Designing Scalarizing Functions Using Grammatical Evolution

Amín V. Bernabé Rodríguez
amin.bernabe@cinvestav.mx
CINVESTAV-IPN, Computer Science Department
Mexico City, Mexico

Carlos A. Coello Coello
ccoello@cs.cinvestav.mx
CINVESTAV-IPN, Computer Science Department
Mexico City, Mexico

ABSTRACT

In this paper, we present a grammatical evolution-based framework to produce new scalarizing functions, which are known to have a significant impact on the performance of both decomposition-based multi-objective evolutionary algorithms (MOEAs) and indicator-based MOEAs which use *R2*. We perform two series of experiments using this framework. First, we produce many scalarizing functions using different benchmark problems to explore the behavior of the resulting functions according to the geometry of the problem adopted to generate them. Then, we perform a second round of experiments adopting two combinations of problems which yield better results in some test instances. We present the experimental validation of these new functions compared against the Achievement Scalarizing Function (ASF), which is known to provide a very good performance. For this comparative study, we adopt several benchmark problems and we are able to show that our proposal is able to generate new scalarizing functions that can outperform ASF in different problems.

CCS CONCEPTS

• **Mathematics of computing** → *Evolutionary algorithms*; • **Theory of computation** → *Genetic programming*.

KEYWORDS

multi-objective optimization, genetic programming, grammatical evolution, scalarizing functions

1 INTRODUCTION

Optimization problems where two or more objectives need to be simultaneously optimized are common in many fields [15]. They are commonly known as multi-objective optimization problems (MOPs), and Multi-objective Evolutionary Algorithms (MOEAs) are a popular choice to solve such problems.

MOEAs present some important advantages compared to classical mathematical programming techniques, from which, perhaps the most remarkable is that MOEAs operate on a set of solutions (called population). This allows MOEAs (if properly manipulated) to generate several optimal solutions in a single execution, which contrasts with mathematical programming techniques, which normally generate a single optimal solution per execution. Additionally, MOEAs require little domain-specific information, contrasting with mathematical programming techniques which normally require additional information (e.g., the gradient of the objectives and constraints of a MOP) [3].

Today, there are three main types of MOEAs available in the specialized literature:

(1) **Pareto-based MOEAs**: These algorithms were developed during the 1990s, and use a ranking procedure (called nondominated sorting) based on Pareto optimality to classify solutions. They also adopt a mechanism responsible for maintaining diversity (which is called density estimator). These MOEAs were very popular for several years, but their use is not effective in MOPs having more than three objectives (the so-called many-objective problems). The reason is that the number of nondominated solutions grows exponentially with the number of objectives, and this quickly dilutes the selection pressure [16].
(2) **Indicator-based MOEAs:** In this case, the idea is to use a performance indicator to select solutions instead of using Pareto optimality [19]. Indicator-based MOEAs were seen first as a curiosity but they attracted a lot of attention when it was possible to corroborate that they are more robust than Pareto-based MOEAs to many-objective problems. However, the main issue with indicator-based MOEAs is that the only performance indicator currently known to be fully Pareto compliant (i.e., strictly monotonic with respect to Pareto optimality) is the hypervolume, which is known to have a computational cost that increases polynomially on the number of solutions but exponentially on the number of objectives. Although indicator-based MOEAs based on *R2* (which is weakly Pareto compliant) are computationally efficient and have a good performance, their use is not very popular today. Approaches based on *R2* use a scalarizing function and their performance is sensitive to the specific scalarizing function adopted [8].
(3) **Decomposition-based MOEAs:** The idea of using decomposition (or scalarization) methods was originally proposed in mathematical programming more than 20 years ago [4] and it consists in transforming the given MOP into several single-objective optimization problems (SOPs) which are then solved to generate the nondominated solutions of the original MOP. Unlike linear aggregating functions, the use of scalarization (or decomposition) methods allows the generation of non-convex portions of the Pareto front and works even in disconnected Pareto fronts. The Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D), introduced in 2007 [17] presents an important advantage with respect to methods proposed in the mathematical programming literature (such as Normal Boundary Intersection (NBI) [4]): it uses neighborhood search to solve simultaneously all the SOPs generated from the transformation. Additionally, MOEA/D is not only effective and efficient, but can also be used for solving MOPs with more than 3 objectives although in such cases it will require higher population sizes. It is worth noting however, that

the performance of decomposition-based MOEAs is closely related to the scalarizing function adopted [7].

There are several scalarizing functions available in the specialized literature [13], each of which may require different parameters, or have different properties such as the type of solutions which can be found with them (for instance Pareto-optimal solutions or weekly Pareto-optimal solutions). However, to the best of the authors' knowledge, all of them are mathematical equations that have been proposed by humans. The only exception that we are aware of is the work presented in [14], where an implementation of genetic programming was used to generate new scalarizing functions which can be used in decomposition-based MOEAs.

In this work, we present a grammatical evolution (GE) based implementation to automatically generate scalarizing functions as well as some experimental work to validate the performance of said functions in different benchmark MOPs. This paper extends the preliminary work presented in [14] by Bernabé and Coello, where epigenetic linear genetic programming (ELGP) was used to generate scalarizing functions. ELGP is a variant of genetic programming (GP) with local search which was initially used to solve symbolic regression problems [2]. However, in their implementation, they combined ELGP with MOMBI-II (an indicator-based MOEA which employs scalarizing functions) to create two new scalarizing functions by solving one benchmark MOP at a time.

Our proposal in this work is to use a `Python`-based GE implementation (called PonyGE2) [6] instead of the `C`-based GP implementation used in [14], as the former is more flexible and allows to use more complex fitness functions, combining two or more benchmark MOPs in the search process of the scalarizing functions. This allows us to design functions combining a certain desired behavior shown in our experiments.

The remainder of this paper is organized as follows. Section 2 provides some basic concepts on multiobjective optimization to make the paper self-contained. In Section 3 we introduce GE as well as the implementation used. Then, in Section 4 we present the details of our implementation to generate scalarizing functions. Next, in Section 5 we show the experimental setup used to evaluate the performance of the scalarizing functions generated, and we present the results obtained and their discussion in Section 6. Finally, in Section 7 we provide our conclusions as well as some possible paths for future research.

2 BASIC CONCEPTS

In multiobjective optimization, the aim is to solve problems of the type[1]:

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

subject to:

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

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

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

[1]Without loss of generality, we will assume only minimization problems.

A few additional definitions are required to introduce the notion of optimality used in multiobjective optimization:

Definition 1. Given two vectors $\vec{x}, \vec{y} \in \mathbf{R}^k$, 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 $\vec{x} \in \mathcal{X} \subset \mathbf{R}^n$ is **nondominated** 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 $\vec{x}^* \in \mathcal{F} \subset \mathbf{R}^n$ ($\mathcal{F}$ is the feasible region) is **Pareto-optimal** if it is nondominated 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:

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

Therefore, our aim is to obtain the Pareto optimal set from the set $\mathcal{F}$ of all the decision variable vectors that satisfy (2) and (3). Note however that in practice, not all the Pareto optimal set is normally desirable or achievable, and decision makers tend to prefer certain types of solutions or regions of the Pareto front [1]. Additionally, the Pareto fronts may have different geometries such as linear, concave, convex or combinations of them. Also, there are some degenerate fronts, which are of a lower dimension than the objective space in which they are embedded [9].

3 GRAMMATICAL EVOLUTION

Genetic Programming (GP) is an evolutionary computation technique used to automatically generate computer programs. Similar to traditional evolutionary algorithms, in this case, there is a population of feasible solutions which are evaluated using a fitness function to measure how well each individual performs at solving a given problem. However, the main difference is that while traditional evolutionary algorithms encode a set of decision variables (e.g., a vector of real numbers), the individuals used in GP encode some sort of executable code [11].

On the other hand, Grammatical Evolution (GE) is a variant of GP, in which the individuals are encoded using either a list of integers or a binary list. In contrast, GP usually employs a tree data structure to encode the individuals. In order to perform the genotype-phenotype mapping, a Backus-Naur form (BNF) grammar must be provided in the case of GE. This grammar determines the nature of the solutions encoded by the individuals [12].

In order to implement GE, we used `PonyGE2`, which is a Python implementation available in the public domain [6]. `PonyGE2` is a GE implementation that allows the user to solve a wide variety of problems such as regression, classification and even multi-objective optimization. However, we added some elements to generate scalarizing functions by combining it with a MOEA as described in the following section. In all the experiments presented in this work,

our GE used position independent grow to initialize the population, tournament selection of size 2, variable one point crossover (probability of 0.75) and codon mutation (probability of one over the length of the genome).[2].

4 OUR PROPOSAL

We used two main components to generate scalarizing functions. First, we adopted PonyGE2 which, given the appropriate BNF grammar, is able to generate the individuals encoding the functions and iteratively combine them using genetic operators to find a good performing function. However, we need another component to measure how well these generated functions work as scalarizing functions. Hence, the second component used is a modified MOEA which is able to use the generated scalarizing functions to solve a MOP. Then, we measure the quality of the solutions obtained using the hypervolume indicator [18], and we use this value to assign the fitness of the individuals. The reference point used for the hypervolume is fixed for each MOP, since the true Pareto fronts are already known. This is done in order to correctly compare different scalarizing functions.

We used MOMBI-II [7], a metaheuristic based on the $R2$ performance indicator, but that uses scalarizing functions. It can be used with several scalarizing functions such as the Achievement Scalarizing Functions (ASF), the Tchebycheff Scalarizing Function and the Penalty-based Boundary Intersection (PBI), among others. Given a decision vector $\vec{x} \in \mathbf{R}^m$ and a reference point $\vec{z} \in \mathbf{R}^m$, we use the image of $\vec{x}$ in objective space modified by $\vec{z}$ as follows: $\vec{f'} := \vec{F}(\vec{x}) - \vec{z}$. Then, given a weight vector $\vec{w} \in \mathbf{R}^m$ we can define ASF:

$$ASF(\vec{f'}, \vec{w}) := \max_i \left(\frac{f'_i}{w_i} \right) \tag{4}$$

This is the function used by default in MOMBI-II, and is also the scalarizing function adopted for our comparisons in all our experiments.

4.1 Fitness function

In order to measure the fitness of a given scalarizing function, we perform the following steps:

(1) We decode the genotype to obtain the final phenotype which can be interpreted by MOMBI-II.
(2) The scalarizing function previously used in MOMBI-II is replaced with the new decoded phenotype obtained.
(3) The modified version of MOMBI-II is used to solve a MOP with a relatively low number of function evaluations (10,000 in the experiments presented here). This corresponds to the number of times the objective vector is computed in MOMBI-II. The purpose of using "few" function evaluations is to avoid spending resources in scalarizing functions which may have a really low performance. Then, the hypervolume values of the resulting Pareto fronts are obtained and averaged. If they are smaller than a given threshold, the final fitness of the individual is the same as the average hypervolume obtained. If they are greater than the threshold, the next step is performed.
(4) The same modified version of MOMBI-II is used to solve either one or several MOPs with a greater number of function evaluations (100,000 in the experiments presented here). Then, the hypervolume values of the resulting Pareto fronts are obtained, averaged, and assigned as the final individual's fitness.

The MOPs used to validate the scalarizing functions' performance belong to the Deb-Thiele-Zitzler (DTLZ) [5] and Walking-Fish-Group (WFG) [9] test suites. We adopted these MOPs since they present a variety of MOPs with different geometries and characteristics. Additionally, the true Pareto fronts of these MOPs are known, which allows us to obtain the maximum hypervolume given a reference point and set the threshold used in step (3).

4.2 Grammar

For simplicity sake, in the grammar used to generate the scalarizing functions we replaced the image of the decision vector previously denoted by f'_i with variable x, and weight vector w_i with variable y. The complete grammar used is the following.

⟨e⟩ ::= max{⟨f⟩}+⟨g⟩ | max{⟨f⟩}+0.⟨c⟩⟨c⟩*⟨g⟩ |
max{⟨f⟩}-⟨g⟩	max{⟨f⟩}-0.⟨c⟩⟨c⟩*⟨g⟩
max{⟨f⟩}*⟨g⟩	max{⟨f⟩}*0.⟨c⟩⟨c⟩*⟨g⟩
max{⟨f⟩}/⟨g⟩	max{⟨f⟩}/0.⟨c⟩⟨c⟩*⟨g⟩
max{⟨f⟩}

⟨f⟩ ::= ⟨*optional_var*⟩+⟨f⟩ | ⟨f⟩+⟨f⟩ |
⟨*optional_var*⟩-⟨f⟩	⟨f⟩-⟨*optional_var*⟩	⟨f⟩-⟨f⟩
⟨*optional_var*⟩*⟨f⟩	⟨f⟩*⟨f⟩
⟨*optional_var*⟩/⟨f⟩	⟨f⟩/⟨*optional_var*⟩	⟨f⟩/⟨f⟩
sqrt(⟨f⟩)
⟨*needed_var*⟩

⟨g⟩ ::= ⟨g⟩+⟨g⟩ | ⟨g⟩-⟨g⟩ | ⟨g⟩*⟨g⟩ | ⟨g⟩/⟨g⟩ | sqrt(⟨g⟩) |
| ⟨*optional_var*⟩

⟨*needed_var*⟩ ::= x | y | x/y

⟨*optional_var*⟩ ::= x_sum | x_avg | x_max | x_min |
| ⟨c⟩⟨c⟩.⟨c⟩⟨c⟩ | 0.⟨c⟩⟨c⟩

⟨c⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

The main derivation rule $< e >$ can be formed by two elements:

- A max operator is applied to obtain the maximum value of a given expression generated using derivation rule $< f >$. This rule is used to guarantee that the expression contains at least one of the required variables x and y.
- An additional term which modifies the resulting maximum value. This is created using derivation rule $< g >$.

In addition to the required variables x and y, we used some additional optional variables which measure the sum of all x_i, as well as its average and maximum and minimum values.

[2]The source code of our implementation is available at https://github.com/amin-vanya/PonyGE2_EMO

5 EXPERIMENTAL WORK

We used our proposed implementation to perform a series of experiments using different MOPs in the fitness function. First, we performed executions using only one MOP at a time. Then, using the information obtained from these results, we performed a second round of experiments adopting two MOPs instead of just one at a time.

5.1 Single MOP experiments

Given a MOP, we considered three different configurations using said MOP with 2, 3 and 5 objectives. This was done to try to avoid the overspecification of the scalarizing functions by considering different instances of the same MOP. However, there is no particular reason as to why 3 and 5 objectives were specifically chosen other than the increase in the number of objectives. We used each of the DTLZ(1-7) and WFG(1-9) benchmark MOPs, which results in 16 different setups. In each case, we set the number of generations to 75, the population size to 20, and the threshold to 15% of the maximum hypervolume. We performed two independent executions for each experimental setup, and we evaluated the performance of the 3 best performing scalarizing functions of each run, obtaining 6 different functions for each setup.

5.2 Combined MOPs experiments

The single MOPs experiments previously mentioned were used to measure how well each individual MOP used in the setup could generalize in solving all the benchmark tests. Using this information, which is discussed in the next section, we chose to perform further experiments combining two promising MOPs in the same execution. The pair of MOPs used were DTLZ3 with DTLZ4 and DTLZ7 with WFG1. In each of these cases, we also used 2, 3 and 5 objectives for each MOP, producing 6 instances to be solved in the fitness function. Additionaly, we increased the number of generations to 100, in an attempt of exploring more functions since now we are solving two MOPs instead of just one, but the population size (20) and threshold (15%) remained the same. We performed three independent executions using these two setups and we evaluated the performance of the three best resulting functions for each setup.

6 RESULTS

To assess the performance of the generated scalarizing functions we adopted once again modified versions of MOMBI-II using the new functions and we compared them against MOMBI-II using ASF, which is the default setup. The MOPs solved were 16 benchmark problems (DTLZ1-7 and WFG1-9), considering 2, 3, 4, 5, 6 and 7 objectives, creating a total of 96 instances. It is important to emphasize that during the generation of the scalarizing functions only one or two MOPs were adopted, and we are evaluating against the whole benchmark to verify their capability to generalize. We performed 30 independent runs per test instance for each scalarizing function. The obtained results were compared using the average hypervolume values and the Wilcoxon rank-sum test with a significance level of 5%.

6.1 Single MOP results

In Tables 1 to 4 we show the results obtained with each of the 16 benchmark MOPs separated by their geometry. In each of these Tables we show the training MOP used and each of the scalarizing functions generated, as well as the comparison of hypervolume values with respect to the use of ASF. The columns "+", "-" and "∼" present the number of MOPs where the new function outperformed ASF, got worse values than ASF or had a similar performance to ASF according to the Wilcoxon rank-sum test respectively.

For each MOP we present the comparison of 6 different scalarizing functions produced by 2 independent executions of our implementation. The best 3 functions found in the first execution are shown in the first three rows of the corresponding MOP, whereas the best 3 functions found in the second execution are shown in the next three rows. Thus, the first three scalarizing functions tend to share similarities in the function phenotype (for instance, functions SF_DTLZ1A and SF_DTLZ1B in Table 1 are different in only one digit), in the same way that the next three functions share some terms. However, this does not always reflect a similar tendency in the hypervolume comparison, since the change of a single term can have a significant impact in the performance of the function solving the benchmark MOPs used (for instance, functions SF_DTLZ2B and SF_DTLZ2C in Table 4 are also different in only one digit, but have a rather different performance).

Most of the functions obtained with linear MOPs such as DTLZ1 and WFG3 have a really poor performance, since the number of MOPs in which the results improve is much smaller than the number of MOPs in which the results worsens. From this set, the best performing function is SF_DTLZ1F, which improves 37 instances but obtains worse values in 20 instances.

There is only one fully convex MOP in both benchmarks, and it is WFG2. The functions generated with it don't show a much better performance, with the exception of SF_WFG2A, which improves 35 MOPs and only obtains worse values in 10.

Regarding mixed MOPs (which are MOPs in which the geometry of their Pareto fronts contain both convex and concave portions) we observe a really bad performance, since not a single resulting function is able to improve more MOPs than the ones in which it worsens.

The concave MOPs are the most frequently represented in both benchmarks and we can observe a more diverse behavior in the resulting functions. For instance, the function DF_DTLZ4A has a really good performance, being able to improve 40 instances and only worsening 2. On the other hand, two other functions generated with DTLZ4 (SF_DTLZ4B and SF_DTLZ4C) as well as SF_WFG5E have a performance identical to ASF, since all 96 instances have similar values.

Additionally, we performed a further analysis to know how many MOPs were being improved by each function with respect to the geometry of the benchmark MOPs. This was done by dividing the 96 test instances using the same classification of linear, convex, mixed and concave MOPs.

We averaged the values obtained with the scalarizing functions produced by each of the training MOPs. These results are shown using a percentage value, because there are different quantities of each geometry in the benchmark sets. Namely, there are 12 linear

Table 1: Scalarizing functions generated using linear benchmark MOPs (DTLZ1 and WFG3).

Training MOP	Scalarizing function	f(x,y)	HV comparison		
			+	-	~
DTLZ1	SF_DTLZ1A	max{x/y}+57.76-x_max	22	33	41
	SF_DTLZ1B	max{x/y}+57.73-x_max	20	34	42
	SF_DTLZ1C	max{x/y/x_sum}+x_avg*x_avg	6	80	10
	SF_DTLZ1D	max{sqrt(x/y)+x_avg}+x_sum	6	85	5
	SF_DTLZ1E	max{x/y}+sqrt(x_sum)	8	59	29
	SF_DTLZ1F	max{x/y}+0.58*(x_avg)	37	20	39
WFG3	SF_WFG3A	max{sqrt(x/y)}+0.67*(x_sum*x_sum*x_max +sqrt(x_sum*x_max))	0	96	0
	SF_WFG3B	max{sqrt(x/y)}+0.67*(x_sum*x_sum*x_max +sqrt(x_max))	0	96	0
	SF_WFG3C	max{x/y}*x_sum	2	86	8
	SF_WFG3D	max{x/y*sqrt(y)-y}+0.98*(x_sum)	5	86	5
	SF_WFG3E	max{x/y*sqrt(y)-y}+0.94*(x_sum)	5	86	5
	SF_WFG3F	max{x/y*sqrt(y)-y}+x_sum	5	86	5

Table 2: Scalarizing functions generated using a convex benchmark MOP (WFG2).

Training MOP	Scalarizing function	f(x,y)	HV comparison		
			+	-	~
WFG2	SF_WFG2A	max{x/y}*x_avg/x_sum-x_avg*x_sum+x_avg	14	76	6
	SF_WFG2B	max{x/y}+0.13*(x_max)	5	7	84
	SF_WFG2C	max{x/y}+x_sum	6	77	13
	SF_WFG2D	max{x/y}+0.67*(x_avg)	39	20	37
	SF_WFG2E	max{sqrt(x/y)}*63.13-x_avg	1	26	69
	SF_WFG2F	max{x/y}+0.66*(x_avg)	38	20	38

Table 3: Scalarizing functions generated using mixed benchmark MOPs (DTLZ7 and WFG1).

Training MOP	Scalarizing function	f(x,y)	HV comparison		
			+	-	~
DTLZ7	SF_DTLZ7A	max{x/y-y-y+x/y}+x_sum	14	71	11
	SF_DTLZ7B	max{x/y-y+x/y}+x_sum	14	68	14
	SF_DTLZ7C	max{x/y-y+sqrt(sqrt(x_max))}+x_avg	17	58	21
	SF_DTLZ7D	max{x/y-x+x/y}+x_sum	12	73	11
	SF_DTLZ7E	max{x/y-x_max*x+x/y}+x_sum	10	74	12
	SF_DTLZ7F	max{x/y-y}+x_sum*sqrt(x_avg/x_sum)	14	68	14
WFG1	SF_WFG1A	max{x/y-y/x_avg}+0.19*(x_avg)	5	87	4
	SF_WFG1B	max{20.99-y*x/y+x/y}+x_avg	12	74	10
	SF_WFG1C	max{x/y}+sqrt(x_sum/x_max)	18	69	9
	SF_WFG1D	max{x/y/y}+0.53*(sqrt(x_max)-x_sum *x_max*sqrt(x_avg)-x_max*x_sum)	22	70	4
	SF_WFG1E	max{x/y/y}+0.05*(x_avg/x_sum/x_max-x_max)	15	78	3
	SF_WFG1F	max{x/y/y}+0.30*(x_avg/80.06-x_max-x_avg -x_max)	14	79	3

instances (DTLZ1, WFG3), 6 convex instances (WFG2), 12 mixed instances (DTLZ7, WFG1) and 66 concave instances (DTLZ2-DTLZ6, WFG4-WFG9), since each MOP is solved using 2-7 objectives.

In Table 5 we present the average percentage of MOPs improved by the functions generated with each training MOP. We used a gradient color to show the number of MOPs improved, the more green a cell is, the greater improvement it represents. Similarly, in Table 6 we show the average percentage of MOPs worsened by the functions. However, in this case, the more red a cell is, the greater number of MOPs worsened it represents.

From these two tables we can observe that no single MOP is able to generate scalarizing functions which improve more than half of the MOPs according to the geometry classification used (the best improvements range from 44 to 47%). On the other hand, worsened problems do get high values, since some MOPs generated really bad

Table 4: Scalarizing functions generated using concave benchmark MOPs (DTLZ2-7, WFG4-9).

Training MOP	Scalarizing function	f(x,y)	HV comparison		
			+	-	~
DTLZ2	SF_DTLZ2A	max{x/y}+0.04*(x_sum)	35	10	51
	SF_DTLZ2B	max{x/y}+0.04*(sqrt(x_sum))	27	11	58
	SF_DTLZ2C	max{x/y}+0.84*(sqrt(x_sum))	13	51	32
	SF_DTLZ2D	max{34.44*x/y}+x_sum*x_sum	5	80	11
	SF_DTLZ2E	max{23.24*x/y}+x_max*x_sum	15	67	14
	SF_DTLZ2F	max{53.23*x/y}+x_sum	15	54	27
DTLZ3	SF_DTLZ3A	max{x/y}+0.03*(x_sum)	29	8	59
	SF_DTLZ3B	max{x/y}+0.07*(sqrt(x_sum)*sqrt(x_avg))	31	9	56
	SF_DTLZ3C	max{x/y}+0.07*(sqrt(x_avg))	26	10	60
	SF_DTLZ3D	max{x+x/y*24.64}+x_sum-x_avg	36	6	54
	SF_DTLZ3E	max{x/y*92.46}+x_sum	27	7	62
	SF_DTLZ3F	max{x/y*24.92}+x_avg	24	7	65
DTLZ4	SF_DTLZ4A	max{x/y-x_sum}+x_sum	40	2	54
	SF_DTLZ4B	max{x/y}*x_sum/x_avg	0	0	96
	SF_DTLZ4C	max{x/y}+x_sum/x_avg	0	0	96
	SF_DTLZ4D	max{x/y+x}+0.12*(x_avg*x_max)	55	21	20
	SF_DTLZ4E	max{x/y+x}+0.02*(x_sum)	55	19	22
	SF_DTLZ4F	max{x/y+x}+0.22*(x_sum)	53	27	16
DTLZ5	SF_DTLZ5A	max{x/y}+0.21*(x_max+x_avg)	32	17	47
	SF_DTLZ5B	max{x*y}+0.88*(x_sum/x_avg)	17	59	20
	SF_DTLZ5C	max{y*x}+0.06*(x_sum*x_sum/13.53)	15	69	12
	SF_DTLZ5D	max{y*x}+0.05*(x_sum*x_sum/13.53)	18	67	11
	SF_DTLZ5E	max{y+x}+0.76*(x_sum*x_avg/13.63)	13	73	10
	SF_DTLZ5F	max{y+y*x-y}+0.01*(x_sum*x_max)	17	69	10
DTLZ6	SF_DTLZ6A	max{x*x/y}+x_sum*x_max	35	52	9
	SF_DTLZ6B	max{x*x/y}+x_sum*x_avg	41	51	4
	SF_DTLZ6C	max{x/y+05.97*x-x_max}+sqrt(x_avg)	26	63	7
	SF_DTLZ6D	max{x+y}+0.04*(x_sum)	10	76	10
	SF_DTLZ6E	max{x+y}+0.04*(x_sum*x_max)	11	77	8
	SF_DTLZ6F	max{x+y}+0.54*(x_avg/x_sum)	12	76	8
WFG4	SF_WFG4A	max{x/y*11.03}+x_sum+71.03	36	15	45
	SF_WFG4B	max{x/y*16.72}+x_max	2	5	89
	SF_WFG4C	max{x/y*x/y}+x_max	10	12	74
	SF_WFG4D	max{x/y}+x_sum/02.81*x_sum-x_sum*x_avg	24	43	29
	SF_WFG4E	max{x/y}+0.29*(x_avg)	34	18	44
	SF_WFG4F	max{x/y}+0.35*(x_sum)	12	46	38
WFG5	SF_WFG5A	max{x_max-x_max+x/y-sqrt(x)}+x_max	15	67	14
	SF_WFG5B	max{x_max+x/y-x}+0.44*(74.97)	11	72	13
	SF_WFG5C	max{x_max+x/y-x}+x_avg	14	70	12
	SF_WFG5D	max{x/y+44.77-x_sum-y}+x_sum	16	65	15
	SF_WFG5E	max{x/y}+sqrt(41.14)	0	0	96
	SF_WFG5F	max{x/y}+sqrt(x_avg)	29	24	43
WFG6	SF_WFG6A	max{x/y-y*x*x_avg}+0.43 *(x_sum-sqrt(x_avg)/x_max)	29	37	30
	SF_WFG6B	max{x/y-y*x*y}+0.43*(x_max)	15	54	27
	SF_WFG6C	max{x/y}+0.82*(x_max+sqrt(x_max *x_sum*x_max))	44	35	17
	SF_WFG6D	max{x/y-y}+0.03*(x_avg*sqrt(sqrt(x_avg*x_sum))+sqrt(sqrt(67.19))*x_avg)	19	59	18
	SF_WFG6E	max{x/y-y}+0.03*(x_avg*x_sum)	17	60	19
	SF_WFG6F	max{x/y-y}+0.04*(x_avg)	17	64	15
WFG7	SF_WFG7A	max{x/y}+x_max/sqrt(x_max+x_avg)/x_sum +x_sum*x_max	44	32	20
	SF_WFG7B	max{x/y}+sqrt(sqrt(x_max+x_avg) /x_sum+x_sum)/x_max*x_max	44	6	46
	SF_WFG7C	max{x/y}*sqrt(sqrt(x_max+x_avg)/x_sum +x_sum)/x_max*x_max	37	18	41
	SF_WFG7D	max{x/y*x/y-x/y}+0.88*(sqrt(sqrt(sqrt(sqrt(x_avg))))-x_max*x_max+sqrt(x_sum))	23	54	19
	SF_WFG7E	max{x/y*x/y-x/y}+0.84*(sqrt(sqrt(sqrt(sqrt(x_sum)))))/x_max)	11	55	30
	SF_WFG7F	max{x/y*x/y-x/y}+0.84*(sqrt(sqrt(sqrt(sqrt(x_sum))))-x_max)	9	49	38
WFG8	SF_WFG8A	max{x_sum*sqrt(x/y)/x_max+03.05*x/y}+69.03 -x_avg	19	65	12
	SF_WFG8B	max{x_sum*sqrt(x/y)/x_max+05.65*x/y} +x_max	33	42	21
	SF_WFG8C	max{x/y/x_avg}+0.96*(x_max*x_max)	11	78	7
	SF_WFG8D	max{x/y-sqrt(x/y)}+72.28-sqrt(x_max)	2	67	27
	SF_WFG8E	max{x/y-x_avg}+0.82*(x_sum)	9	71	16
	SF_WFG8F	max{x/y-x_avg}+0.22*(sqrt(72.21-sqrt(x_max)))	6	69	21
WFG9	SF_WFG9A	max{x/y-y}+0.27*(sqrt(x_avg))	18	51	27
	SF_WFG9B	max{x/y-y}*04.90+sqrt(x_sum)	18	53	25
	SF_WFG9C	max{x/y}+0.34*(x_sum)	15	48	33
	SF_WFG9D	max{x/y}+0.42*(sqrt(x_max*x_avg))	36	19	41
	SF_WFG9E	max{x/y}+0.42*(sqrt(x_sum))	34	20	42
	SF_WFG9F	max{x/y}+0.40*(sqrt(x_sum))	34	20	42

Table 5: Average percentage of MOPs improved by geometry using single MOP functions

Training MOP	Geometry	Average MOPs improved (%)			
		Linear	Convex	Mixed	Concave
DTLZ1	Linear	25.00	19.45	13.89	16.16
WFG3		6.95	0.00	0.00	3.03
WFG2	Convex	13.89	22.22	19.45	17.93
DTLZ7	Mixed	44.45	33.34	30.56	3.79
WFG1		40.28	47.22	27.78	5.05
DTLZ2	Concave	4.17	8.33	27.78	21.21
DTLZ3		0.00	8.34	47.23	34.34
DTLZ4		11.12	11.11	12.50	45.96
DTLZ5		20.00	16.67	30.00	18.18
DTLZ6		1.39	0.00	4.17	33.08
WFG4		1.39	13.89	27.78	23.23
WFG5		30.56	30.56	30.56	7.58
WFG6		34.73	30.55	43.06	18.69
WFG7		13.89	5.56	20.84	35.61
WFG8		31.95	16.67	9.73	11.12
WFG9		18.06	30.56	36.11	26.52

Table 6: Average percentage of MOPs worsened by geometry using single MOPs functions

Training MOP	Geometry	Average MOPs worsened (%)			
		Linear	Convex	Mixed	Concave
DTLZ1	Linear	27.78	47.22	54.17	59.34
WFG3		79.17	91.67	94.45	95.45
WFG2	Convex	20.84	41.67	31.95	43.69
DTLZ7	Mixed	38.89	33.33	38.89	86.87
WFG1		54.17	44.45	52.78	91.92
DTLZ2	Concave	26.39	63.89	29.17	53.03
DTLZ3		4.17	36.11	0.00	7.83
DTLZ4		33.34	36.11	22.23	4.04
DTLZ5		53.33	56.67	45.00	63.33
DTLZ6		90.28	91.67	95.84	57.58
WFG4		15.28	47.22	9.73	26.26
WFG5		33.34	36.11	20.84	62.12
WFG6		44.45	55.56	29.17	59.60
WFG7		51.39	63.89	37.50	32.07
WFG8		33.34	69.44	59.72	75.76
WFG9		20.84	52.78	15.28	41.92

scalarizing functions which worsen almost all test instances. For instance, functions generated with DTLZ6 and WFG3 have little to no improvement in most cases, while having a really high number of MOPs worsened. This is an indication that these MOPs tend to generate overspecified scalarizing functions. Hence, if we are looking for scalarizing functions that generalize a good performance in other MOPs, we are interested in functions with a relatively good amount of MOPs improved while having a low amount of MOPs worsened. For example, altough WFG3 (a linear MOP) has a poor performance, DTLZ1 (the other linear MOP in the benchmarks used) generated functions which have a better performance (specially in linear MOPs, which is intuitively expected), and a smaller number of MOPs worsened overall.

Regarding WFG2, the functions generated have a similar improvement across all geometries, having the best improvement in the corresponding geometry of WFG2 (convex). However, the amount of problems worsened is always greater than the amount of problems improved, which is an expected behavior since WFG2 is the only fully convex problem in the whole benchmark.

Functions generated with mixed MOPs (both DTLZ7 and WFG1) are the ones which have the largest number of linear and convex MOPs improved, while improving also some of the mixed MOPs. However, they have some of the worst number of concave MOPs worsened, which may indicate that the functions that improve the linear, convex and mixed geometries tend to perform badly in concave MOPs.

Finally, once again, there are different behaviors in the functions generated with concave MOPs. On the one hand, DTLZ4 is the MOP which produced the functions that have the largest concave MOPs improvement, and it has a low improvement in linear, convex and mixed MOPs. Its counterpart could be the functions generated with WFG5, which have one of the lowest improvement in concave MOPs but have a decent improvement in linear, convex and mixed MOPs.

6.2 Combined MOPs results

Using the information from single MOP results, we decided to perform more experiments combining functions. First, we combined DTLZ3 and DTLZ4, since they are two MOPs which generated some of the best results in concave MOPs. Functions generated with WFG7 actually have the second best concave results, however, the number of MOPs worsened in all geometries is larger than those of DTLZ3. The goal of this experiment is to find a function that has a good performance in concave MOPs. Additionally, we combined DTLZ7 and WFG1, since they are the MOPs that generated the functions which have the best improvements across all three remaining geometries (linear, convex and mixed). Consequently, the goal of this experiment is to find a function that has a good performance in these MOPs.

The three resulting functions from combining DTLZ3 and DTLZ4 are SF_D3D4A-C, and the three resulting functions from combining DTLZ7 and WFG1 are SF_D7W1A-C. These are shown in Table 7, along with the number of MOPs improved (+), worsened (-) and similar (~) comparing against ASF. Here, we can observe that all three functions generated with DTLZ3 and DTLZ4 improve more test instances than the amount of instances worsened. While the exact opposite occurs with all three functions generated with DTLZ7 and WFG1. This is due to the fact that the benchmark MOPs used are mostly concave MOPs (such as DTLZ3 and DTLZ4).

This is backed up by the percentage of MOPs improved by each function according to the MOPs geometry shown in Table 8, and the percentage of MOPs worsened shown in Table 9.

From these results we can observe that, from the functions generated using DTLZ3 and DTLZ4, SF_D3D4A has the greatest improvement in concave MOPs, at the expense of worsening almost

Table 7: Scalarizing functions generated using combination of DTLZ3 with DTLZ4 and DTLZ7 with WFG1.

Scalarizing function	f(x,y)	HV comparison		
		+	-	~
SF_D3D4A	max{x/y+x}+0.06*x_avg	49	22	25
SF_D3D4B	max{x/y+x/y}+0.19*x_avg	30	5	61
SF_D3D4C	max{x/y+x/y}+x_avg/92.83	13	2	81
SF_D7W1A	max{x/y-x}-0.13*x_avg*x_max	34	57	5
SF_D7W1B	max{x/y-x}-0.13*x_sum	28	55	13
SF_D7W1C	max{x/y-x}+89.95-sqrt(x_max)/x_max	15	71	10

Table 8: Percentage of MOPs improved by geometry using combined MOPs functions.

Scalarizing function	Training Geometry	MOPs improved (%)			
		Linear	Convex	Mixed	Concave
SF_D3D4A	Concave	8.34	0.00	8.34	71.21
SF_D3D4B		0.00	16.67	41.67	36.36
SF_D3D4C		0.00	0.00	33.34	13.64
SF_D7W1A	Mixed	50.00	83.33	50.00	25.76
SF_D7W1B		50.00	50.00	58.33	18.18
SF_D7W1C		33.34	50.00	41.67	4.55

Table 9: Percentage of MOPs worsened by geometry using combined MOPs functions.

Scalarizing function	Training Geometry	MOPs worsened (%)			
		Linear	Convex	Mixed	Concave
SF_D3D4A	Concave	67.00	83.00	42.00	6.00
SF_D3D4B		0.00	17.00	0.00	6.00
SF_D3D4C		0.00	0.00	0.00	3.00
SF_D7W1A	Mixed	50.00	17.00	17.00	73.00
SF_D7W1B		33.00	0.00	25.00	73.00
SF_D7W1C		67.00	50.00	25.00	86.00

all convex instances, and a lot of linear and mixed MOPs. Interestingly, SF_D3D4B, which is the function with the second best improvement in concave MOPs, has a relatively low worsening in all geometries while still having a decent improvement in mixed MOPs. This may be due to the fact that functions generated with DTLZ3 have the highest improvement of mixed MOPs in Table 5, which is also interesting since DTLZ3 is a degenerate MOP and this property could be playing a role in the amount of problems improved. The last function generated with DTLZ3 and DTLZ4, SF_D3D4C, has a very similar behavior as SF_D3D4B, but with lower improvement and worsening values.

Regarding the functions generated using DTLZ7 and WFG1, we can notice that SF_D7W1A is the one with the greatest improvement values in linear, convex and mixed MOPs. And it even improves 25% of the concave MOPs. However, it worsens half of the linear instances. All three functions generated in these experiments have a similar behavior in improving linear, convex and mixed MOPs at the expense of worsening a lot (73-86%) of the concave MOPs. It is interesting to notice that two of the three resulting functions generated using this combination improve more concave MOPs than the average improvement shown by DTLZ7 and WFG1 on their own in Table 5.

Finally, we show the average execution time of the experiments using our implementation in Figure 1. The hardware used to execute these experiments consists of an Intel Core i7-8700 CPU, (6 cores, 3.20 GHz) with 16 GB of RAM. The differences in execution time correspond to the cost of solving each MOP during the GE fitness function. Hence, the relatively easy MOPs such as DTLZ1 has one of the smallest execution times while WFG1, which is more time-consuming has the worst execution time in single MOP experiments. Also, the amount of functions that get through step 3 of the fitness function into step 4 has a great impact in the execution time, since step 4 involves the solution of the training MOP with a large number of function evaluations. As can be seen, all the execution times shown are relatively high, ranging from 3.8 hrs up to 13.3 hrs in the worst case, without using any sort of parallelisation or multithreading optimisation. However, it must be noted that it is not our intention to generate a custom scalarizing function in order to solve a given MOP, but rather to find scalarizing functions which can outperform other commonly used functions in at least some types of MOPs.

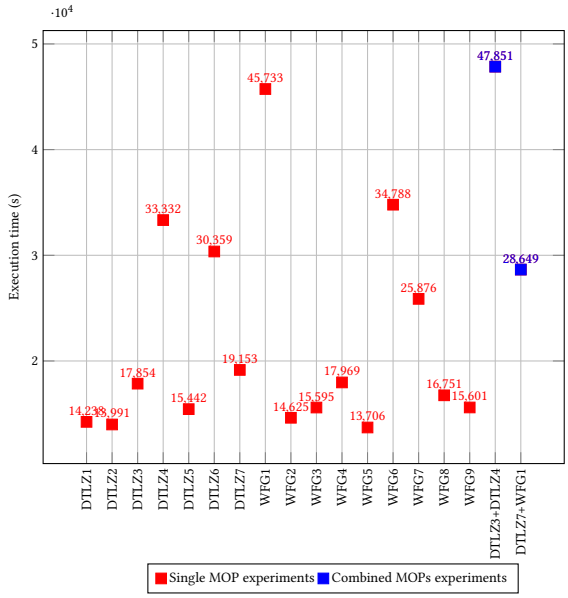

Figure 1: Average execution times of single and combined MOPs experiments.

7 CONCLUSIONS AND FUTURE WORK

In this work, we have presented an implementation that allows the generation of scalarizing functions given one or several MOPs.

In the first round of experiments presented, the experimental validation shows mixed results, since some functions are able to improve some instances compared to ASF, but at the expense of worsening several other instances. However, we have shown a brief analysis of the behavior of these scalarizing functions, considering the geometries of the MOPs adopted to validate their performance.

Using this information, we have designed a second round of experiments using our implementation, but this time, combining two MOPs that exhibit an average improvement in certain geometries. The corresponding results, along with their geometries analysis show that these scalarizing functions exhibit a consistent behavior with our previous experiments. Meaning that the use of DTLZ3 and DTLZ4 (which had good improvements in concave MOPs) in the GE search process created a scalarizing function with an even greater improvement in concave MOPs. Whereas the use of DTLZ7 and WFG1 (which had good improvements in linear, convex and/or mixed MOPs) produced functions with a decent performance in all three geometries. Hence, we could identify which MOPs allow the generation of better functions to solve a certain type of MOPs.

Another expected behavior that we could observe in most of our results is that when the best results are obtained in a given geometry (or set of geometries, such as linear, convex and mixed), the worst results will obtained in the remaining cases. This makes sense, since the better a function solves a particular type of MOP, the more specific the scalarizing function is and, therefore, it will perform the worse in MOPs that are different to the one being successfully solved. Hence, we can conclude that the combination of two MOPs with similar characteristics (such as geometry of the Pareto front) in our proposed implementation allows us to find scalarizing functions with a better performance in the MOPs with said characteristics, at the expense of potentially worsening MOPs with different characteristics. Thus, there is a trade off in the MOPs improved/worsened with the resulting functions, but we should still be able to generate different functions which work well for each of the geometries present in the benchmark tests. This could be particularly useful since, even though we may not know the geometry of a given MOP in a real-world application, we could use an ensemble of scalarizing functions which work well in different geometries, and develop a decomposition-based MOEA which alternates the scalarizing function used according to the performance of each of them in solving the given MOP.

Some of the future work derived from the results we have presented here include a deeper analysis of the behavior of the generated functions, as well as more experiments considering different or additional combinations of the MOPs used. Another aspect that may be worth exploring is to change the indicator used to evaluate the performance of the scalarizing functions in the fitness function of our GE. We adopted hypervolume since it assesses both convergence and maximum spread of the Pareto front. However, we could employ a different performance indicator in order to specifically improve, for example, diversity in the solutions. Also, a different set of MOPs could be used, such as inverse DTLZ MOPs [10], since regular DTLZ and WFG are composed by concave MOPs in their majority, and other geometries are underrepresented, causing some differences in the amount of experimental validation done for each geometry.

ACKNOWLEDGMENTS

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

REFERENCES

[1] Jürgen Branke and Kalyanmoy Deb. 2005. Integrating User Preferences into Evolutionary Multi-Objective Optimization. In *Knowledge Incorporation in Evolutionary Computation*, Yaochu Jin (Ed.). Springer, Berlin Heidelberg, 461–477. ISBN 3-540-22902-7.

[2] William La Cava, Thomas Helmuth, Lee Spector, and Kourosh Danai. 2015. Genetic Programming with Epigenetic Local Search. In *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO 15*. ACM Press. https://doi.org/10.1145/2739480.2754763

[3] Carlos A. Coello Coello. 1999. A Comprehensive Survey of Evolutionary-Based Multiobjective Optimization Techniques. *Knowledge and Information Systems* 1, 3 (Aug. 1999), 269–308. https://doi.org/10.1007/bf03325101

[4] Indraneel Das and J.E. Dennis. 1998. Normal-Boundary Intersection: A New Method for Generating the Pareto Surface in Nonlinear Multicriteria Optimization Problems. *SIAM Journal on Optimization* 8, 3 (1998), 631–657.

[5] Kalyanmoy Deb, Lothar Thiele, Marco Laumanns, and Eckart Zitzler. 2005. Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary multiobjective optimization*. Springer, 105–145.

[6] Michael Fenton, James McDermott, David Fagan, Stefan Forstenlechner, Michael O'Neill, and Erik Hemberg. 2017. PonyGE2: Grammatical Evolution in Python. (2017). https://doi.org/10.48550/ARXIV.1703.08535

[7] Raquel Hernández Gómez and Carlos A. Coello Coello. 2015. 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. https://doi.org/10.1145/2739480.2754776

[8] Raquel Hernández Gómez and Carlos A. Coello Coello. 2017. A Hyper-Heuristic of Scalarizing Functions. In *2017 Genetic and Evolutionary Computation Conference (GECCO'2017)*. ACM Press, Berlin, Germany, 577 –584. ISBN 978-1-4503-4920-8..

[9] Simon Huband, Luigi Barone, Lyndon While, and Phil Hingston. 2005. A Scalable Multi-objective Test Problem Toolkit. In *Evolutionary Multi-Criterion Optimization*, Carlos A. Coello Coello, Arturo Hernández Aguirre, and Eckart Zitzler (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 280–295.

[10] Himanshu Jain and Kalyanmoy Deb. 2014. 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 (Aug. 2014), 602–622. https://doi.org/10.1109/tevc.2013.2281534

[11] John R. Koza. 1989. Hierarchical Genetic Algorithms Operating on Populations of Computer Programs.. In *IJCAI*, Vol. 89. Springer-Verlag, 768–774.

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

[13] Miriam Pescador-Rojas, Raquel Hernández Gómez, Elizabeth Montero, Nicolás Rojas-Morales, María-Cristina Riff, and Carlos A. Coello Coello. 2017. An Overview of Weighted and Unconstrained Scalarizing Functions. In *Evolutionary Multi-Criterion Optimization, 9th International Conference, EMO 2017*, Heike Trautmann, Günter Rudolph, Kathrin Klamroth, Oliver Schütze, Margaret Wiecek, Yaochu Jin, and Christian Grimme (Eds.). Springer. Lecture Notes in Computer Science Vol. 10173, Münster, Germany, 499–513. ISBN 978-3-319-54156-3.

[14] Amín V. Bernabé Rodríguez and Carlos A. Coello Coello. 2020. Generation of New Scalarizing Functions Using Genetic Programming. In *Parallel Problem Solving from Nature – PPSN XVI*. Springer International Publishing, 3–17. https://doi.org/10.1007/978-3-030-58115-2_1

[15] Theodor Stewart, Oliver Bandte, Heinrich Braun, Nirupam Chakraborti, Matthias Ehrgott, Mathias Göbelt, Yaochu Jin, Hirotaka Nakayama, Silvia Poles, and Danilo Di Stefano. 2008. Real-World Applications of Multiobjective Optimization. In *Multiobjective Optimization*. Springer Berlin Heidelberg, 285–327. https://doi.org/10.1007/978-3-540-88908-3_11

[16] Qian Xu, Zhanqi Xu, and Tao Ma. 2020. A Survey of Multiobjective Evolutionary Algorithms Based on Decomposition: Variants, Challenges and Future Directions. *IEEE Access* 8 (2020), 41588–41614. https://doi.org/10.1109/access.2020.2973670

[17] Qingfu Zhang and Hui Li. 2007. MOEA/D: A Multiobjective Evolutionary Algorithm Based on Decomposition. *IEEE Transactions on Evolutionary Computation* 11, 6 (December 2007), 712–731.

[18] Eckart Zitzler. 1999. *Evolutionary algorithms for multiobjective optimization: methods and applications*. Ph. D. Dissertation. Swiss Federal Institute of Technology (ETH).

[19] Eckart Zitzler and Simon Künzli. 2004. Indicator-based Selection in Multiobjective Search. In *Parallel Problem Solving from Nature - PPSN VIII*, Xin Yao et al. (Ed.). Springer-Verlag. Lecture Notes in Computer Science Vol. 3242, Birmingham, UK, 832–842.