangle \quad &::= \langle e \rangle + \langle e \rangle \mid \langle e \rangle - \langle e \rangle \mid \langle e \rangle * \langle e \rangle \mid \langle e \rangle \,/\, \langle e \rangle \\ &\mid \text{sqrt}(\langle e \rangle) \mid \langle c \rangle\langle c \rangle.\langle c \rangle\langle c \rangle \mid f_i(\vec{x}) \mid w_i \end{aligned}$$

$$\langle c \rangle \quad ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$

3.2. Hypervolume approximations

Let $A = \{x_1, \ldots, x_n\} \in \mathbb{R}^m$ be a set of points in an $m-$dimensional space, and let $r \in \mathbb{R}^m$ be a reference point which is dominated by every point in A. Then, the set $H(A, r)$ is formed by all the points that are dominated by at least one element in A and which also dominate r:

$$H(A, r) = \{z \in \mathbb{R}^m \mid \exists\, x \in A : x \prec z \prec r\}. \tag{8}$$

The hypervolume indicator $I_H(A, r)$ is defined as $I_H(A, r) = \lambda(H(A, r))$, where λ represents the Lebesgue measure [35]. In order to compare the hypervolume values of two different sets, the calculation must be made using the same reference point r.

3.2.1. Monte Carlo approximation

One of the most popular methods to approximate the hypervolume value of a non-dominated set is the Monte Carlo approach [16]. Given a data set $X = \{x_1, \ldots, x_n\}$, $x_i \in \mathbb{R}^m$ we can approximate its hypervolume by sampling a given number of points in the region delimited by X. Then, for each sample point we determine if it is dominated by any x_i, counting the dominated

sample points in variable *hits*. Then, the Monte Carlo approximation is obtained using the following expression:

$$MC(X) = \lambda(X) * \frac{hits}{\text{sample size}}. \tag{9}$$

The quality of the approximation depends on the sample size: the larger the sample is, the better the approximation results. However, the computational cost increases proportionally to this value.

3.2.2. GP generated approximations

In the work presented in [30], 3 different approximations are proposed to approximate the hypervolume value in 3, 4 and 5-dimensional spaces. For each data set $X \in \mathbb{R}^{n \times m}$, there are some statistical features that need to be computed to obtain its approximation. These features are shown in Table 1. All of them are obtained for each of the m objectives in the data.

Table 1: Statistical features extracted from data to obtain its hypervolume approximation.

Notation	Statistical feature
$\vec{\mu}$	mean
$\vec{\sigma}$	standard deviation
$\vec{Q1}$	1st quartile
$\vec{Q2}$	2nd quartile
$\vec{Q3}$	3rd quartile
$\vec{\kappa}$	kurtosis
$\vec{\lambda}$	skewness

The hypervolume approximation functions for 3, 4 and 5 dimensions are defined as shown in eqns (10), (11) and (12) respectively. These equations were extracted from the source code provided in citesandoval22.

$$
\begin{aligned}
M^3_{4,6}(X) = & \frac{6236736683876353 * \vec{\mu}_1 * \vec{\mu}_2 * \vec{Q1}_1}{4503599627370496} \\
& - \frac{3173082762593237 * \vec{\mu}_2}{2251799813685248} - \frac{1771343023238655 * \vec{\mu}_3}{2251799813685248} \\
& - \frac{5290754323525305 * \vec{\sigma}_1}{4503599627370496} - \frac{8680371570911475 * \vec{Q1}_1}{36028797018963968} \\
& - \frac{2893457190303825 * |\vec{\gamma}_3|}{36028797018963968} - \frac{5906898887207671 * log(\vec{\kappa}_1)}{36028797018963968} \\
& - \frac{6829965597182259 * log(\vec{\sigma}_3)}{9007199254740992 * log(10)} \\
& - \frac{2294397315973779 * log(\vec{\kappa}_2 + \vec{\gamma}_2 + \vec{Q2}_2 * \vec{\gamma}_1)}{18014398509481984 * log(10)} \\
& - \frac{2893457190303825 * \vec{Q2}_3 * \vec{\gamma}_1}{36028797018963968} - \frac{6826873663546647 * \vec{Q2}_3 * \vec{\gamma}_2}{36028797018963968} \\
& - \frac{6066879653575323 * \vec{\mu}_1}{4503599627370496} + \frac{1215041356731309 * \vec{Q2}_3 * \vec{Q1}_1 * \vec{\gamma}_2}{4503599627370496} \\
& + \frac{5647080252797291}{2251799813685248}
\end{aligned}
\tag{10}
$$

$$
\begin{aligned}
M_{5,6}^{4}(X) = {} & \frac{3818342324243663 * (\vec{\sigma}_{3}^{9})^{1/2}}{17592186044416} - \frac{8367714107802115 * \vec{Q1}_{4}}{9007199254740992} \\
& - \frac{4684844483679697 * \sin(\sin(\vec{\mu}_{2} * \vec{\sigma}_{4}))}{1125899906842624} \\
& - \frac{4831284340333437 * \sin((\vec{\sigma}_{2}^{2} * \vec{\gamma}_{2}) / \cos(\vec{Q1}_{3}))}{4503599627370496} \\
& - \frac{3387877962052739 * exp(exp(\vec{\sigma}_{3}^{3}))}{70368744177664} - \frac{8649112683782109 * \vec{Q1}_{3}}{18014398509481984} \\
& + \frac{2421873864284579}{35184372088832 * \cos(\vec{\sigma}_{3})} + \frac{188607530293811 * 1 / \tan(\vec{\sigma}_{4})^{1/4}}{562949953421312} \\
& - \frac{12795601803451 * \vec{\mu}_{1} * \vec{\sigma}_{3}}{8796093022208} \\
& - \frac{3714273495887619 * \vec{Q1}_{2} * \cos(\vec{Q1}_{2}) * \sin(\vec{\gamma}_{4})}{18014398509481984} \\
& + \frac{8842052222550475}{140737488355328}
\end{aligned}
\tag{11}
$$

$$
\begin{aligned}
M^5_{1,5}(X) = & \frac{226757085449091 * \vec{\mu}_4}{562949953421312} - \frac{49212418854775 * \vec{\mu}_1}{140737488355328} \\
& + \frac{316327101333597 * \vec{Q2}_2}{562949953421312} + \frac{4208801459950455 * \vec{Q2}_3}{18014398509481984} \\
& + \frac{4208801459950455 * \vec{\sigma}_2}{9007199254740992} - \frac{4420306567464985 * \vec{\sigma}_3}{281474976710656} \\
& + \frac{1059206653244855 * \vec{Q1}_5}{18014398509481984} + \frac{50672178223625 * \vec{Q3}_1}{70368744177664} \\
& + \frac{8274937251716701 * \sin^{-1}(\sin^{-1}(\vec{Q2}_2 * \vec{\sigma}_3))}{2251799813685248} \\
& - \frac{8229011520467723 * \sin^{-1}(\vec{\sigma}_3^2)}{562949953421312} \\
& + \frac{2896651095433129 * \sin^{-1}(\tan(\vec{\sigma}_3))}{281474976710656} \\
& + \frac{501121421306637 * \cos(\vec{Q2}_2 + \vec{Q2}_3 - \vec{Q1}_5 + \vec{Q3}_4 + \vec{Q3}_5 + log(\vec{\kappa}_3))}{4503599627370496} \\
& + \frac{5140569679074477 * \cos(\vec{\mu}_3 + \vec{\mu}_4 + \vec{Q1}_5 + \vec{Q3}_1 + |\vec{\sigma}_2|)}{4503599627370496} \\
& - \frac{49212418854775 * \tan(\vec{Q2}_2)}{281474976710656} + \frac{2896651095433129 * \tan(\vec{\sigma}_3)}{281474976710656} \\
& - \frac{49212418854775}{281474976710656 * \cos(\vec{Q3}_5)} + \frac{4208801459950455 * \vec{Q3}_1 * \vec{Q3}_4}{18014398509481984} \\
& - \frac{49212418854775 * \vec{Q1}_2 * \vec{\gamma}_5}{281474976710656} \\
& - 5.22905069610532887168119486887 01 * \vec{Q2}_2 * \sin(\vec{\sigma}_5) \\
& + 0.27075243546777693026683664356824.
\end{aligned}
\tag{12}
$$

3.2.3. R2 hypervolume approximation

Another recent proposal to approximate the hypervolume value of a non-dominated set as well as the individual hypervolume contribution is based on the use of the R2 indicator [36]. This is a linear-based approach, defined as follows.

$$R_2^{HV}(X, \Lambda, \vec{r}, m) = \frac{1}{|\Lambda|} \sum_{\vec{\lambda} \in \Lambda} \max_{\vec{x} \in X} \left(g^{mtch}(\vec{x}|\vec{\lambda}, \vec{r}) \right)^m \tag{13}$$

where X is a set of non-dominated solutions $\vec{x} \in \mathbb{R}^m$, Λ is a set of direction vectors, each direction vector $\vec{\lambda} = \{\lambda_1, \ldots, \lambda_m\} \in \Lambda$ satisfies $\|\lambda\|_2 = 1$ and $\lambda_i \geq 0,\ i = 1, \ldots, m,\ \vec{r} = \{r, \ldots, r\}$ is the reference point and m is the dimensionality of the space. The function g^{mtch} is defined as follows.

$$g^{mtch}(\vec{x}|\vec{\lambda}, \vec{r}) = \min_{j \in 1, \ldots, m} \left(\frac{|r_j - a_j|}{\lambda_j} \right). \tag{14}$$

3.2.4. Our proposal

We propose to generate hypervolume approximations by averaging a cumulative-sum of function values. The overall process is depicted in Fig 2. First, we evaluate our training data (described in the next section) using the new approximation function. Then, we obtain the mean squared error using the new approximation and the real hypervolume and assign the individual's fitness accordingly.

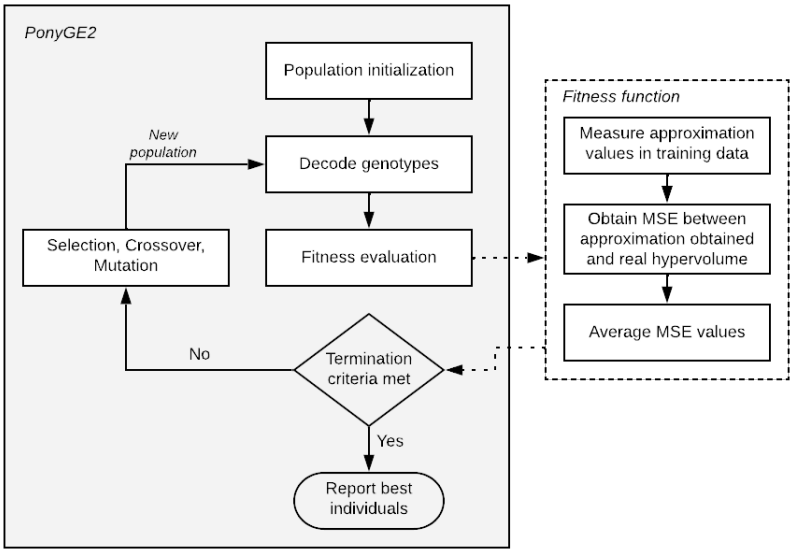


Figure 2: Diagram of grammatical evolution used to generate hypervolume approximations.

In Algorithm 2 we present the outline of the corresponding fitness function. In order to evaluate the fitness of each individual we require its decoded phenotype, a training set $\vec{X} = \{X_1, \ldots, X_n\}$, where each X_i contains a set of non-dominated points, and the corresponding set of real hypervolume values $\vec{I_H} = \{I_1, \ldots, I_n\}$, where I_i is the real hypervolume of the file X_i. Then, we iterate through each file X_i to evaluate the phenotype using vector-wise information. This means that for a given file of size n we will evaluate the phenotype n times, storing the result in a cumulative-sum variable. Then, we average this value with respect to n. This process is repeated with every training set, and we use the average mean squared error (MSE) of each approximation to assign the final fitness.

Algorithm 2: Fitness function used to generate hypervolume approximations using our proposal

Input : phenotype, $\vec{X} = \{X_1, \ldots, X_n\}$, $\vec{I_H} = \{I_1, \ldots, I_n\}$;
Output: fitness;

```
1  mse ← 0;
2  for i ← 0 to n do
3  |   approximation ← 0;
4  |   foreach x⃗ ∈ X_i do
5  |   |   approximation ← approximation+evaluate(phenotype,x⃗);
6  |   end
7  |   approximation ← approximation/size(X_i);
8  |   mse ← mse + (I_i − approximation)^2;
9  end
10 fitness ← mse/size(X⃗);
11 return fitness;
```

We adopted two different grammars to generate hypervolume approximations using our averaging variant. The first one is shown below, where $\vec{x} \in \mathbb{R}^m$ corresponds to the m-dimensional non-dominated points in a given training set X of size n, and $\vec{sum} \in \mathbb{R}^m$ stores the sum of all objectives of $\vec{x}$ considering $sum_j = \sum_{i=1}^{n} x_{i,j}$. In contrast with the grammar adopted to generate scalarizing functions, we decided to include sin, cos, log and exp in addition of the basic arithmetic operators.

$$
\begin{aligned}
\langle e \rangle \quad &::= \langle e \rangle + \langle e \rangle \mid \langle e \rangle \text{ - } \langle e \rangle \mid \langle e \rangle * \langle e \rangle \mid \langle e \rangle / \langle e \rangle \\
&\mid \sin(\langle e \rangle) \mid \cos(\langle e \rangle) \mid \log(\langle e \rangle) \mid \exp(\langle e \rangle) \\
&\mid \vec{x} \mid \vec{sum} \mid n \\
&\mid \langle c \rangle \langle c \rangle . \langle c \rangle \langle c \rangle
\end{aligned}
$$

$$
\langle c \rangle \quad ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9
$$

The second grammar adopted is similar to the previous one, but with the addition of the statistical features used in [30]. In a similar way to $\vec{sum}$, all statistical features are calculated component-wise.

$$
\begin{aligned}
\langle e \rangle \quad &::= \langle e \rangle + \langle e \rangle \mid \langle e \rangle \text{ - } \langle e \rangle \mid \langle e \rangle * \langle e \rangle \mid \langle e \rangle / \langle e \rangle \\
&\mid \sin(\langle e \rangle) \mid \cos(\langle e \rangle) \mid \log(\langle e \rangle) \mid \exp(\langle e \rangle) \\
&\mid \vec{x} \mid \vec{sum} \mid n \\
&\mid \vec{\mu} \text{ (mean)} \mid \vec{\sigma} \text{ (standard deviation)} \\
&\mid \vec{Q1} \text{ (1st quartile)} \mid \vec{Q2} \text{ (2nd quartile)} \mid \vec{Q3} \text{ (3rd quartile)} \\
&\mid \vec{\kappa} \text{ (kurtosis)} \mid \vec{\lambda} \text{ (skewness)} \\
&\mid \langle c \rangle \langle c \rangle . \langle c \rangle \langle c \rangle
\end{aligned}
$$

$$
\langle c \rangle \quad ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9 \quad .
$$

4. Experimental work

In this section we describe all the experimental setup used to generate two new scalarizing functions with our proposed methodology. We also describe the process followed to generate the training data used to generate eight different hypervolume approximations.

4.1. Scalarizing functions

Using the implementation previously described, we performed a series of executions adopting the benchmark problems from the Deb-Thiele-Laumanns-Zitzler (DTLZ) [37] and Walking-Fish-Group (WFG) [38] test suites.

We chose one test problem per execution. However, we solved the problem with 2, 3 and 5 objectives, in an attempt to create scalarizing functions which could have a good performance in different dimensions. We used DTLZ1-DTLZ7 and WFG1-WFG9. In Tables 2 to 17 we show the parameters that

we adopted for each execution. As can be seen, the sample MOP is exactly the same as the one adopted to train. However, in our experiments, we only performed 3 MOMBI-II executions. Then, if the function is worth more evaluations we proceed to execute MOMBI-II for each of the training MOPs for a total of 30 times.

Table 2: Parameters adopted to generate scalarizing functions using DTLZ1.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ1	2	2.434	[1.6,1.6]	30
		3	3.322	[1.5,1.5,1.5]	30
		5	7.565	[1.5,1.5,1.5,1.5,1.5]	30
Sample MOP		2	2.433	[1.6,1.6]	3

Table 3: Parameters adopted to generate scalarizing functions using DTLZ2.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ2	2	3.21	[2,2]	30
		3	7.353	[2,2,2]	30
		5	31.598	[2,2,2,2,2]	30
Sample MOP		2	3.21	[2,2]	3

Table 4: Parameters adopted to generate scalarizing functions using DTLZ3.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ3	2	3.619	[2.1,2.1]	30
		3	8.637	[2.1,2.1,2.1]	30
		5	40.484	[2.1,2.1,2.1,2.1,2.1]	30
Sample MOP		2	2.111	[2.1,2.1]	3

Additionally, for each execution we set the maximum number of generations to 40, and the threshold to 0.15. To validate the performance of the

Table 5: Parameters adopted to generate scalarizing functions using DTLZ4.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ4	2	3.21	[2,2]	30
		3	7.37	[2,2,2]	30
		5	31.686	[2,2,2,2,2]	30
Sample MOP		2	3.21	[2,2]	3

Table 6: Parameters adopted to generate scalarizing functions using DTLZ5.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ5	2	3.21	[2,2]	30
		3	4.731	[1.8,1.8,2]	30
		5	104.98	[1.8,1.8,4.4,4.5,2]	30
Sample MOP		2	3.21	[2,2]	3

Table 7: Parameters adopted to generate scalarizing functions using DTLZ6.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ6	2	5.26	[2.5,2.5]	30
		3	6.522	[2,2,2.2]	30
		5	2997.455	[4,3.5,8.5,9.7,2.8]	30
Sample MOP		2	4.662	[2.5,2.5]	3

Table 8: Parameters adopted to generate scalarizing functions using DTLZ7.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ7	2	4.148	[1.9,5]	30
		3	11.878	[1.9,1.9,7]	30
		5	86.182	[1.9,1.9,2,2,11.8]	30
Sample MOP		2	4.146	[1.9,5]	3

Table 9: Parameters adopted to generate scalarizing functions using WFG1.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG1	2	5.308	[3.7,4.1]	30
		3	71.692	[3.5,5.6,6.5]	30
		5	81.055	[3.6,2.2,2.7,2.9,5.6]	30
Sample MOP		2	4.724	[3.7,4.1]	3

Table 10: Parameters adopted to generate scalarizing functions using WFG2.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG2	2	7.444	[2.4,5.1]	30
		3	80.645	[2.8,4.6,6.9]	30
		5	3492.889	[2.5,4,5.5,7.2,9.3]	30
Sample MOP		2	7.381	[2.4,5.1]	3

Table 11: Parameters adopted to generate scalarizing functions using WFG3.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG3	2	11.311	[3.1,5.1]	30
		3	59.045	[4,3.1,7.1]	30
		5	10353.854	[3.9,5.3,7.6,8.6,11.1]	30
Sample MOP		2	10.743	[3.1,5.1]	3

Table 12: Parameters adopted to generate scalarizing functions using WFG4.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG4	2	8.897	[3.1,5.1]	30
		3	83.189	[3.1,5.1,7.1]	30
		5	9901.827	[3.1,5.1,7.1,9.1,11.1]	30
Sample MOP		2	8.031	[3.1,5.1]	3

Table 13: Parameters adopted to generate scalarizing functions using WFG5.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG5	2	8.668	[3.1,5.1]	30
		3	79.515	[3.1,5.1,7.1]	30
		5	9437.534	[3.1,5.1,7.1,9.1,11.1]	30
Sample MOP		2	8.044	[3.1,5.1]	3

Table 14: Parameters adopted to generate scalarizing functions using WFG6.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG6	2	8.827	[3.1,5.1]	30
		3	80.543	[3.1,5.1,7.1]	30
		5	9506.954	[3.1,5.1,7.1,9.1,11.1]	30
Sample MOP		2	8.297	[3.1,5.1]	3

Table 15: Parameters adopted to generate scalarizing functions using WFG7.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG7	2	7.836	[3.1,5.3]	30
		3	83.5	[3.1,5.1,7.1]	30
		5	9989.461	[3.1,5.1,7.1,9.1,11.1]	30
Sample MOP		2	7.303	[3.1,5.3]	3

Table 16: Parameters adopted to generate scalarizing functions using WFG8.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG8	2	8.89	[3.3,5.2]	30
		3	83.425	[3.3,5.1,7.1]	30
		5	11786.065	[4,5.5,7.1,9.1,11.1]	30
Sample MOP		2	7.702	[3.3,5.2]	3

Table 17: Parameters adopted to generate scalarizing functions using WFG9.

	Problem	m	Maximum Hypervolume	Reference point	n
Training MOPs	WFG9	2	8.945	[3.1,5.1]	30
		3	79.929	[3.2,5.1,7.1]	30
		5	9195.075	[3.2,5.2,7.2,9.2,11.2]	30
Sample MOP		2	8.608	[3.1,5.1]	3

best scalarizing functions obtained in each execution, we used each of them to solve all 16 test problems (DTLZ1-DTLZ7 and WFG1-WFG9) considering 2, 3, 4, 5, 6 and 7 objectives. This generates a total of 96 test problems. We performed 30 independent runs and computed the hypervolume and the s-energy values of each Pareto front. S-energy is a performance indicator used to measure the uniformity of the distribution of a set of points. The lower the s-energy value the more uniform the distribution is [39]. Then, we used the Wilcoxon rank-sum test under the null hypothesis that the indicator results generated with each new scalarizing function come from the same distribution as the indicator results generated with ASF, considering a confidence level of 5%. We counted the number of problems where the null hypothesis is rejected and the new scalarizing function indicator average is greater than that of ASF under the column "+" of each comparison. Conversely, if the null hypothesis is rejected but the ASF indicator average is greater, we count these under the column "-". Finally, all problems in which the null hypothesis could not be rejected are counted in the column "$\sim$". We show these results in Table 18, including the execution time required to complete the 40 generations. These executions were performed on an Intel Core i7-8700 CPU, with 16 GB of RAM.

From these initial results, we can appreciate that many scalarizing functions (especifically those generated using DTLZ3, DTLZ6, WFG4-WFG9) include the $f_i(\vec{x})/w_i$ term from the original ASF. In fact, the scalarizing function generated with DTLZ3 was indeed, ASF. The best performing function is the one generated with DTLZ6, as it outperforms ASF in more problems (44) and it worsens them in 23 problems with respect to the hypervolume. A similar behavior is observed with respect to the s-energy values. However, we considered that it should be possible to obtain a better scalarizing function using more test problems and more generations.

Table 18: Behavior of scalarizing functions generated with Grammatical Evolution. The comparison of the hypervolume and S-energy values show the number of test problems (a total of 96) in which the results improved(+), worsened(-) or were similar($\sim$) with respect to ASF.

Training problem	$SF(f_i(\vec{x}), w_i)$	Execution time (s)	Hypervolume comparison			S-energy comparison		
			+	-	$\sim$	+	-	$\sim$
DTLZ1	$f_i(\vec{x}) * w_i$	4455.67	17	59	20	21	48	27
DTLZ2	$w_i/w_i/w_i + w_i - \sqrt{w_i} + \sqrt{f_i(\vec{x})}$	3997.07	10	82	4	3	83	10
DTLZ3	$f_i(\vec{x})/w_i$	3949.98	0	0	96	0	0	96
DTLZ4	$\sqrt{w_i} * f_i(\vec{x})$	3403.82	9	81	6	34	49	13
DTLZ5	$\sqrt{f_i(\vec{x})} + w_i$	5831.65	15	72	9	31	53	12
DTLZ6	$f_i(\vec{x})/w_i + f_i(\vec{x})$	4667.40	44	23	29	37	25	34
DTLZ7	$f_i(\vec{x}) * w_i$	4357.98	17	59	20	21	48	27
WFG1	$\sqrt{w_i} * 84.75 + f_i(\vec{x}) * f_i(\vec{x})$	8338.57	3	92	1	1	91	4
WFG2	$\sqrt{f_i(\vec{x}) * w_i * w_i}$	11567.13	19	72	5	10	81	5
WFG3	$w_i + 01.52 + \sqrt{f_i(\vec{x})}$	11135.62	15	72	9	30	50	13
WFG4	$f_i(\vec{x})/w_i - w_i - w_i$	12056.09	19	71	6	4	74	18
WFG5	$f_i(\vec{x})/w_i - w_i - w_i$	9838.63	19	71	6	4	74	18
WFG6	$f_i(\vec{x})/w_i - w_i * w_i$	11726.39	13	58	25	14	48	34
WFG7	$f_i(\vec{x})/w_i - f_i(\vec{x})$	9361.49	15	73	8	8	63	25
WFG8	$f_i(\vec{x})/w_i - w_i - w_i$	11093.16	19	71	6	4	74	18
WFG9	$f_i(\vec{x})/w_i - w_i/f_i(\vec{x})$	10509.27	11	7	78	11	13	72

Hence, we performed another round of experiments. This time we used two different MOPs per execution. In Table 19 we present the parameters of the problems used to generate another scalarizing function. The threshold parameter was set to 0.15 and we set the maximum number of generations to 80.

Table 19: Parameters adopted to generate a scalarizing function using DTLZ4 and WFG4.

	Problem	Objectives	Maximum Hypervolume	Reference point	n
Training MOPs	DTLZ4	2	0.210	[1,1]	30
	DTLZ4	3	0.420	[1,1,1]	30
	DTLZ4	5	0.7	[1,1,1,1,1]	30
	WFG4	2	2.100	[2.1,4.1]	30
	WFG4	3	21.500	[2.1,4.1,6.1]	30
	WFG4	5	2035	[2.1,4.1,6.1,8.1,10.1]	30
Sample MOP	DTLZ4	2	0.210	[1,1]	3

We performed three independent executions using these parameters, and here we present the best performing individual, which we call GE_SF1, and it is defined as follows.

$$GE\_SF1(\vec{f}(\vec{x}), \vec{w}) := \max_{i \in 1,\ldots,m} \left(\frac{f_i(\vec{x})}{w_i} * f_i(\vec{x}) + \frac{f_i(\vec{x})}{w_i} - f_i(\vec{x}) \right). \tag{15}$$

Additionally, we performed three independent executions using I-DTLZ4

as the training problem, which is a variant belonging to the inverse DTLZ test problems [40]. The number of objectives used was 2, with a maximum hypervolume of 18.0, a reference point [1,1] and n set to 30. The resulting function, GE_SF2, is defined as follows.

$$GE\_SF2(\vec{f}(\vec{x}), \vec{w}) := \max_{i \in 1,\ldots,m} \left(f_i(\vec{x}) + w_i + 1 - \frac{f_i(\vec{x})}{22.42} \right). \tag{16}$$

4.2. Hypervolume approximations

In order to generate our training/validation data, we used some of the different geometries provided by problems from the DTLZ and from the WFG test suites. In Table 20, we show the problems adopted to obtain sets with different geometries. There is a difference in the number of problems since some problems (such as DTLZ5 and WFG3) are degenerate with 3 or more objectives, changing the shape that they present in their 2-objectives versions.

Table 20: Test problems used in the generation of training/validation sets grouped by their geometry.

(a) Problems used to generate 2-dimensional training sets

Geometry	Test problems
Linear	DTLZ1
Concave	DTLZ2
Mixed	DTLZ7, WFG1, WFG2

(b) Problems used to generate 3, 4 and 5 dimensional training sets

Geometry	Test problems
Linear	DTLZ1, WFG3
Convex	WFG2
Concave	DTLZ2, DTLZ5
Mixed	DTLZ7, WFG1

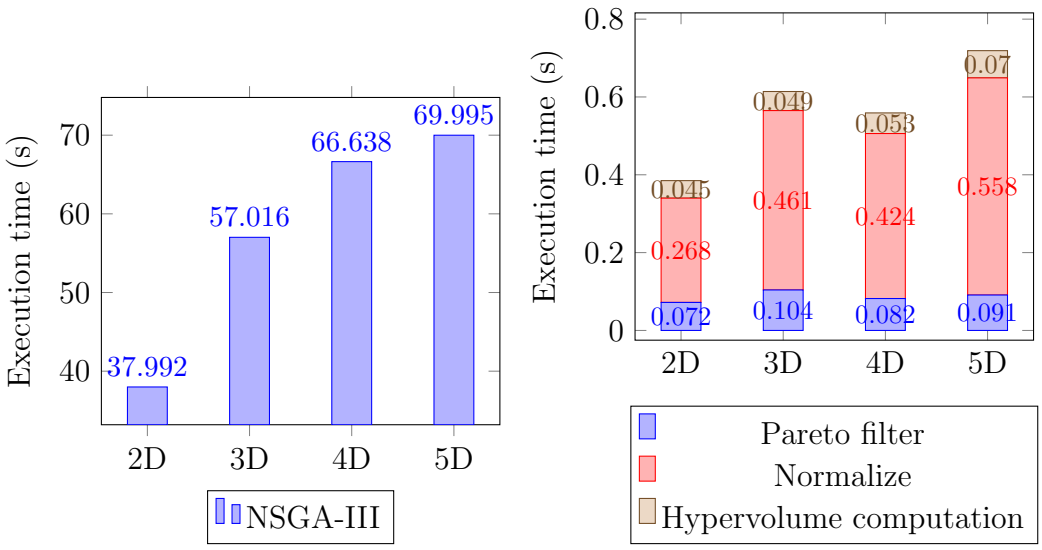
We used NSGA-III [41] to solve each of the selected problems and stored the population at every 100 generations. For problems with 2 objectives we set the population size to be 100 individuals, whereas the problems with 3 and 4 objectives were set to 120 individuals and the problems with 5 objectives were set to 140 individuals. Then, we filtered each of the resulting files to delete all dominated solutions present in the data. This changed the size of elements in each file, since not all solutions are non-dominated in each generation. Next, we normalized the remaining solutions to the interval [0,1]. This is done to avoid the definition of a reference point in the hypervolume approximation function's grammar. Once these files were obtained, we randomly separated them into the training and the validation set. The

total number of data files generated with each problem, as well as the size of the corresponding training/validation sets are shown in Table 21. The final training/validation sets were created by combining all resulting files for each problem in each dimension, creating 4 different training/validation sets.

Table 21: Number of files generated using each of the selected test problems.

Objectives	Problem	Total files generated	Training set size	Validation set size
2	DTLZ1	996	491	505
	DTLZ2	891	432	459
	DTLZ7	894	433	461
	WFG1	750	383	367
	WFG2	747	363	384
3	DTLZ1	832	438	394
	DTLZ2	868	401	467
	DTLZ5	832	416	416
	DTLZ7	832	431	401
	WFG1	750	388	362
	WFG2	832	400	432
	WFG3	832	398	434
4	DTLZ1	873	408	465
	DTLZ2	868	435	433
	DTLZ5	832	404	428
	DTLZ7	832	411	421
	WFG1	750	382	368
	WFG2	832	433	399
	WFG3	832	421	411
5	DTLZ1	831	406	425
	DTLZ2	833	406	427
	DTLZ5	792	401	391
	DTLZ7	792	392	400
	WFG1	752	362	390
	WFG2	792	406	386
	WFG3	792	375	417

In Fig. 3 we show the computational time used in each step of the process to create the training/validation data. In the first step we used the PlatEMO [42] implementation of NSGA-III to store the populations. Then, we used Python scripts to Pareto filter such files and to normalize them. Fi-



(a) Total computational time used to generate the populations files using NSGA-III.

(b) Total computational time used to filter, to normalize and to obtain the real hypervolume of the population files.

Figure 3: Computational times used to generate the training/validation sets used for hypervolume approximations.

nally, we employed a C implementation [43] to obtain the real hypervolume values of each file. All these steps were performed on an Intel Core i7-8700 CPU, with 16 GB of RAM.

We performed 8 different executions, considering the two grammars previously described for each of the four training sets generated. The maximum number of generations was set to 300, whereas the population size was set to 20. Here, we present the best result obtained from each of these executions.

In eqns (17), (18), (19) and (20) we present the best hypervolume approximation functions found using the first grammar for data with 2, 3, 4 and 5 objectives, respectively.

$$
\begin{aligned}
GEHV1_{2D}(X) =& \frac{1}{n}\sum_{i=1}^{n}\Bigg(\cos(\vec{x}_{i,1}/\cos(\vec{x}_{i,2}+\sin(34.56)) \\
&+\sin(\sin(\vec{x}_{i,1}/\cos(\sin(\vec{x}_{i,2})) \\
&+\vec{x}_{i,2}*\cos(\sin(\sin(\vec{x}_{i,1}*\sin(n-\vec{x}_{i,2}*\cos(\sin(\sqrt{\log(n)})) \\
&*\sqrt{\vec{x}_{i,2}}/\cos(\sin(\sin(\sin(\sin(\vec{x}_{i,1}*\sin(\vec{x}_{i,2})))) \\
&*\cos(\sin(\cos(\sin(\sin(\sin(\sin(\cos(\cos(\vec{x}_{i,1}*\sin(\vec{x}_{i,2})))) \\
&*\vec{x}_{i,2}/\cos(\vec{x}_{i,2})))*\cos(\sin(\vec{x}_{i,1}*\sin(\vec{x}_{i,2})*\vec{x}_{i,2}/\sin(\exp(\vec{x}_{i,2})) \\
&*\vec{x}_{i,2}*\vec{x}_{i,2})*\vec{x}_{i,2})))))*\vec{x}_{i,2}*\cos(\sin(\sin(\sin(\vec{x}_{i,1}*\sin(\vec{x}_{i,2}) \\
&*\cos(\sin(\sin(\sin(\cos(07.07+\vec{x}_{i,1})*s\vec{u}m_1*\sin(\vec{x}_{i,1} \\
&*\sin(\vec{x}_{i,2})*\vec{x}_{i,2}/\cos(\vec{x}_{i,2})))))))*\cos(\exp(\vec{x}_{i,1})*n))*\vec{x}_{i,2}) \\
&*\cos(n)))))*\sin(\sin(n*\sin(18.34))*\cos(\sin(\vec{x}_{i,2})) \\
&+\vec{x}_{i,2}))*\vec{x}_{i,2})*\sin(\sin(\vec{x}_{i,1}*\sin(18.71)) \\
&*\log(\sin(\exp(\sin(34.51)))))))*\sin(\sqrt{\vec{x}_{i,2}})))\Bigg)
\end{aligned}
\tag{17}
$$

$$
\begin{aligned}
GEHV1_{3D}(X) =& \frac{1}{n}\sum_{i=1}^{n}\Bigg(\cos(\vec{x}_{i,1}+\sin(\cos(\vec{x}_{i,3}))*\exp(\vec{x}_{i,3}/\sqrt{\cos(\vec{x}_{i,3})})) \\
&+\cos(\sin(\vec{x}_{i,2}*\sin(\exp(\vec{x}_{i,2}))+\cos(\sin(\cos(\cos(\vec{x}_{i,3}) \\
&*\cos(\sin(\cos(\log(\cos(\cos(n+\vec{x}_{i,3}))))*\vec{x}_{i,3} \\
&+\vec{x}_{i,3}/\cos(\sin(\cos(s\vec{u}m_1))))/\cos(\sin(n \\
&*\sin(\cos(\cos(\vec{x}_{i,3}+\cos(\sin(s\vec{u}m_2)-\vec{x}_{i,2} \\
&-\cos(\vec{x}_{i,1}+\sin(\cos(\vec{x}_{i,3}))+s\vec{u}m_2))))) \\
&+\cos(n+\vec{x}_{i,3})-\vec{x}_{i,3})))))))\Bigg)
\end{aligned}
\tag{18}
$$

$$
GEHV1_{4D}(X) = \frac{1}{n}\sum_{i=1}^{n}\cos(\vec{x}_{i,1}+\vec{x}_{i,2}*\vec{x}_{i,3}+\vec{x}_{i,4}) \tag{19}
$$

$$
\begin{aligned}
GEHV1_{5D}(X) =& \frac{1}{n}\sum_{i=1}^{n}\Bigg(\cos(\vec{x}_{i,1}+\sin(\vec{x}_{i,5})+\vec{x}_{i,3}*\sin(\vec{x}_{i,1}+\sin(\vec{x}_{i,1}) \\
&-\sin(\sin(\vec{sum}_5/98.27*\vec{x}_{i,4}*\vec{x}_{i,3} \\
&+\sin(\sin(\vec{x}_{i,5}-\vec{x}_{i,3}+\sin(\sin(98.22)) \\
&-\sin(\vec{sum}_2-\vec{sum}_3+\vec{x}_{i,1})-\sin(\vec{x}_{i,5}*\sin(\vec{x}_{i,5})+\vec{x}_{i,3}+n) \\
&-\vec{x}_{i,4}+\vec{x}_{i,1}-\sin(\vec{x}_{i,5})+\vec{x}_{i,3}*\sin(\vec{sum}_1+68.66))-\sin(\vec{x}_{i,2}) \\
&+\vec{x}_{i,5}*\vec{x}_{i,5}-\vec{x}_{i,3}-\vec{x}_{i,1}+\sin(\exp(\sin(\vec{sum}_1 \\
&*\sin(\vec{sum}_1+\sin(\vec{x}_{i,2})-\vec{x}_{i,5}*\sin(\vec{x}_{i,3}+\vec{x}_{i,2}*\vec{x}_{i,5}+\vec{sum}_1 \\
&+\sqrt{\vec{x}_{i,5}}*\vec{x}_{i,3}*\sin(\sin(\vec{x}_{i,5})))))))+\vec{x}_{i,3}*\sin(\vec{x}_{i,1}/n*\sin(\vec{x}_{i,2}) \\
&*\vec{x}_{i,5}+\sin(\sqrt{\vec{sum}_4}-98.06+\vec{x}_{i,3}*\sqrt{\vec{x}_{i,1}+\vec{sum}_3-\vec{x}_{i,2}*\vec{x}_{i,5}} \\
&*\sin(\cos(98.67)+\vec{x}_{i,5}+96.65*97.82*\sin(\exp(\vec{x}_{i,1})*\vec{x}_{i,3} \\
&-98.62*\sin(\vec{x}_{i,1}*\vec{x}_{i,1})+\exp(\sin(\vec{sum}_1+\vec{x}_{i,2}*\vec{x}_{i,5}+\vec{sum}_1 \\
&+29.20*\sqrt{\vec{x}_{i,5}}))-\vec{x}_{i,3})+\exp(n))*\sin(\vec{x}_{i,3}*\exp(\vec{sum}_3) \\
&*\vec{x}_{i,4}+\vec{x}_{i,1}+\vec{sum}_2*\sin(\vec{x}_{i,3})*\vec{x}_{i,3})*\sin(\vec{sum}_1*\vec{x}_{i,1})))))) \\
&*\sin(\vec{x}_{i,1})-\sin(\sin(\sqrt{\vec{sum}_5+\vec{sum}_3}+\sqrt{\sqrt{\vec{x}_{i,3}}}))))\Bigg)\ .
\end{aligned}
\tag{20}
$$

In eqns (21), (22), (23) and (24) we present the best hypervolume approximation functions found using the second grammar for data with 2, 3, 4 and 5 objectives, respectively.

$$
GEHV2_{2D}(X) = \frac{1}{n}\sum_{i=1}^{n}\sin(\cos(\vec{Q2}_2*\cos(\vec{\sigma}_1)+\vec{Q2}_1)) \tag{21}
$$

$$
\begin{aligned}
GEHV2_{3D}(X) =& \frac{1}{n}\sum_{i=1}^{n}\Bigg(\cos(\vec{\mu}_1 + \vec{\mu}_3 + \sin(\sin(\vec{Q1}_2 * \sin(\sin(\vec{Q2}_1 * \vec{\lambda}_1 * \vec{x}_{i,3} \\
&+ \sin(\sqrt{\sin(\vec{x}_{i,2})}) * \vec{\mu}_3)) + \exp(\vec{\sigma}_2) * \vec{Q1}_2 * \vec{\sigma}_2 \\
&* (\vec{x}_{i,2} + \vec{x}_{i,1} + \sin(\vec{\mu}_1 + \vec{\mu}_3 + \sin(\sin(\vec{Q1}_2 \\
&* \sin(\sin(\vec{x}_{i,3} * \vec{\lambda}_1 * \sin(\sqrt{\sqrt{\vec{x}_{i,2}}}) * \vec{Q3}_3 + \vec{Q1}_2 \\
&* \sin(\cos(\sin(\vec{x}_{i,2} * \vec{x}_{i,1} * \sin(\sin(\sin(\vec{Q3}_3 + \vec{Q1}_2)) \\
&* \sin(\cos(\vec{\mu}_1 + \vec{\mu}_3 * \vec{\mu}_1 * \vec{\mu}_3 - \sin(\sin(\vec{Q1}_2 \\
&* \sin(\sin(s\vec{um}_3 * \vec{\sigma}_1)) + s\vec{um}_2 * \vec{\lambda}_1 * \vec{x}_{i,3} \\
&* \sin(\sqrt{\sqrt{\vec{x}_{i,2}}})))))) + \vec{\mu}_3 * \exp(\vec{\sigma}_2) - \vec{Q1}_2 * \vec{\sigma}_2)))) \\
&+ \sqrt{\vec{x}_{i,2} + \sqrt{\vec{x}_{i,2}} + \vec{x}_{i,1}} * \sin(\vec{\mu}_1 + \vec{\mu}_1 + \vec{\mu}_3 \\
&+ \sin(\sin(\vec{\mu}_1 * \vec{\mu}_3 * \vec{Q1}_3 * \vec{Q1}_3)) + \vec{Q1}_2 \\
&* \sin(\sin(\vec{Q2}_1 * \vec{\lambda}_1 * \cos(\sin(\sqrt{\sin(\vec{x}_{i,2})}))) + \vec{\mu}_3 \\
&* \exp(\vec{\sigma}_2))) + \vec{Q1}_2 * \vec{\sigma}_2 \\
&* (\vec{x}_{i,2} + \vec{x}_{i,1} + \sin(\sin(\sin(\vec{Q3}_3 + \vec{\sigma}_1)) \\
&* \sin(\cos(\vec{x}_{i,3} * \sin(\vec{\kappa}_2) - \vec{\sigma}_2^2))) + \vec{x}_{i,2} + \sqrt{\vec{x}_{i,2}})^{1/2}) \\
&* \log(\vec{\mu}_1 + \vec{\mu}_3))) + \vec{Q2}_1))^{1/2}))))\Bigg)
\end{aligned}
\tag{22}
$$

$$
\begin{aligned}
GEHV2_{4D}(X) =& \frac{1}{n}\sum_{i=1}^{n}\Bigg(\cos(\vec{x}_{i,1} + \vec{Q1}_3 + \vec{Q2}_4 + \exp(\vec{Q3}_4) * \vec{\sigma}_2 * \vec{\sigma}_4 \\
& * \cos(\vec{x}_{i,1} * \vec{Q3}_4 * \vec{\sigma}_2 * \vec{\sigma}_4 * \cos(\vec{Q3}_4 * \vec{\sigma}_4) \\
& * \exp(\vec{\kappa}_3) * \vec{x}_{i,3} * \vec{\sigma}_4 * \cos(\vec{sum}_4) * \vec{\sigma}_4 * \cos(\vec{Q1}_4) \\
& * \vec{x}_{i,1} + \vec{Q1}_3 + \vec{Q2}_4 - \vec{\lambda}_2 * \vec{\sigma}_2 * \vec{\lambda}_4 + \vec{Q1}_3 \\
& + \vec{\sigma}_3^2 / \exp(\vec{Q3}_4) * \vec{Q1}_1 * \vec{\sigma}_4 * \vec{Q3}_1 - \vec{\lambda}_2 * \vec{\sigma}_2 \\
& * \vec{\lambda}_4 + \vec{Q1}_3 + \vec{Q2}_1 * \exp(\vec{Q3}_4) * \vec{x}_{i,1} * \vec{\sigma}_4 * \vec{\sigma}_2 * \vec{x}_{i,1} \\
& * \vec{Q3}_1 - \vec{\lambda}_2 * \vec{\sigma}_2 * \vec{\kappa}_2 + \vec{Q1}_3 + \vec{\sigma}_3^2 * \vec{x}_{i,4} \\
& * \exp(\vec{Q3}_1) * \vec{\lambda}_2 * \vec{Q3}_4 * \vec{\kappa}_2 + \vec{Q1}_3 + \vec{\sigma}_2 * \exp(\vec{Q3}_1) \\
& * \vec{\sigma}_2 * \vec{\sigma}_4 * \vec{\sigma}_4 * \cos(\vec{\mu}_4 * \vec{\sigma}_2 * \vec{\sigma}_2 * \vec{\sigma}_4 \\
& * \cos(\vec{x}_{i,1} * \vec{\sigma}_4^2 * \vec{\sigma}_4^2 * \vec{x}_{i,3} * \vec{x}_{i,2} * \cos(\vec{Q1}_4) \\
& * \vec{sum}_3 + \exp(\vec{x}_{i,2}) - \vec{Q1}_3 * \vec{Q2}_4 + \vec{\lambda}_2 * \vec{\sigma}_2) \\
& * \vec{Q1}_2 * \vec{\sigma}_2^2 + \sin(\vec{\kappa}_2 + \vec{Q1}_3 * \vec{\sigma}_3^2 * \vec{Q1}_3 * \vec{\sigma}_3^2 * \vec{x}_{i,4} \\
& - \exp(\vec{Q3}_1) * \vec{\lambda}_2 * \vec{\sigma}_2 * \vec{\sigma}_1 + \vec{\mu}_2 * \vec{\sigma}_2 * \vec{Q3}_3 * \vec{\sigma}_2 \\
& * \vec{\sigma}_4 * \cos(\vec{Q3}_1) * \vec{\lambda}_4 + \vec{Q1}_3 + \vec{\sigma}_3^2) * \exp(\vec{\sigma}_2) \\
& * \vec{Q2}_4 * \vec{\sigma}_2 * \vec{\sigma}_1 + \vec{\mu}_2 * \vec{\sigma}_2 * \vec{Q3}_3 * \vec{\sigma}_2 * \vec{\sigma}_4 \\
& * \cos(\vec{Q3}_1) * \vec{\lambda}_4 + \vec{Q1}_3 + \vec{\sigma}_3^2 * \exp(\vec{\sigma}_2) * \vec{Q2}_4 * \vec{\lambda}_2 \\
& * \vec{\lambda}_2 * \vec{\sigma}_2 * \vec{\kappa}_2 + \vec{Q1}_3 + \vec{\sigma}_2 * \vec{\sigma}_2 * \vec{\sigma}_4 * \cos(\vec{Q2}_4) \\
& + \vec{\lambda}_2 * \vec{\sigma}_2 * \vec{Q1}_2 * \vec{\sigma}_2^2 + \sin(\vec{sum}_3 * \exp(\vec{Q3}_1)) \\
& * \vec{\sigma}_2 * \vec{\sigma}_3^2 + \cos(\vec{Q3}_1)) * \vec{\lambda}_4 + \exp(\vec{Q1}_2) * \vec{\sigma}_3^2 * \vec{\sigma}_2 \\
& * \vec{\sigma}_2 * \vec{\sigma}_4 * \vec{\lambda}_4 * \vec{sum}_4))\Bigg)
\end{aligned} \tag{23}
$$

$$
GEHV2_{5D}(X) = \frac{1}{n}\sum_{i=1}^{n}\cos(\vec{Q3}_5 * \vec{x}_{i,5} + \vec{\mu}_4 + \vec{\mu}_1). \tag{24}
$$

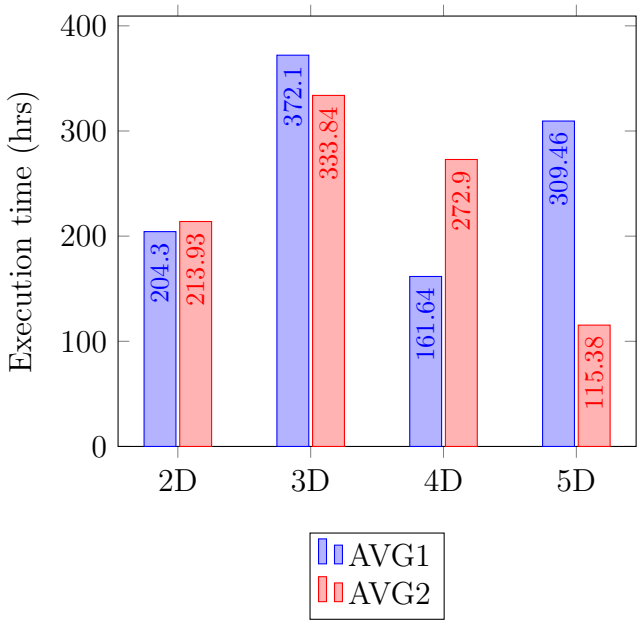


Figure 4: Computational time used to generate each of the hypervolume approximation functions using our proposal.

5. Results

5.1. Scalarizing functions

To assess the performance of GE_SF1 and GE_SF2 we coupled them with MOMBI-II and solved the benchmark problems DTLZ1-DTLZ7, WFG1-WFG9 and I-DTLZ1-I-DTLZ7 using 2, 3, 4, 5, 6 and 7 objectives. We performed 30 independent executions and measured the hypervolume values of the Pareto fronts generated. We repeated the same process with the original ASF, as well as with TCH and PBI.

In the following 3 tables, we show the best average hypervolume for each problem in **boldface**. Aditionally, we performed the Wilcoxon rank-sum test under the null hypothesis that the hypervolume results generated with one scalarizing function come from the same distribution as the hypervolume results generated with another scalarizing function, considering a confidence level of 5%. This test was performed with each pair of scalarizing functions, and we show the cases in which the null hypothesis was rejected and the current function's average hypervolume is greater under the "+" column of each function. Additionally, we indicate with gray cells all of the scalarizing

function results in which the null hypothesis could not be rejected when comparing against the best result (marked in **boldface**).

In Tables 22 and 23 we show the average results of the hypervolume indicator in standard benchmark problems (DTLZ and WFG).

On the other hand, in Table 24 we show the average results of the hypervolume indicator in inverse DTLZ problems.

From Tables 22 and 23 we counted the number of gray cells of each scalarizing function (i.e., the best performing functions for each problem) and grouped them by number of objectives. We show these results in Table 25. Here, we can observe that GE_SF1 is the best performing scalarizing function, having the best results in 56 out of 96 test instances, followed by PBI, which obtained the best results in 36 cases. This is the behavior we were looking for by combining DTLZ4 and WFG4, which is to improve the performance obtained with ASF. However, not only GE_SF1 outperformed ASF, but it also outperformed the other scalarizing functions adopted in this comparison. Conversely, GE_SF2 is the scalarizing function with the worst performance, as it only obtained the best results in 10 test instances, which was also an expected behavior since it was trained using I-DTLZ4. Nonetheless it is interesting to see that it was able to obtain top results in DTLZ5 with 2, 6 and 7 objectives as well as DTLZ6 with 2 and 3 objectives, even though it was not designed to successfully solve standard MOPs.

Even though GE_SF1 attained the largest number of problems improved, it is evident that not even this function is able to outperform the others in every test problem, since it only obtained top results in 58.33% of the test instances. This is also an expected behavior since we are dealing with many different types of problems/number of objectives/geometries, which makes it difficult for a single scalarizing function to improve results in every possible setup.

From Table 24 we counted the number of gray cells of each scalarizing function and we show these results in Table 26. These results correspond to the validation using inverse DTLZ problems. Here, we can observe that GE_SF2, which was generated specifically using an inverse test MOP in the search process of `PonyGE2`, obtains the largest number of problems improved. Overall, GE_SF2 is able to obtain top results in 36 out of 42 test instances.

Finally, we performed two additional comparisons against MOEA/D [34] and NSGA-III, as well as the default version of MOMBI-II with ASF. In the first comparison, shown in Table 27 we compare these three MOEAs against MOMBI-II with GE_SF1, since it was the best performing scalarizing

Table 22: Hypervolume comparison using different scalarizing functions in problems with 2-4 objectives.

Problem	m	ASF(1)		PBI(2)		TCH(3)		GE_SF1(4)		GE_SF2(5)	
		HV	+	HV	+	HV	+	HV	+	HV	+
DTLZ1	2	2.327E-01	–	**2.337E-01**	1,5	2.337E-01	1,5	2.337E-01	1,5	2.336E-01	1
DTLZ2	2	4.182E-01	–	4.201E-01	1,3	4.201E-01	1	4.203E-01	1,2,3	**4.205E-01**	1,3,2,4
DTLZ3	2	4.172E-01	–	4.174E-01	–	4.188E-01	1,2	4.188E-01	1,2	**4.191E-01**	1,2
DTLZ4	2	2.988E-01	5	3.101E-01	1,3,5	3.101E-01	1,5	**3.103E-01**	1,3,2,5	2.692E-01	–
DTLZ5	2	4.182E-01	–	4.201E-01	1,3	4.201E-01	1	4.203E-01	1,2,3	**4.205E-01**	1,3,2,4
DTLZ6	2	1.274E+00	–	1.268E+00	–	1.261E+00	–	**1.291E+00**	3	1.265E+00	–
DTLZ7	2	6.433E-01	–	6.445E-01	1	6.447E-01	1,2,5	**6.448E-01**	1,3,2,5	6.446E-01	1,2
WFG1	2	2.252E+00	2,3,5	1.519E+00	–	1.589E+00	2	**2.363E+00**	1,3,2,5	1.546E+00	–
WFG2	2	4.906E+00	2,3,5	4.300E+00	–	4.315E+00	2	**4.929E+00**	1,3,2,5	4.323E+00	2,3
WFG3	2	4.529E+00	2,3,5	4.470E+00	–	4.491E+00	2	**4.543E+00**	1,3,2,5	4.503E+00	2,3
WFG4	2	2.272E+00	2,3,5	2.223E+00	–	2.235E+00	2,5	**2.281E+00**	1,3,2,5	2.223E+00	–
WFG5	2	1.973E+00	2,3,5	1.963E+00	–	1.966E+00	2	**1.980E+00**	1,3,2,5	1.965E+00	2
WFG6	2	2.024E+00	–	2.047E+00	–	2.062E+00	–	2.035E+00	–	**2.071E+00**	1,4
WFG7	2	2.686E+00	2,3,5	2.350E+00	–	2.371E+00	2,5	**2.707E+00**	1,3,2,5	2.343E+00	–
WFG8	2	3.795E+00	2,5	3.424E+00	–	3.743E+00	2	**3.848E+00**	1,2,5	3.747E+00	2
WFG9	2	1.997E+00	–	2.210E+00	1,4	**2.221E+00**	1,2,4	1.917E+00	–	2.216E+00	1,4
DTLZ1	3	3.805E+01	5	**3.845E+01**	1,4,3,5	3.844E+01	1,4,5	3.805E+01	5	3.614E+01	–
DTLZ2	3	7.394E-01	3,5	**7.494E-01**	1,4,3,5	7.099E-01	5	7.410E-01	1,3,5	6.533E-01	–
DTLZ3	3	2.387E+02	–	**2.408E+02**	1,4,3,5	2.408E+02	1,4,5	2.387E+02	–	2.075E+02	–
DTLZ4	3	8.609E-01	3,5	**8.704E-01**	1,4,3,5	8.111E-01	5	8.628E-01	1,3,5	7.519E-01	–
DTLZ5	3	2.188E+01	4	2.207E+01	1,4	**2.212E+01**	1,4,2,5	2.182E+01	–	2.211E+01	1,2,4
DTLZ6	3	2.079E+02	4	2.087E+02	1,4	2.094E+02	1,2,4	2.074E+02	–	**2.097E+02**	1,3,2,4
DTLZ7	3	1.449E+00	2,3,5	1.409E+00	3	1.389E+00	–	**1.450E+00**	2,3,5	1.344E+00	–
WFG1	3	3.462E+01	2,5	3.265E+01	5	3.504E+01	2,5	**3.504E+01**	2,5	2.692E+01	–
WFG2	3	4.455E+01	2,3,5	4.440E+01	3,5	4.333E+01	5	**4.486E+01**	1,3,2,5	4.088E+01	–
WFG3	3	4.824E+01	2,5	4.785E+01	–	**4.850E+01**	1,4,2,5	4.819E+01	2,5	4.788E+01	–
WFG4	3	2.401E+01	2,3,5	2.312E+01	3,5	2.190E+01	5	**2.409E+01**	1,3,2,5	2.063E+01	–
WFG5	3	2.192E+01	2,3,5	2.172E+01	3,5	1.987E+01	5	**2.203E+01**	1,3,2,5	1.917E+01	–
WFG6	3	2.444E+01	2,3,5	2.352E+01	3,5	2.218E+01	5	**2.450E+01**	2,3,5	2.088E+01	–
WFG7	3	2.426E+01	2,3,5	2.359E+01	3,5	2.206E+01	5	**2.434E+01**	1,3,2,5	2.075E+01	–
WFG8	3	2.566E+01	2,3,5	2.547E+01	3,5	2.461E+01	5	**2.591E+01**	1,3,2,5	2.335E+01	–
WFG9	3	2.519E+01	2,3,5	2.391E+01	3,5	2.313E+01	–	**2.555E+01**	1,3,2,5	2.237E+01	–
DTLZ1	4	1.015E+01	5	**1.034E+01**	1,4,3,5	1.033E+01	1,4,5	1.016E+01	5	8.635E+00	–
DTLZ2	4	1.014E+00	3,5	**1.032E+00**	1,4,3,5	8.857E-01	5	1.016E+00	1,3,5	7.018E-01	–
DTLZ3	4	1.017E+00	3,5	**1.018E+00**	1,3,5	8.739E-01	5	1.018E+00	3,5	7.006E-01	–
DTLZ4	4	1.021E+00	3,4,5	**1.032E+00**	1,4,3,5	8.642E-01	5	1.018E+00	3,5	7.113E-01	–
DTLZ5	4	4.380E+01	5	**4.420E+01**	1,4,3,5	4.419E+01	1,4,5	4.382E+01	5	4.350E+01	–
DTLZ6	4	6.929E+03	2,5	6.890E+03	–	6.925E+03	2,5	**6.934E+03**	1,3,2,5	6.909E+03	2
DTLZ7	4	1.872E+00	2,3,5	1.820E+00	3,5	1.607E+00	–	**1.875E+00**	1,3,2,5	1.701E+00	3
WFG1	4	2.510E+02	2,5	2.419E+02	5	**2.753E+02**	1,4,2,5	2.504E+02	2,5	2.038E+02	–
WFG2	4	3.868E+02	2,3,5	3.695E+02	3,5	3.623E+02	–	**3.873E+02**	2,3,5	3.616E+02	–
WFG3	4	3.774E+02	5	**3.886E+02**	1,4,3,5	3.836E+02	1,4,5	3.788E+02	5	3.729E+02	–
WFG4	4	**2.511E+02**	2,4,3,5	2.421E+02	3,5	1.623E+02	–	2.499E+02	2,3,5	1.612E+02	–
WFG5	4	2.331E+02	2,3,5	2.301E+02	3,5	1.473E+02	–	**2.334E+02**	2,3,5	1.530E+02	3
WFG6	4	**2.341E+02**	2,3,5	2.289E+02	3,5	1.367E+02	–	2.337E+02	2,3,5	1.517E+02	3
WFG7	4	2.538E+02	2,3,5	2.516E+02	3,5	1.706E+02	5	**2.547E+02**	1,3,2,5	1.636E+02	–
WFG8	4	2.944E+02	3,5	**3.130E+02**	1,4,3,5	2.142E+02	–	3.106E+02	1,3,5	2.205E+02	3
WFG9	4	2.228E+02	3,5	2.273E+02	1,3,5	2.161E+02	–	**2.294E+02**	1,3,5	1.580E+02	3

Table 23: Hypervolume comparison using different scalarizing functions in problems with 5-7 objectives.

Problem	m	ASF(1)		PBI(2)		TCH(3)		GE_SF1(4)		GE_SF2(5)	
		HV	+	HV	+	HV	+	HV	+	HV	+
DTLZ1	5	7.580E-02	3,4,5	**7.673E-02**	1,4,3,5	7.467E-02	5	7.575E-02	3,5	6.270E-02	–
DTLZ2		1.291E+00	3,5	**1.308E+00**	1,4,3,5	1.146E+00	5	1.294E+00	1,3,5	7.531E-01	–
DTLZ3		1.129E+04	4,5	1.129E+04	1,4,5	**1.130E+04**	1,4,2,5	1.128E+04	5	8.178E+03	–
DTLZ4		1.304E+00	3,5	1.308E+00	1,3,5	1.146E+00	5	**1.310E+00**	1,3,2,5	7.864E-01	–
DTLZ5		1.198E+02	–	**1.285E+02**	1,4,3,5	1.238E+02	1,4	1.222E+02	1	1.229E+02	1,4
DTLZ6		9.618E+04	2,3,5	9.590E+04	3	9.510E+04	–	**9.626E+04**	1,3,2,5	9.593E+04	3
DTLZ7		2.985E+00	3,5	2.994E+00	1,3,5	2.588E+00	5	**3.006E+00**	1,3,2,5	1.886E+00	–
WFG1		9.512E+02	2,4,5	9.023E+02	5	**1.022E+03**	1,4,2,5	9.355E+02	2,5	8.513E+02	–
WFG2		**3.952E+03**	2,3,5	3.842E+03	5	3.899E+03	2,4,5	3.878E+03	2,5	3.801E+03	–
WFG3		5.134E+03	3	5.141E+03	3,5	4.812E+03	–	**5.196E+03**	1,3,2,5	5.094E+03	3
WFG4		2.980E+03	2,3,5	2.864E+03	3,5	1.955E+03	5	**2.989E+03**	1,3,2,5	1.593E+03	–
WFG5		2.767E+03	2,3,5	2.749E+03	3,5	1.534E+03	–	**2.793E+03**	1,3,2,5	1.485E+03	–
WFG6		2.890E+03	2,3,5	2.850E+03	3,5	1.069E+03	–	**2.908E+03**	2,3,5	1.532E+03	3
WFG7		3.085E+03	2,3,5	3.059E+03	3,5	2.076E+03	5	**3.094E+03**	1,3,2,5	1.631E+03	–
WFG8		4.742E+03	3,5	**5.218E+03**	1,4,3,5	3.059E+03	–	4.861E+03	1,3,5	3.092E+03	–
WFG9		2.690E+03	3,5	**2.815E+03**	1,3,5	1.192E+03	–	2.751E+03	1,3,5	1.667E+03	3
DTLZ1	6	4.607E-02	3,5	**4.643E-02**	1,4,3,5	4.597E-02	5	4.604E-02	5	3.836E-02	–
DTLZ2		1.536E+00	3,5	**1.549E+00**	1,4,3,5	1.432E+00	5	1.539E+00	1,3,5	8.118E-01	–
DTLZ3		7.854E+03	4,5	**7.857E+03**	1,4,3,5	7.857E+03	1,4,5	7.854E+03	5	7.301E+03	–
DTLZ4		1.548E+00	3,5	1.551E+00	1,3,5	1.438E+00	5	**1.555E+00**	1,3,2,5	8.471E-01	–
DTLZ5		4.951E+01	2,3	4.444E+01	–	4.806E+01	2	5.031E+01	1,2,3	**5.037E+01**	1,2,3
DTLZ6		3.316E+05	2,3,5	3.297E+05	3	3.150E+05	–	**3.318E+05**	2,3,5	3.295E+05	3
DTLZ7		3.173E+00	3,4,5	**3.206E+00**	1,4,3,5	2.736E+00	5	3.145E+00	3,5	1.915E+00	–
WFG1		2.555E+02	2,5	2.414E+02	–	**2.683E+02**	1,4,2,5	2.561E+02	2,5	2.403E+02	–
WFG2		4.426E+04	2	4.334E+04	–	**4.580E+04**	1,2,5	4.519E+04	1,2,5	4.458E+04	1,2
WFG3		7.153E+04	2,3,5	6.801E+04	3	6.512E+04	–	**7.230E+04**	1,3,2,5	6.924E+04	3
WFG4		3.845E+04	2,3,5	3.681E+04	3,5	2.644E+04	5	**3.929E+04**	1,3,2,5	1.886E+04	–
WFG5		3.849E+04	2,3,5	3.236E+04	3,5	2.207E+04	5	**3.910E+04**	1,3,2,5	1.893E+04	–
WFG6		4.297E+04	2,3,5	4.218E+04	3,5	1.429E+04	–	**4.324E+04**	1,3,2,5	2.127E+04	3
WFG7		4.006E+04	2,3,5	3.265E+04	5	2.835E+04	5	**4.036E+04**	1,3,2,5	1.903E+04	–
WFG8		6.954E+04	3,5	**7.983E+04**	1,4,3,5	3.340E+04	–	7.044E+04	1,3,5	4.440E+04	3
WFG9		3.328E+04	3,5	**3.763E+04**	1,4,3,5	1.251E+04	–	3.454E+04	1,3,5	2.141E+04	3
DTLZ1	7	3.240E-02	3,4,5	**3.258E-02**	1,4,3,5	3.176E-02	5	3.227E-02	3,5	2.622E-02	–
DTLZ2		1.761E+00	3,4,5	**1.773E+00**	1,4,3,5	1.366E+00	5	1.751E+00	3,5	8.475E-01	–
DTLZ3		1.744E+00	3,5	1.754E+00	1,3,5	1.339E+00	5	**1.755E+00**	1,3,5	9.017E-01	–
DTLZ4		1.773E+00	3,4,5	**1.774E+00**	1,4,3,5	1.469E+00	5	1.771E+00	3,5	9.113E-01	–
DTLZ5		8.148E+00	2,3	6.169E+00	–	7.842E+00	2	8.228E+00	2,3	**8.532E+00**	1,3,2,4
DTLZ6		6.208E+05	3,5	**6.302E+05**	1,3,5	4.979E+05	–	6.281E+05	1,3,5	6.166E+05	3
DTLZ7		**3.360E+00**	3,4,5	3.197E+00	3,5	2.468E+00	5	3.307E+00	2,3,5	2.446E+00	–
WFG1		1.392E+02	2	1.237E+02	–	**1.675E+02**	1,4,2,5	1.429E+02	1,2,5	1.391E+02	2
WFG2		6.303E+05	2,5	6.070E+05	–	**6.350E+05**	2,5	6.223E+05	2,5	6.176E+05	2
WFG3		1.105E+06	2,3	1.075E+06	3	9.999E+05	–	1.094E+06	2,3	**1.109E+06**	2,3
WFG4		5.364E+05	3,5	5.426E+05	1,3,5	3.237E+05	5	**5.506E+05**	1,3,5	2.694E+05	–
WFG5		5.268E+05	2,3,5	3.514E+05	3,5	2.125E+05	–	**5.402E+05**	1,3,2,5	2.420E+05	3
WFG6		**6.158E+05**	2,3,5	5.490E+05	3,5	3.520E+05	5	6.150E+05	3,5	2.901E+05	–
WFG7		5.822E+05	2,3,5	5.345E+05	3,5	3.862E+05	5	**5.904E+05**	1,3,5	2.800E+05	–
WFG8		9.453E+05	3,5	**1.196E+06**	1,4,3,5	2.015E+05	–	9.908E+05	1,3,5	5.819E+05	3
WFG9		4.745E+05	3,5	**6.323E+05**	1,4,3,5	3.626E+05	–	5.271E+05	1,3,5	3.747E+05	3

Table 24: Hypervolume comparison using different scalarizing functions in inverted problems.

Problem	m	ASF(1)		PBI(2)		TCH(3)		GE_SF1(4)		GE_SF2(5)	
		HV	+	HV	+	HV	+	HV	+	HV	+
DTLZ1_MINUS	2	**1.503E+05**	2,5	1.50E+05	5	**1.503E+05**	2,5	1.503E+05	2,5	1.50E+05	–
DTLZ2_MINUS	2	9.56E+00	2	9.53E+00	–	9.56E+00	2	9.56E+00	1,2,3	**9.572E+00**	1,3,2,4
DTLZ3_MINUS	2	3.79E+06	2	3.78E+06	–	3.79E+06	2	3.79E+06	1,2,3	**3.794E+06**	1,3,2,4
DTLZ4_MINUS	2	9.56E+00	2	9.53E+00	–	9.56E+00	2	9.56E+00	1,2,3	**9.572E+00**	1,3,2,4
DTLZ5_MINUS	2	9.56E+00	2	9.53E+00	–	9.56E+00	2	9.56E+00	1,2,3	**9.572E+00**	1,3,2,4
DTLZ6_MINUS	2	9.44E+01	2	9.42E+01	–	9.44E+01	2	9.45E+01	1,2,3	**9.455E+01**	1,3,2,4
DTLZ7_MINUS	2	6.55E-01	2	4.28E-01	–	6.55E-01	2	6.55E-01	2	**6.548E-01**	1,3,2,4
DTLZ1_MINUS	3	1.76E+07	2,3	1.74E+07	–	1.76E+07	2	1.82E+07	1,2,3	**2.176E+07**	1,3,2,4
DTLZ2_MINUS	3	1.88E+01	–	1.91E+01	1,3,4	1.89E+01	1	1.91E+01	1,3	**2.025E+01**	1,3,2,4
DTLZ3_MINUS	3	4.70E+09	–	4.76E+09	1,3	4.71E+09	1	4.76E+09	1,2,3	**5.053E+09**	1,3,2,4
DTLZ4_MINUS	3	1.88E+01	–	1.91E+01	1,3	1.88E+01	1	1.91E+01	1,3	**2.025E+01**	1,3,2,4
DTLZ5_MINUS	3	1.95E+01	2,3	1.95E+01	3	1.93E+01	–	1.96E+01	1,2,3	**2.024E+01**	1,3,2,4
DTLZ6_MINUS	3	5.90E+02	–	5.96E+02	1,3	5.92E+02	1	5.98E+02	1,2,3	**6.286E+02**	1,3,2,4
DTLZ7_MINUS	3	4.44E-01	2,3	1.80E-01	–	4.01E-01	2	4.45E-01	1,2,3	**4.453E-01**	1,3,2,4
DTLZ1_MINUS	4	5.00E+08	2,3	4.17E+08	3	1.25E+08	–	6.25E+08	1,2,3	**1.345E+09**	1,3,2,4
DTLZ2_MINUS	4	1.69E+01	–	2.20E+01	1,3,4	2.14E+01	1,4	2.05E+01	1	**3.223E+01**	1,3,2,4
DTLZ3_MINUS	4	2.67E+12	–	3.42E+12	1,3,4	3.39E+12	1,4	3.25E+12	1	**5.069E+12**	1,3,2,4
DTLZ4_MINUS	4	1.67E+01	–	2.18E+01	1,3,4	2.13E+01	1,4	2.04E+01	1	**3.224E+01**	1,3,2,4
DTLZ5_MINUS	4	2.31E+01	–	2.75E+01	1,3,4	2.60E+01	1,4	2.42E+01	1	**3.256E+01**	1,3,2,4
DTLZ6_MINUS	4	1.88E+03	–	2.32E+03	1,3,4	2.21E+03	1,4	2.13E+03	1	**3.145E+03**	1,3,2,4
DTLZ7_MINUS	4	**1.905E+00**	2,4,3,5	2.00E-01	–	1.40E+00	2	1.86E+00	2,3,5	1.85E+00	2,3
DTLZ1_MINUS	5	7.49E+08	2	6.10E+07	–	1.74E+10	1,2,4	9.61E+08	2	**4.107E+10**	1,3,2,4
DTLZ2_MINUS	5	1.58E+00	–	2.22E+01	1,3,4	2.00E+01	1,4	1.28E+01	1	**4.467E+01**	1,3,2,4
DTLZ3_MINUS	5	3.40E+14	–	2.24E+15	1,3,4	2.04E+15	1,4	1.32E+15	1	**4.424E+15**	1,3,2,4
DTLZ4_MINUS	5	1.66E+00	–	2.20E+01	1,3,4	1.88E+01	1,4	1.26E+01	1	**4.460E+01**	1,3,2,4
DTLZ5_MINUS	5	1.81E+01	–	3.55E+01	1,3,4	3.34E+01	1,4	2.06E+01	1	**4.563E+01**	1,3,2,4
DTLZ6_MINUS	5	2.26E+03	–	8.08E+03	1,3,4	7.36E+03	1,4	4.45E+03	1	**1.374E+04**	1,3,2,4
DTLZ7_MINUS	5	2.04E+00	2,3	1.00E-01	–	1.58E+00	2	**2.188E+00**	1,2,3	1.911E+00	2,3
DTLZ1_MINUS	6	2.73E+10	2	8.05E+08	–	**1.874E+11**	1,4,2,5	3.39E+10	2	1.13E+11	1,2,4
DTLZ2_MINUS	6	2.70E+00	–	2.58E+01	1,3,4	4.09E+00	1,4	3.01E+00	1	**4.872E+01**	1,3,2,4
DTLZ3_MINUS	6	2.08E+17	–	1.67E+18	1,3,4	4.62E+17	1,4	2.62E+17	1	**3.048E+18**	1,3,2,4
DTLZ4_MINUS	6	1.04E+00	–	2.46E+01	1,3,4	3.10E+00	1	1.68E+00	1	**4.834E+01**	1,3,2,4
DTLZ5_MINUS	6	2.06E+01	–	4.02E+01	1,3,4	2.96E+01	1,4	2.09E+01	1	**5.094E+01**	1,3,2,4
DTLZ6_MINUS	6	8.08E+03	–	2.93E+04	1,3,4	9.66E+03	1,4	8.47E+03	1	**4.756E+04**	1,3,2,4
DTLZ7_MINUS	6	**3.968E+00**	2,3,5	1.60E+00	–	3.48E+00	2	3.785E+00	2,3,5	3.19E+00	2
DTLZ1_MINUS	7	**3.333E+12**	2,3,5	1.24E+11	3,5	4.88E+09	5	3.321E+12	2,3,5	1.99E+09	–
DTLZ2_MINUS	7	3.24E+00	3	2.29E+01	1,3,4	2.30E-02	–	2.96E+00	3	**3.890E+01**	1,3,2,4
DTLZ3_MINUS	7	1.50E+20	3	9.72E+20	1,3,4	2.97E+18	–	1.60E+20	3	**1.553E+21**	1,3,2,4
DTLZ4_MINUS	7	1.17E+00	3	2.15E+01	1,3,4	3.53E-03	–	1.27E+00	3	**3.810E+01**	1,3,2,4
DTLZ5_MINUS	7	2.24E+01	3	2.98E+01	1,3,4	1.33E+01	–	2.24E+01	3	**4.337E+01**	1,3,2,4
DTLZ6_MINUS	7	2.64E+04	3	7.29E+04	1,3,4	2.52E+03	–	2.69E+04	3	**1.187E+05**	1,3,2,4
DTLZ7_MINUS	7	3.745E+00	2,5	1.60E+00	–	**3.919E+00**	2,5	3.457E+00	2,5	2.38E+00	2

Table 25: Number of DTLZ and WFG problems in which each scalarizing function obtained the best performance (or a statistically similar performance to the best one) using the hypervolume indicator.

Test problems	m	Total problems	ASF	PBI	TCH	GE_SF1	GE_SF2
DTLZ1-7, WFG1-9	2	16	1	3	5	**12**	6
	3	16	3	4	3	**9**	1
	4	16	4	**8**	1	**8**	0
	5	16	2	5	2	**10**	0
	6	16	1	6	2	**9**	1
	7	16	4	**10**	2	8	2
		Total	15	36	15	**56**	10

Table 26: Number of I-DTLZ problems in which each scalarizing function obtained the best performance (or a statistically similar performance to the best one) using the hypervolume indicator.

Test problems	m	Total problems	ASF	PBI	TCH	GE_SF1	GE_SF2
I-DTLZ1-7	2	7	1	0	1	1	**6**
	3	7	0	0	0	0	**7**
	4	7	1	0	0	0	**6**
	5	7	0	0	0	1	**7**
	6	7	1	0	1	1	**5**
	7	7	2	0	1	2	**5**
		Total	5	0	3	5	**36**

function for standard MOPs. In this comparison, MOMBI-II with GE_SF1 obtained the largest number of top results, with 54 out of 96, followed by NSGA-III with 40 top results.

Table 27: Comparison of the number of problems improved, using the hypervolume indicator, with different MOEAs in the DTLZ and the WFG test problems.

Test problems	m	Total problems	MOEA/D	NSGA-III	MOMBI-II$_{ASF}$	MOMBI-II$_{GESF1}$
DTLZ1-7, WFG1-9	2	16	4	5	0	**14**
	3	16	3	8	3	**10**
	4	16	2	**7**	6	**7**
	5	16	3	5	2	**9**
	6	16	1	5	2	**9**
	7	16	3	**10**	4	5
		Total	16	40	17	**54**

In the second comparison against other MOEAs, we compared the performance of GE_SF2, since it was the best performing scalarizing function for inverted MOPs. These results are shown in Table 28. Similar to the scalarizing functions comparison from Table 24, MOMBI-II with GE_SF2 obtained the best results across all dimensions, and significantly improved the results obtained by all the other MOEAs.

Table 28: Comparison of the number of problems improved, using the hypervolume indicator, with different MOEAs in the I-DTLZ problems.

Test problems	m	Total problems	MOEA/D	NSGA-III	MOMBI-II$_{ASF}$	MOMBI-II$_{GE_SF2}$
I-DTLZ1-7	2	7	1	2	1	**5**
	3	7	0	0	1	**6**
	4	7	0	0	1	**6**
	5	7	1	0	0	**7**
	6	7	0	1	1	**5**
	7	7	0	1	2	**5**
		Total	2	4	6	**34**

In Figure 5 we show the contour lines of the 5 scalarizing functions used in our comparisons, with 5 different weight vectors. All these plots are shown in the interval [0,1] for two objectives, for an easier comparison. We can observe that the contour lines from ASF and GE_SF1 are really similar. In an analogous way, the contour lines of TCH and GE_SF2 also share some similarities. However, GE_SF1 exhibits a sharper steepness with respect to ASF, since for all 5 weight vectors it reached a smaller value in the lower

values of the plot. Conversely, GE_SF2 exhibits a lower steepness when compared to TCH, since this time TCH reaches the smallest values with all the 5 weight vectors adopted.

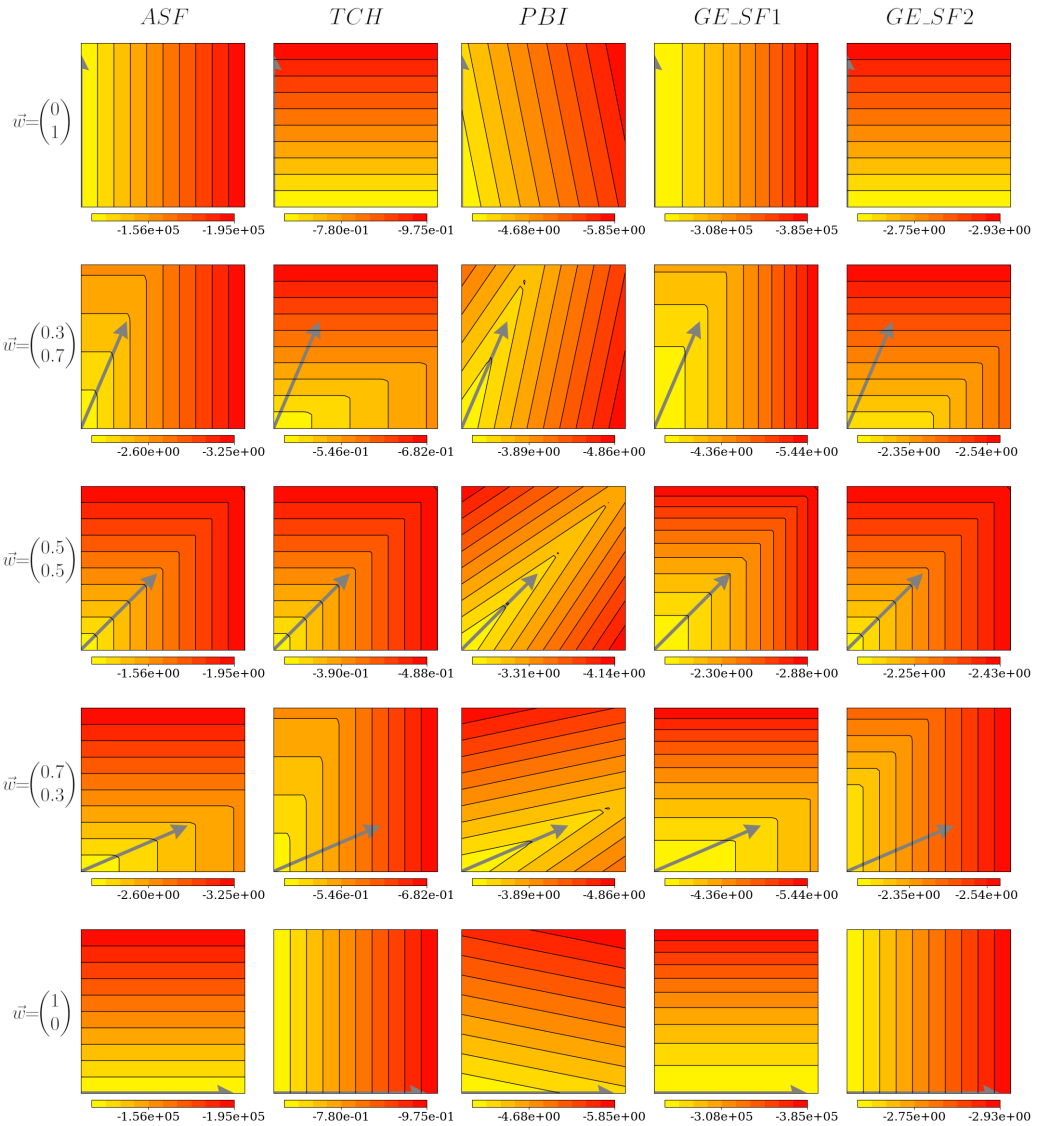
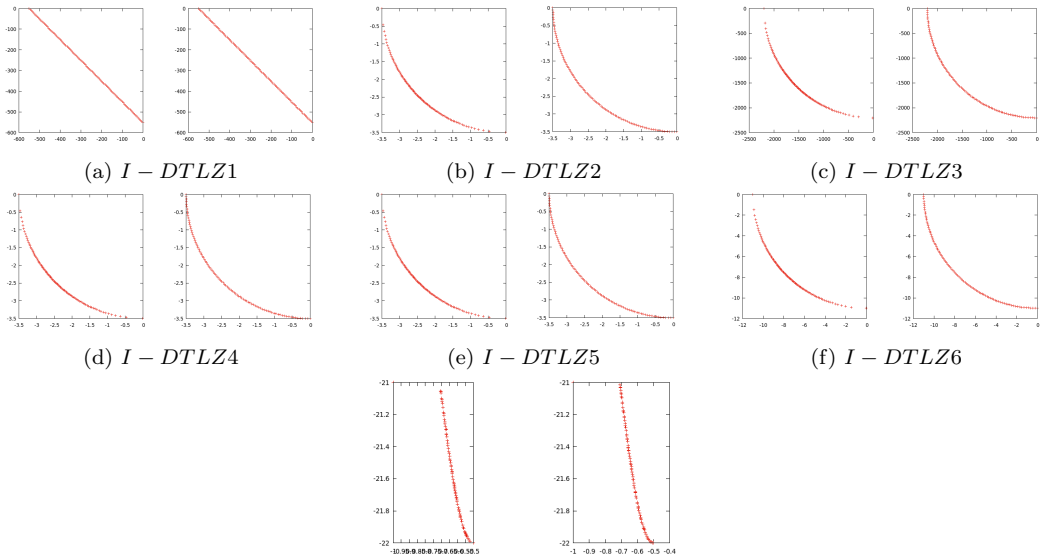


Figure 5: Contour lines for ASF, TCH, PBI, GE_SF1 and GE_SF2 with different weight vectors $\vec{w}$.

Finally, we show a comparison of the Pareto fronts obtained in each in-



(a) $I - DTLZ1$ (b) $I - DTLZ2$ (c) $I - DTLZ3$

(d) $I - DTLZ4$ (e) $I - DTLZ5$ (f) $I - DTLZ6$

(g) $I - DTLZ7$

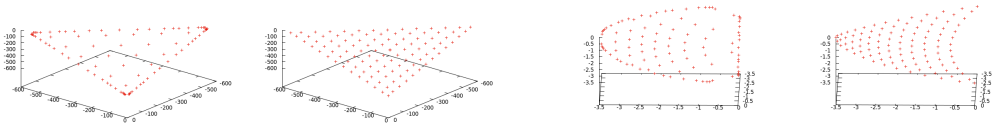
Figure 6: Pareto fronts obtained using ASF (left) and GE_SF2 (right) in DTLZ inverse problems with 2 objectives.

verted test problem with 2 objectives in Fig. 6 and in inverted problems with 3 objectives in Fig. 7. The fronts shown in these figures correspond to the results at the median of the hypervolume values obtained from 30 independent executions. In all cases, we can observe that the Pareto fronts generated with GE_SF2 have a better distribution than those generated with ASF.

It is particularly interesting to notice that there was no weight vector adaptation mechanism used in the inverse problems, meaning that the same weight vectors used for the standard DTLZ and WFG test problems are used to solve the inverse DTLZ problems. However, from the plots presented in Figures 6 and 7 it is noticeable that GE_SF2 is replacing the role of a weight vector adaptation, which would typically be the easier way to improve the obtained results in inverse problems.

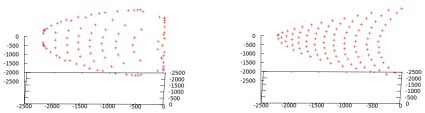
5.2. Hypervolume approximations

In order to validate the performance of our hypervolume approximation functions, we compared them against the Monte Carlo method, considering 10,000 sample points, against the $R_2^H V$ approximation and against the GP-generated approximations [30] using two measures: the average MSE and

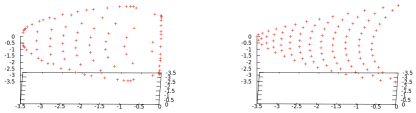


(a) $I - DTLZ1$

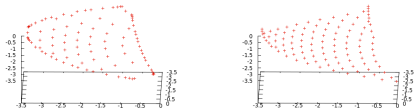
(b) $I - DTLZ2$



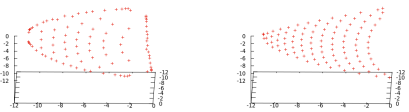
(c) $I - DTLZ3$



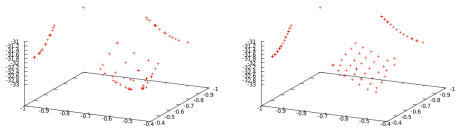
(d) $I - DTLZ4$



(e) $I - DTLZ5$



(f) $I - DTLZ6$



(g) $I - DTLZ7$

Figure 7: Pareto fronts obtained using ASF (left) and GE_SF2 (right) in the DTLZ inverse problems with 3 objectives.

the average execution time. All these approximations were obtained using Python in the same hardware mentioned in the previous section.

For the $R_2^H V$ approximation, we generated the direction vectors as follows. We used the simplex lattice design method [34] to generate a set of weight vectors. Then, for each weight vector $\vec{w}$, we generated the corresponding direction vector $\vec{\lambda} = \frac{\vec{w}}{\|\vec{w}\|_2}$, as stated in [44]. In total, we used 9870 direction vectors for 2 and 3-dimensions, 9880 vectors for 4-dimensions and 8855 vectors for 5 dimensions. On the other hand, the reference point $\vec{r} = \{r, \ldots, r\}$ was defined as follows for all the experiments.

$$r = 1 + \frac{1}{H} \tag{25}$$

where H is an integer satisfying $C_{m-1}^{H+m-1} \leq N < C_{m-1}^{H+m}$, and C_m^n is the total number of combinations for choosing m elements from a set of n elements, as used in [45].

In Table 29, we present the comparison of the average MSE obtained computing the different hypervolume approximations previously mentioned in the validation data. We used a gradient to better illustrate the results. The darker the cell, the better the value, which in this case is the smallest value. In all 4 cases, Monte Carlo obtained the smallest error with a two orders of magnitude improvement compared to the second best performing approximation, which are the approximations generated using our proposal. Although all 8 of our approximations share the same order of magnitude, which in turn is two or three orders of magnitude better than the GP-generated approximations, the use of the second grammar produced slightly better results in 3, 4 and 5-dimensional data. This hints that the use of statistical features provides useful information to generate better hypervolume approximation functions. The next best performing approximations are the GP-generated approximations. Lastly, R_2^{HV} obtained the worst result in all cases.

On the other hand, we present the comparison of the average computational time, per file, in Table 30. Once again, the darker the cell, the better the value, meaning the smallest value. Here, we can observe that the GP-generated approximations obtained the best results in all three available cases, whereas Monte Carlo obtained the worst results in 2 and 3-dimensional data, and R_2^{HV} obtained the worst results in 4 and 5-dimensional data. Regarding the approximations generated with our proposal, they ranged from one order up to two orders of magnitude worsening compared to the best results. This is an expected behavior, since our averaging variant involves an

Table 29: Average MSE comparison of hypervolume approximations in validation data.

m	Validation files	Average MSE				
		M^m	$GEHV1_{mD}$	$GEHV2_{mD}$	Monte Carlo	R_2^{HV}
2	2176	-	2.211E-03	3.721E-03	2.209E-05	4.709E-02
3	2906	1.431E-01	2.242E-03	2.018E-03	1.542E-05	8.770E-01
4	2925	2.129E-01	5.879E-03	2.436E-03	1.459E-05	9.202E+00
5	2836	2.765E-01	5.272E-03	5.238E-03	1.395E-05	7.357E+01

additional iteration of the data against the GP-generated approximations. Additionaly, most of the functions generated with our proposal are more complex. However, three of our eight functions obtained relatively better execution times due to them being the simplest equations generated with our proposal, namely $GEHV1_{4D}$, $GEHV2_{2D}$ and $GEHV2_{5D}$.

Table 30: Average computational time comparison of hypervolume approximations in validation data.

m	Validation files	Average Computation Time (s)				
		M^m	$GEHV1_{mD}$	$GEHV2_{mD}$	Monte Carlo	R_2^{HV}
2	2176	-	2.050E-02	1.983E-03	2.224E+00	1.518E+00
3	2906	9.935E-04	1.755E-02	3.288E-02	3.032E+00	2.910E+00
4	2925	1.038E-03	2.617E-03	3.988E-02	2.405E+00	3.873E+00
5	2836	1.113E-03	3.311E-02	2.670E-03	2.340E+00	4.518E+00

From these results we can notice that although Monte Carlo obtained the best approximations in terms of quality (measured by MSE), it is also the most computationally costly. In contrast, our functions obtained better values in terms of quality against the GP-generated approximations at the expense of increasing the computational time required.

Finally, we used the average ratio $\frac{HV_{approx}}{HV_{real}}$ to better illustrate the quality of the approximations obtained with our functions against the GP-generated approximations. The closer this value is to 100%, the closer the approximation is to the real hypervolume value. A value smaller than 100% represents an underestimation. Consequently a value greater than 100% indicates an overestimation of the hypervolume. In Tables 31 to 33 we show the average ratio for all available approximations in the validation data classified by the

problem used to generate each file.

Table 31: Average $\frac{HV_{approx}}{HV_{real}}$ ratio comparison of hypervolume approximations in 3-dimensional validation data.

Validation set	Validation files	HV_{approx}/HV_{real}		
		$M^3_{4,6}$	$GEHV1_{3D}$	$GEHV2_{3D}$
DTLZ1_3D	394	143.67%	94.50%	93.59%
DTLZ2_3D	467	161.85%	99.23%	106.01%
DTLZ5_3D	416	333.23%	110.20%	106.61%
DTLZ7_3D	401	184.97%	94.77%	99.97%
WFG1_3D	362	164.59%	105.03%	99.73%
WFG2_3D	432	150.79%	97.81%	93.96%
WFG3_3D	434	165.92%	113.92%	102.01%
Average		186.43%	102.21%	100.27%

Table 32: Average $\frac{HV_{approx}}{HV_{real}}$ ratio comparison of hypervolume approximations in 4-dimensional validation data.

Validation set	Validation files	HV_{approx}/HV_{real}		
		$M^4_{5,6}$	$GEHV1_{4D}$	$GEHV2_{4D}$
DTLZ1_4D	465	151.95%	90.89%	95.04%
DTLZ2_4D	433	172.80%	106.59%	106.68%
DTLZ5_4D	428	223.63%	105.03%	98.16%
DTLZ7_4D	421	199.50%	96.65%	99.92%
WFG1_4D	368	161.74%	96.78%	100.02%
WFG2_4D	399	153.36%	93.39%	97.64%
WFG3_4D	411	168.79%	118.16%	99.67%
Average		175.97%	101.07%	99.59%

From these results we can observe that the approximation functions generated with our proposal have a consistent behavior close to 100% in the final average for all three tables. However, this does not occur with the GP-generated approximations. In 3-dimensional data there is an average

Table 33: Average $\frac{HV_{approx}}{HV_{real}}$ ratio comparison of hypervolume approximations in 5-dimensional validation data.

Validation set	Validation files	HV_{approx}/HV_{real}		
		$M^5_{1,5}$	$GEHV1_{5D}$	$GEHV2_{5D}$
DTLZ1_5D	425	161.56%	93.58%	94.70%
DTLZ2_5D	427	160.83%	96.46%	100.86%
DTLZ5_5D	391	171.04%	96.86%	91.84%
DTLZ7_5D	400	189.48%	107.87%	99.04%
WFG1_5D	390	168.16%	96.92%	98.99%
WFG2_5D	386	163.94%	95.71%	97.06%
WFG3_5D	417	199.49%	114.09%	120.85%
Average		173.50%	100.21%	100.48%

overestimation of 186.43&, whereas in 4-dimensional data there is an average underestimation of 175.97%. Finally, in 5-dimensional data, the average overestimation consists of 173.5%.

6. Conclusions and Future work

In this work we have presented a methodology that allows the automatic generation of scalarizing functions as well as hypervolume approximations, using grammatical evolution.

Using this methodology we have proposed two different scalarizing functions (GE_SF1 and GE_SF2). The former was obtained using DTLZ4 and WFG4 and the latter was obtained using I-DTLZ4. We have provided experimental evidence that shows that these two functions outperform ASF, TCH and PBI in the test problems considered, as well as other MOEAs such as MOEA/D and NSGA-III. GE_SF1 obtained the largest number of wins in the comparisons using the standard DTLZ and WFG problems, whereas GE_SF2 obtained the largest number of wins in the comparisons using the inverted DTLZ problems. Since our methodology employs hypervolume calculations, it can become computationally expensive. In the worst case, which occurred when using WFG4 as the training problem, it took 12,056 seconds to complete 40 generations, averaging close to 300 seconds per generation. However, it is important to notice that this is the cost of generating the scalarizing function. Once we have obtained it, using such a scalarizing function has a similar computational cost to that of ASF or TCH.

Also, in the case of standard benchmark problems, the percentage of problems with top results obtained with GE_SF1 was under 60%, which evidences that even the best performing scalarizing function from our comparisons is not able to generalize the improvements in all benchmark problems. Thus, we can conjecture that in order to achieve improvements in more test instances, an ensemble of multiple complementary scalarizing functions could be used, and possibly couple it to a weight vector adaptation mechanism.

Furthermore, we have also proposed one variant to generate hypervolume approximations and we obtained two different approximation functions for 2, 3, 4 and 5-dimensional data. From our results we can conclude that the use of statistical features seems to improve the quality of the approximation functions generated using our proposal. In all cases, we found a consistent behavior both in terms of quality and computational cost: Monte Carlo obtained the best quality values while the GP-generated approximations obtained the best execution time values. However, approximations found with our proposal obtained better trade-offs between these two measures, since they consistently obtained better quality values when compared against the GP-generated approximations at a significantly lower computational time

when compared against the Monte Carlo method.

Even though the hypervolume becomes considerably computationally expensive with 5 or more objectives, we consider that the methodology that we proposed here can be easily extended to generate approximations for higher dimensional spaces.

Also, we showed a ratio comparison, between the hypervolume approximation and the real hypervolume, which we consider to be evidence that the approximation functions generated with GP are possibly not generalizing its good performance in the type of data files we created in this work. This is particularly noticeable in validation data generated for DTLZ5 with 3 dimensions (with an average overestimation of 333.23%) and DTLZ5 with 4 dimensions (with an average overestimation of 223.63%). These are the extreme cases, but we can find bad quality approximations across all validation data adopted. However, this same behavior is potentially also present in the approximations we created using GE for different validation data. We believe that both GP and GE are able to generate good results for a certain type of data and even obtain a good generalization but up to a certain point. And in order to ensure a consistent capability to generalize in different data/problems it seems to be necessary to use a large number of different training data.

Finally, in both methodologies presented in this paper, the functions found using GE were trained using a specific problem or specific data, but there is evidence that they can generalize their good performance to problems/data that were not considered in the training. Based on this evidence, we believe that this implementation can be used to obtain better results in a wide variety of problems. Here, we have presented its applications in two particular cases: a scalarizing function and hypervolume approximations. These two cases can benefit decomposition-based and indicator-based MOEAs, respectively.

However, we claim that a similar methodology can be used to obtain different MOEA components, such as new density estimators, or even new performance indicators, and this will be, indeed, part of our future work.

Acknowledgements

Carlos A. Coello Coello gratefully acknowledges support from CONACyT grant no. 2016-01-1920 (Investigación en Fronteras de la Ciencia 2016).

References

[1] T. Stewart, O. Bandte, H. Braun, N. Chakraborti, M. Ehrgott, M. Göbelt, Y. Jin, H. Nakayama, S. Poles, D. D. Stefano, Real-world applications of multiobjective optimization, in: Multiobjective Optimization, Springer Berlin Heidelberg, 2008, pp. 285–327. doi:10.1007/978-3-540-88908-3_11.

[2] C. He, Y. Tian, H. Wang, Y. Jin, A repository of real-world datasets for data-driven evolutionary multiobjective optimization, Complex & Intelligent Systems 6 (1) (2019) 189–197. doi:10.1007/s40747-019-00126-2.

[3] C. A. C. Coello, A comprehensive survey of evolutionary-based multiobjective optimization techniques, Knowledge and Information Systems 1 (3) (1999) 269–308. doi:10.1007/bf03325101.

[4] A. Trivedi, D. Srinivasan, K. Sanyal, A. Ghosh, A survey of multiobjective evolutionary algorithms based on decomposition, IEEE Transactions on Evolutionary Computation (2016) 1–1doi:10.1109/tevc.2016.2608507.

[5] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, Q. Zhang, Multiobjective evolutionary algorithms: A survey of the state of the art, Swarm and Evolutionary Computation 1 (1) (2011) 32–49. doi:10.1016/j.swevo.2011.03.001.

[6] Q. Xu, Z. Xu, T. Ma, A survey of multiobjective evolutionary algorithms based on decomposition: Variants, challenges and future directions, IEEE Access 8 (2020) 41588–41614. doi:10.1109/access.2020.2973670.

[7] K. Miettinen, M. M. Mäkelä, On scalarizing functions in multiobjective optimization, OR Spectrum 24 (2) (2002) 193–213. doi:10.1007/s00291-001-0092-9.

[8] M. Pescador-Rojas, R. H. Gómez, E. Montero, N. Rojas-Morales, M.-C. Riff, C. A. C. Coello, An overview of weighted and unconstrained scalarizing functions, in: Lecture Notes in Computer Science, Springer International Publishing, 2017, pp. 499–513. doi:10.1007/978-3-319-54157-0_34.

[9] X. Zhou, X. Wang, X. Gu, A decomposition-based multiobjective evolutionary algorithm with weight vector adaptation, Swarm and Evolutionary Computation 61 (2021) 100825. doi:10.1016/j.swevo.2020.100825.
URL https://doi.org/10.1016/j.swevo.2020.100825

[10] D. Brockhoff, T. Wagner, H. Trautmann, On the Properties of the R2 Indicator, in: 2012 Genetic and Evolutionary Computation Conference (GECCO'2012), ACM Press, Philadelphia, USA, 2012, pp. 465–472, iSBN: 978-1-4503-1177-9.

[11] A. V. B. Rodríguez, C. A. C. Coello, Generation of new scalarizing functions using genetic programming, in: Parallel Problem Solving from Nature – PPSN XVI, Springer International Publishing, 2020, pp. 3–17. doi:10.1007/978-3-030-58115-2_1.
URL https://doi.org/10.1007/978-3-030-58115-2_1

[12] E. Zitzler, S. Künzli, Indicator-based selection in multiobjective search, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2004, pp. 832–842. doi:10.1007/978-3-540-30217-9_84.

[13] J. G. Falcón-Cardona, S. Zapotecas-Martínez, A. García-Nájera, Pareto compliance from a practical point of view, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2021, pp. 395–402. doi:10.1145/3449639.3459276.
URL https://doi.org/10.1145/3449639.3459276

[14] R. Hernández Gómez, Parallel Hyper-Heuristics for Multi-Objective Optimization, Ph.D. thesis, Department of Computer Science, CINVESTAV-IPN, Mexico City, México (March 2018).

[15] L. While, L. Bradstreet, L. Barone, A fast way of calculating exact hypervolumes, IEEE Transactions on Evolutionary Computation 16 (1) (2012) 86–95. doi:10.1109/tevc.2010.2077298.
URL https://doi.org/10.1109/tevc.2010.2077298

[16] J. Bader, K. Deb, E. Zitzler, Faster hypervolume-based search using monte carlo sampling, in: Lecture Notes in Economics and Mathematical Systems, Springer Berlin Heidelberg, 2009, pp. 313–326. doi:10.1007/978-3-642-04045-0_27.
URL https://doi.org/10.1007/978-3-642-04045-0_27

[17] J. Deng, Q. Zhang, Approximating hypervolume and hypervolume contributions using polar coordinate, IEEE Transactions on Evolutionary Computation 23 (5) (2019) 913–918. doi:10.1109/tevc.2019.2895108.
URL https://doi.org/10.1109/tevc.2019.2895108

[18] K. Shang, W. Liao, H. Ishibuchi, HVC-net: Deep learning based hypervolume contribution approximation, in: Lecture Notes in Computer Science, Springer International Publishing, 2022, pp. 414–426. doi:10.1007/978-3-031-14714-2_29.
URL https://doi.org/10.1007/978-3-031-14714-2_29

[19] J. R. Koza, Hierarchical genetic algorithms operating on populations of computer programs., in: IJCAI, Vol. 89, Springer-Verlag, 1989, pp. 768–774.

[20] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Transactions on Evolutionary Computation 5 (4) (2001) 349–358. doi:10.1109/4235.942529.

[21] M. Nicolau, A. Agapitos, Understanding grammatical evolution: Grammar design, in: Handbook of Grammatical Evolution, Springer International Publishing, 2018, pp. 23–53. doi:10.1007/978-3-319-78717-6_2.
URL https://doi.org/10.1007/978-3-319-78717-6_2

[22] R. Aler, D. Borrajo, P. Isasi, Grammars for learning control knowledge with GP, in: Proceedings of the 2001 Congress on Evolutionary Computation (IEEE Cat. No.01TH8546), IEEE, 2001. doi:10.1109/cec.2001.934330.
URL https://doi.org/10.1109/cec.2001.934330

[23] D. Borrajo, M. Veloso, Artificial Intelligence Review 11 (1/5) (1997) 371–405. doi:10.1023/a:1006549800144, [link].
URL https://doi.org/10.1023/a:1006549800144

[24] R. Aler, D. Borrajo, P. Isasi, Using genetic programming to learn and improve control knowledge, Artificial Intelligence 141 (1-2) (2002) 29–56. doi:10.1016/s0004-3702(02)00246-1.
URL https://doi.org/10.1016/s0004-3702(02)00246-1

[25] D. Rivero, J. Dorado, J. R. Rabuñal, A. Pazos, Modifying genetic programming for artificial neural network development for data mining,

Soft Computing 13 (3) (2008) 291–305. doi:10.1007/s00500-008-0317-9.
URL https://doi.org/10.1007/s00500-008-0317-9

[26] D. Rivero, J. Dorado, J. Rabuñal, A. Pazos, Generation and simplification of artificial neural networks by means of genetic programming, Neurocomputing 73 (16-18) (2010) 3200–3223. doi:10.1016/j.neucom.2010.05.010.
URL https://doi.org/10.1016/j.neucom.2010.05.010

[27] F. Zhang, Y. Mei, S. Nguyen, M. Zhang, Evolving scheduling heuristics via genetic programming with feature selection in dynamic flexible job-shop scheduling, IEEE Transactions on Cybernetics 51 (4) (2021) 1797–1811. doi:10.1109/tcyb.2020.3024849.
URL https://doi.org/10.1109/tcyb.2020.3024849

[28] W. L. Cava, T. Helmuth, L. Spector, K. Danai, Genetic programming with epigenetic local search, in: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO 15, ACM Press, 2015. doi:10.1145/2739480.2754763.

[29] R. Hernández Gómez, C. A. C. Coello, Improved metaheuristic based on the r2 indicator for many-objective optimization, in: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO 15, ACM Press, 2015, p. 679–686. doi:10.1145/2739480.2754776.

[30] C. Sandoval, O. Cuate, L. C. González, L. Trujillo, O. Schütze, Towards fast approximations for the hypervolume indicator for multi-objective optimization problems by genetic programming, Applied Soft Computing 125 (2022) 109103. doi:10.1016/j.asoc.2022.109103.
URL https://doi.org/10.1016/j.asoc.2022.109103

[31] M. Fenton, J. McDermott, D. Fagan, S. Forstenlechner, M. O'Neill, E. Hemberg, Ponyge2: Grammatical evolution in python (2017). doi:10.48550/ARXIV.1703.08535.

[32] A. P. Wierzbicki, The use of reference objectives in multiobjective optimization, in: Lecture Notes in Economics and Mathematical Systems, Springer Berlin Heidelberg, 1980, pp. 468–486. doi:10.1007/978-3-642-48782-8_32.
URL https://doi.org/10.1007/978-3-642-48782-8_32

[33] V. J. Bowman, On the relationship of the tchebycheff norm and the efficient frontier of multiple-criteria objectives, in: Lecture Notes in Economics and Mathematical Systems, Springer Berlin Heidelberg, 1976, pp. 76–86. doi:10.1007/978-3-642-87563-2_5.
URL https://doi.org/10.1007/978-3-642-87563-2_5

[34] Q. Zhang, H. Li, MOEA/d: A multiobjective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 11 (6) (2007) 712–731. doi:10.1109/tevc.2007.892759.
URL https://doi.org/10.1109/tevc.2007.892759

[35] J. Bader, E. Zitzler, HypE: An algorithm for fast hypervolume-based many-objective optimization, Evolutionary Computation 19 (1) (2011) 45–76. doi:10.1162/evco_a_00009.
URL https://doi.org/10.1162/evco_a_00009

[36] K. Shang, H. Ishibuchi, X. Ni, R2-based hypervolume contribution approximation, IEEE Transactions on Evolutionary Computation 24 (1) (2020) 185–192. doi:10.1109/tevc.2019.2909271.
URL https://doi.org/10.1109/tevc.2019.2909271

[37] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable test problems for evolutionary multiobjective optimization, in: Advanced Information and Knowledge Processing, Springer-Verlag, 2005, pp. 105–145. doi:10.1007/1-84628-137-7_6.

[38] S. Huband, L. Barone, L. While, P. Hingston, A scalable multi-objective test problem toolkit, in: Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2005, pp. 280–295. doi:10.1007/978-3-540-31880-4_20.

[39] J. G. Falcon-Cardona, H. Ishibuchi, C. A. Coello Coello, Exploiting the Trade-off between Convergence and Diversity Indicators, in: 2020 IEEE Symposium Series on Computational Intelligence (SSCI'2020), IEEE, 2020, pp. 141–148, iSBN 978-1-7281-2548-0. doi:10.1109/ssci47803.2020.9308469.
URL https://doi.org/10.1109/ssci47803.2020.9308469

[40] H. Jain, K. Deb, An evolutionary many-objective optimization algorithm using reference-point based nondominated sorting approach,

part II: Handling constraints and extending to an adaptive approach, IEEE Transactions on Evolutionary Computation 18 (4) (2014) 602–622. doi:10.1109/tevc.2013.2281534.

[41] K. Deb, H. Jain, 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) (2014) 577–601. doi:10.1109/tevc.2013.2281535.
URL https://doi.org/10.1109/tevc.2013.2281535

[42] Y. Tian, R. Cheng, X. Zhang, Y. Jin, PlatEMO: A MATLAB platform for evolutionary multi-objective optimization [educational forum], IEEE Computational Intelligence Magazine 12 (4) (2017) 73–87. doi:10.1109/mci.2017.2742868.
URL https://doi.org/10.1109/mci.2017.2742868

[43] R. H. Gómez, J. G. Falcón-Cardona, C. A. C. Coello, Considerations in the incremental hypervolume algorithm of the WFG, in: Advances in Computational Intelligence, Springer Nature Switzerland, 2022, pp. 410–422. doi:10.1007/978-3-031-19493-1_32.
URL https://doi.org/10.1007/978-3-031-19493-1_32

[44] K. Shang, H. Ishibuchi, M.-L. Zhang, Y. Liu, A new r2 indicator for better hypervolume approximation, in: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, 2018. doi:10.1145/3205455.3205543.
URL https://doi.org/10.1145/3205455.3205543

[45] K. Shang, H. Ishibuchi, A new hypervolume-based evolutionary algorithm for many-objective optimization, IEEE Transactions on Evolutionary Computation 24 (5) (2020) 839–852. doi:10.1109/tevc.2020.2964705.
URL https://doi.org/10.1109/tevc.2020.2964705