Utrecht University

Department of Information and Computing Sciences

Master Thesis in Artificial Intelligence

The influence of using Adaptive Operator Selection in a Multiobjective Evolutionary Algorithm Based on Decomposition

Author
Jordi Verheul (4297369)

First Supervisor
Dr. Ir. D. Thierens

Second Supervisor
Dr. N. A. Alechina

June 2020

Abstract

Recently, a new algorithm has arisen for solving Multiobjective Optimization Problems (MOPs), called Multiobjective Evolutionary Algorithms based on Decomposition (MOEA/D). This algorithm decomposes a MOP into a number of single optimization subproblems and optimizes them simultaneously using an evolutionary process. Adaptive Operator Selection (AOS) mechanisms are added to MOEA/D to improve this evolutionary process by choosing the right mutation operators from a pool to be applied at the reproduction phase. This thesis provides an overview of existing MOEAs, AOS mechanisms and the combination of these two. Also, a new AOS mechanism is proposed. It combines a fitness-rate-rank based (FRR) credit assignment from existing research with a probability based operator selection mechanism that uses tournament selection. Unlike existing AOS methods using FRR, the selection probabilities of the operators are solely based on the relative order of the estimated rewards. The purpose of this is to improve the balance between the exploration and exploitation of the operators, and therefore improve the performance. This AOS is then used within the MOEA/D framework to form a new algorithm: MOEA/D-FRR-TS. Several experiments were conducted to test the performance of this algorithm. Results of these experiments show that there is no significant difference between MOEA/D-FRR-TS and other probability based AOS + MOEA/D combinations. Furthermore, it is shown that MOEA/D-FRR-TS using a pool of operators can improve MOEA/D that uses a single operator.

Abbreviations

AOS Adaptive Operator Selection

AP Adaptive Pursuit

AUC Area Under Curve

DE Differential Evolution

DMAB Dynamic Multi-Armed Bandit

EA Evolutionary Algorithm

EVB Extreme Value Based

FIR Fitness Improvement Rates

FRR Fitness-Rate-Ranked Based

GD Generational Distance

HV Hypervolume

IBEA Indicator-Based Evolutionary Algorithm

IGD Inverted Generational Distance

MAB Multi-Armed Bandit

MOEA Multi-Objective Evolutionary Algorithm

MOEA/D MOEA based on Decomposition

MOEA/D-DRA MOEA/D with Dynamical Resource Allocation

MOP Multi-objective Optimization Problem

PF Pareto front

PM Probability Matching

PS Pareto set

SADE Self-Adaptive Differential Evolution

SlMAB Sliding Multi-Armed Bandit

SOP Single-objective Optimization Problem

SR Sum of Ranks

TS Tournament Selection

UCB Upper Confidence Bound

Contents

1 Introduction

Evolutionary Algorithms (EAs) are known for being able to find near-optimal solutions in hard optimization problems, which are problems with a very large solution space, and their ability to be applied in a wide range of fields. Due to the exponential increase of computational power, EAs can become even more useful, since they will be able to perform the same amount of work in less time. The idea behind EAs is to mimic the evolution which we know from biology. It uses the same stochastic mechanisms such as reproduction, mutation, crossover and selection. An EA starts off with a random initial population of solutions. Then it will loop through the following steps, which together are called a generation, until some stopping criteria is met:

1. **Selection**: the x best solutions from the population are selected (the parents) according to its fitness
2. **Reproduction**: these parents will then be mutated and recombined using variation operators to create new solutions (the offspring)
3. **Evaluation**: the fitness of the offspring is computed

The solutions in the population at the time that the algorithm has terminated are the best solutions found so far, which are either optimal or near-optimal, provided that the algorithm has run long enough. [1] gives an example of an optimization problem that can be solved using an EA. As you can see, the problem contains a single objective function that needs to be optimized. In this case, this function needs to be minimized.

Besides the use of EAs in single-objective optimization problems (SOPs), where only one objective has to be maximized or minimized, are EAs also applied in the context of multiobjective optimization problems (MOPs). These algorithms are called multiobjective evolutionary algorithms (MOEAs). In a MOP, the problem is defined by multiple objective functions, which need to be optimized simultaneously. Since we don't know the weights to these objectives, we can not assign a single value (a fitness) to a solution, whereas this is the case for SOPs. We can therefore also not compare the solutions easily by a single value. Here, we can only state that solution $x = (x_1, \ldots, x_n)$ is better than solution $y = (y_1, \ldots, y_n)$ if (assuming that the problem is a minimization problem):

1. $f_i(x) \leq f_i(y)\ \forall i \in \{1, 2, ..., m\}$, *and*

2. $f_j(x) < f_j(y)$ *for at least one* $j \in \{1, 2, ..., m\}$

where f_i is the ith objective function. In this case, we say that solution x *dominates* solution y. A solution that is not dominated by any other solution in the objective space is called *Pareto optimal*. The corresponding vector containing all objective values belonging to this solution is the *Pareto optimal vector*. The set of all Pareto optimal solutions is called the *Pareto set* (PS) and the set of all Pareto optimal vectors is called the *Pareto front* (PF). All solutions in the PS are not absolutely better than any other solution in the set, which makes all of them acceptable.

Since in most MOPs, the objectives are contradictory, we can not find a single solution that minimizes (or maximizes in a maximization problem) all objectives, i.e. a single optimal solution. However, we can try to approximate the PF. This is exactly what a MOEA does. By evolving the solutions through many generations, the MOEA would hopefully find multiple non-dominated solutions as close to the pareto set as possible. After the pareto set is found, a decision maker chooses a final solution from this set. Important to note is that an MOEA has three goals when approximating the pareto set [2]:

1. Minimizing the distance between the approximated pareto front and the real pareto front (also known as convergence)

2. Maximizing the quality of distribution of the approximated pareto set (also known as diversity)

3. Maximizing the size of the approximated pareto set (also known as coverage)

Many versions of MOEAs have been proposed and investigated through the years. All these different implementations are using variation operators to create a more diverse offspring. Although these MOEAs show promising results, there is a downside. First of all, they often only use one type of variation operator in their algorithm. However, one operator might be better for one problem and another for another problem. So, it is probably better to choose amongst a set of operators and then use the one that performs the best for the problem we are optimizing. However, this would be a sufficient method if 1) we are sure that the operator that is used for a problem really is the best one to use, and 2) the environment is stationary, meaning that

the reward distribution[1] of the operators is the same at each moment of the optimization process and does not change. The former would be possible if there is enough training to learn how every operator performs, however this can be very computationally expensive. The latter is something we can't assume, as mentioned by [3]. An operator might be the best at one point of the evolutionary process, but another might be better later on. So, even if we have learned what the best operator for a particular problem is, it is not robust against changes in the environment in terms of changing operator rewards. A solution to this is the integration of an Adaptive Operator Selection (AOS) mechanism into MOEA. An AOS mechanism captures the performance of the different operators during the process and keeps track of it. Based on the history of the operators' performances, the AOS can make a more proper decision on what operator should be applied at a particular moment.

1.1 Goals

The first goal of this paper is to give an overview of what MOEAs and AOS mechanisms exists, as well as the combination of these two. Secondly, a new AOS mechanism is proposed. Unlike existing mechanisms, it uses tournament selection to select operators, which is solely based on the relative order of the estimated rewards. The purpose of this is to improve the balance between exploration and exploitation of operators, and therefore improve the performance as well. The new AOS algorithm is combined with a special version of MOEA, called MOEA based on Decomposition (MOEA/D). An experimental study is performed to assess the quality of the new algorithm.

1.2 Outline

This paper will be structured as follows. In Section 2, more background is given about the subject as well as related work. Here, we will elaborate on MOPs and how different types of MOEAs can be used to solve them. This is followed by an extensive explanation of a special kind of MOEA that is based on Decomposition, i.e. MOEA/D. Lastly, it explains what AOS consists of and how it is combined with MOEA/D. Then in Section 3, a new algorithm is proposed and an overview is given of the experimental studies that are

[1]The operator reward distribution represents the performance of the operators.

performed. This is followed by an explanation of the results in Section 4. Finally, Section 5 discusses the results, together with future work and a conclusion of this thesis.

2 Background and related work

In this section, we will look at previous work performed by other researchers that can be interesting for this paper. Additionally, more background knowledge is given about the theory that is used throughout this paper.

2.1 Multiobjective Optimization Problem

In order to talk about different methods and algorithms that are being used to solve a Multiobjective Optimization Problem (MOP) in this paper and in other relevant research, it is important to understand what a MOP is exactly. A MOP consists of multiple conflicting objective functions that need to be optimized. It will either be minimized or maximized, depending on the problem. We can define a MOP as follows [4]:

$$\begin{aligned} &\text{maximize/minimize } F(x) = (f_1(x), f_2(x), ..., f_m(x)) \\ &\text{subject to } x \in \Omega \end{aligned} \tag{1}$$

where Ω is the decision space, $\mathrm{x} = (x_i, ..., x_n) \in \Omega$ is the decision vector and $f_i(x)$ the objective function corresponding to the ith objective. Additionally, the problem may contain a number of constraints.

As [5] explains, in a single-objective optimization problem (SOP) the goal is to find the best solution by looking at the objective function that correspond to the problem. Since this objective function has a single value, we can just maximize or minimize this function to determine the best solution. However, in MOPs it is not that trivial. Since the problems consists of multiple (probably) conflicting objectives, we cannot talk about a single optimal solution. Instead, we can define a Pareto set as mentioned before, containing all the non-dominated solutions.

2.1.1 Solving a MOP

To find this Pareto set, or at least approximate it, an algorithm needs to be applied. There are several ways of doing this. Most of them are solved using evolutionary algorithms, i.e. MOEAs. Nevertheless, there are methods that use other principles. First, we will take a look at the latter.

2.1.2 Non-MOEA based methods

One method that can solve a MOP is with the use of Simulated Annealing (SA) [6]. SA is an improvement upon local search which aims at avoiding ending up in local optima. When local search is applied to a MOP, it keeps track of the Pareto set containing all non-dominated solutions so far. It starts with an initial solution and can already place this in the Pareto set, since it is empty and is therefore not dominated by a solution in the set. It will then create a new solution in the neighborhood of the current solution. If the solution is an improvement, i.e. it dominates the current solution, it will replace the current solution. It will also replace that solution in the Pareto set, since it dominates. If we continue this procedure, the Pareto set will always end up with only one solution after the algorithm has terminated, which is the local optimum. This is where SA does the trick: instead of always rejecting a worse solution from the neighborhood, it sometimes accepts it. This allows the algorithm to escape the local optimum and find new ones, possibly new pareto optimal solutions. This way, the full solution space is explored which leads to a better approximation of the Pareto set. A number of different implementations where SA is used in multiobjective optimization algorithms is proposed in [6].

Another non-MOEA based method is Particle Swarm Optimization (PSO) [7]. In PSO, a population of so called *particles* is used to approximate the Pareto set. All these particles have a direction in which they move. It has some overlap with MOEAs in the way that it uses a population of individuals (in this context called particles) whose internal values are updated iteratively. One main difference is that PSO does not select the best particles to create the next offspring with. Instead, the particles are updated by following the so called "leaders". The amount of exploration is hereby dependent on difference in direction between the leaders and the particular particle. PSO has become popular due to its simplicity and applicability in a wide range of fields. [7] gives a comprehensive classification of different PSO-based algorithms for MOPs (MOP-SOs) and corresponding examples.

2.1.3 MOEAs

Despite the existence of some methods that are not based on evolutionary algorithms as described in the previous section, most methods for solving MOPs are based on Multi-Objective Evolutionary Algorithms (MOEAs).

These algorithms use evolutionary algorithms in the optimization process.

The first concept of MOEA was brought up by Schaffer [8] who found out that genetic algorithms (GAs), a specific type of EA, can help to optimize a MOP. He implemented this idea into a software system called Vector Evaluated Genetic Algorithm (VEGA). The idea behind this algorithm is the same as every other EA: the fittest survive. The best solutions move on to the next generation and will generate offspring. However, Schaffer already pointed out a potential problem. Since the selection was done based on dominance, solutions that excel in one objective function have a higher chance of surviving and end as part of the approximated Pareto set. Solutions which have values that are less outstanding (but still above average perhaps) might not survive, but could actually be more of interest. This would be less of a problem if a utopian solution[2] exists, however most of the time that is not the case.

Partially because of this work, the interest in MOEA grew. Many MOEAs were invented, including newer version of old algorithms. [2] categorizes MOEAs into three classes: *Domination-based*, *Indicator-based* and *Decomposition-based*.

2.1.4 Domination-based MOEAs

To determine which solutions may proceed to the next generation and may be used to generate offspring, a selection mechanism is needed. To make these selections, it should be able to compare solutions with each other. A domination-based MOEA does this be using the pareto dominance relationship we have talked about in the introduction. A solution will then continue if it dominates its parent.

A problem regarding MOEAs that compare solutions based on dominance, is that two solutions that do not dominate each other are incomparable, in principle. In a problem with many objectives, this would happen very often, leading to many solutions that are evenly "good". Around the year 2000, several of these MOEAs were designed with techniques to reduce this problem. The most well-known and popular were PAES [9], SPEA2 [10] and NSGA-II [11]. When comparing two solutions with each other and determining which is the best, it does not only look at just both solutions, but also uses dominance information from solutions of the population. This

[2] A utopian solution is a solution where all objective values are better than the objective values of any other solution in the solution space

way, the set of all solutions witness a relation that is closer to a total order [3]. SPEA2 is based on its predecessor, SPEA. SPEA introduced the concept of an archive, which keeps track of the nondominated solutions, i.e. the Pareto set. This archive can be seen as an external population, besides the regular population of the EA. In SPEA2, the individuals (solutions) are assigned a fitness based on both the solutions it dominates and the solutions by which it is dominated.

PAES also wants to minimize the amount of incomparable solutions. In PAES, new solutions (the offspring) are generated by mutating current solutions (the parents), also known as local search. Here, if the offspring is an improvement, i.e. if it dominates its parent, it replaces the parent. However, if they are not dominated by each other, the offspring will be compared with the non-dominated solutions from the archive. If it dominates a solution from this archive, it will become the new parent after all. If not, it still has the opportunity to become the parent if it resides in a less crowded region of the objective space than its parent does. This way, PEAS is always able to create a total order of the solutions.

NSGA-II uses another technique to be able to compare solutions that do not dominate each other. First, the individuals are split into different classes where all solutions in the same class are not dominated by any solution in that same class. To define an order within each class, it uses a measure that represents the average distance between a solution and the two solutions that are the closest along each objective.

Even though these algorithms showed good results, they are not suitable for many-objective optimization. [12] showed that when a problem consists of many objectives, almost all solutions in a population are non-dominated. As [13] correctly mentions, these domination-based MOEAs have a hard time in generating any selection pressure toward the Pareto front, since it cannot decide which solutions are more promising and therefore useful for reproduction. On top of that, since all these solutions are nondominated, an explicit diversity preservation schema is needed to sustain diversity among the population [2], leading to an increase of the time complexity.

[3] A *total order* means that we can compare all pairs of solutions. So, for every two solutions A and B in the objective space, A is either better than B, worse than B or evenly good as B.

2.1.5 Indicator-based MOEAs

To deal with many-objective optimization problems, a better approach would be with the use of indicators, also known as Indicator-Based Evolutionary Algorithms (IBEAs). In IBEAs, additional performance indicators are used to determine which solution has a better contribution to the Pareto set, and therefore must be assigned a better fitness. [14] suggests a general IBEA that compares pairs of solutions with the use of an arbitrary indicator. This way, any preference information can be adopted in this algorithm through an indicator. An example of such an indicator that is often used, is the hypervolume (HV). The HV is calculated for a Pareto set and is equal to size of the objective space that is dominated by that Pareto set. The idea is that a Pareto set that has a greater HV, is a better approximated Pareto set. Hence, if a new solution is generated, it would be seen as an improvement if the Pareto set with this solution would also give a greater HV. Several hypervolume-based MOEAs are developed, such as SMS-EMOA [15]. These algorithms seemed to be good alternatives to dominance based algorithms such as NSGA-II and SPEA2. An additional advantage is that IBEAs in general do not require mechanisms that preserve the diversity among solutions in the approximated pareto optimal set. However, they still face some flaws. The time complexity of these algorithms is exponential in the number of objectives. So, even though it is able to deal with more than just a few objectives, it will take a long time. [16] showed how reduction techniques can be added to a hypervolume-based MOEA in order to reduce this problem. [17] proposed HypE, a hypervolume-based MOEA that uses Monte Carlo to approximate the hypervolumes. This approach is faster, since it does not use, and therefore does not have to calculate, the exact hypervolume values. Instead, it compares solutions by their rank induced by the hypervolumes. [18] developed QHV-II, a hypervolume-based MOEA, which uses a divide and conquer scheme to split the problem of calculating the hypervolume into smaller subproblems.

Despite of these improvements, the computational costs to calculate the hypervolumes still grows exponentially with the increase of the number of objectives. Other indicators that are less computationally expensive have been tried as well. Examples are the enhanced inverted generational distance (IGD-NS) [19] and R2 [20] indicators, which are applied in MOEAs in for example [21] and [22], respectively.

2.1.6 Decomposition-based MOEAs

As we have seen with the MOEAs we have talked about so far, the assignment of fitness to solutions in order to compare them has been a major issue. This is because it is very hard to assign a single value to a solution. In 2007, a completely new variant of MOEA was proposed, called Multi-Objective Evolutionary Algorithm Based on Decomposition (MOEA/D) [23], which would remove this problem. MOEA/D uses an approach that decomposes the MOP into several single-objective optimization subproblems. These subproblems are then solved simultaneously using an evolutionary algorithm. Since these subproblems are single objective, the solutions can be assigned a single value and therefore easily be compared. This allows it to be solved by existing single-objective techniques. Also, it allows us to incorporate Adaptive Operator Selection (AOS) mechanisms into the algorithm, which will be discussed later on in this paper. Results show that MOEA/D outperforms other successful algorithms, such as NSGA-II. The success of this algorithm is among the reasons why in this paper we are using it to see how it can be improved. In section 2.2, we will give a more comprehensive explanation of MOEA/D.

2.1.7 Performance indicators

We have now seen various ways methods and algorithms that can be used to solve a MOP. If we want to know how well an algorithm performs and how it compares to other algorithms, we need a way to measure the performance of its final output. We call these measurements *performance indicators*. When working with Single Optimization Problems, we can simply take the solution that has the best fitness and use this as a measurement of the quality of the algorithm. This is not possible in MOEAs, since we cannot attach a single value to a solution. Therefore, we cannot take out one best solution and compare it to another algorithm's best solution. What we can do however, is compare the final set of solutions with the final set of solutions of another algorithm. In section 2.1.5, we have seen that indicators can be used to determine how much a new solution contributes to the approximated Pareto set. It then assigns a fitness to a solution based on this contribution. There are several ways of comparing approximated Pareto sets of different algorithms, which allows us to determine the performance of an algorithm.

A well-known performance indicator in MOEAs is the *hypervolume* (HV) [23]. The hypervolume represents the area of the objective space that is

dominated by at least one solution from the approximated Pareto set. Figure 1 shows an example of a two-objective problem. The blue points form the approximated PF. The red point is used as the reference point to calculate the hypervolume. When your goal is to minimize a problem, the larger hypervolume, the better. The hypervolume measures the convergence of the approximated PF.

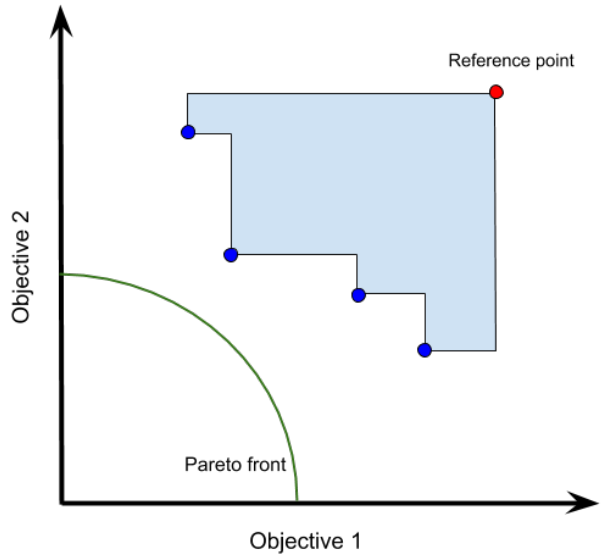


Figure 1: The hypervolume of a two-objective problem denoted by the light blue are

An advantage of HV is that algorithms can be compared without the use of a reference front. So, even if the real PF is unknown, we can still run the algorithms, take a reference point and determine which performs best. On the other hand, since it does not take into account the real PF, we can't say anything about how much improvement is left, which might be something that is desirable.

Another performance indicator is the *Generational Distance* (GD) indicator [24]. This indicator gives an idea of how "far" the solutions in the approximated front are from those of the real PF. Consider S, the approximated PF and P, the points on the real PF. The GD is calculated as follows:

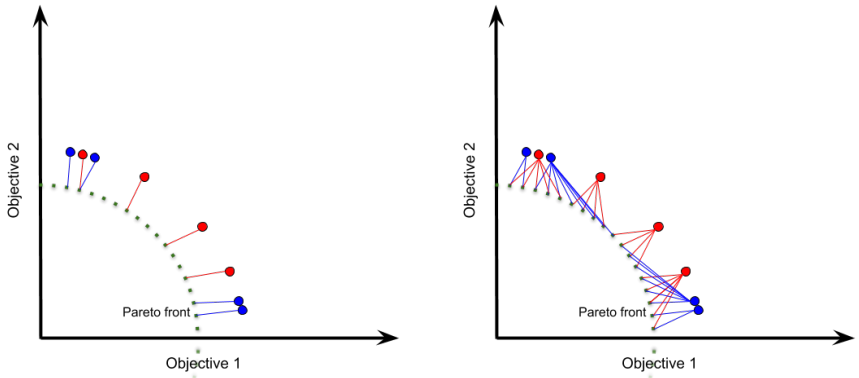
$$GD(S,P) = \frac{\sqrt{\sum_{i \in S}^{n} d(i,P)^2}}{|S|} \tag{2}$$

where $d(i, P)$ is the euclidean distance between point i and the nearest point in P. The GD measures the convergence of S; the smaller the value, the better.

[25] made a variation of this, the *Inverted Generational Distance* (IGD), by changing up the calculation:

$$IGD(S, P) = \frac{\sum_{i \in P}^{n} d(i, S)}{|P|} \tag{3}$$

where $d(i, S)$ is the euclidean distance between point i and the nearest point in S. In contrary to the IG and HV, IGD now measures both convergence and diversity of S, providing that P is large enough. To illustrate this, look at figure 2. The red and blue points form two different approximated Pareto Fronts, and the red and blue lines indicate the distance between the points on the approximated PF and points on the real PF. The GD is used in Figure 2a and the IGD in Figure 2b. Both sets of points are about equal in terms of convergence (they all have about the same distance to the real PF). However, they are not the same in terms of diversity, since the red points are better scattered then the blue ones. Still, the GD (Figure 2a) of both sets will be about the same, since all the red and blue lines have about the same length. So, the GD does not measure diversity. For the IGD on the other hand, the sum of the distances will be significantly larger and therefore also the IGD. Thus, the IGD does measure diversity.



(a) GD **(b)** IGD

Figure 2: The distances between points of the real PF and the approximated PF using (a) the Generational Distance and (b) the Inverted Generational Distance.

2.2 MOEA/D

As argued by [26], a pareto optimal solution to a MOP could be an optimal solution of a single-objective optimization problem whose objective is a weighted aggregation of the all the objectives. This is the rationality behind MOEA/D. It decomposes the MOP into N single-objective optimization subproblems (SOPs) by aggregating the objectives using different weight vectors. It maintains a population containing the best solution for every subproblem found so far. Therefore, the population size is equal to N. The neighborhood of a solution is defined by the weight vectors of the solutions. If two weight vectors are close to each other, the corresponding solutions should also be very similar. Therefore, to generate new offspring, a parent is combined with several other solutions that has weight vectors close to the parent's corresponding weight vector. During the search, an external population is kept track off to store the non-dominated solutions found during the search. After the algorithm has terminated, this population is returned as the approximated Pareto set.

2.2.1 Decomposition methods

Thus, before the MOEA/D algorithm can begin, the problem has to be decomposed into N single objective subproblems, with every subproblem having its own objective function that will be optimized simultaneously. This objective function is a weighted aggregation of all objectives in the MOP. The weights that are used are defined as follows. Let $\lambda^i = (\lambda^i_1, \ldots, \lambda^i_m),\ i = 1...N$ be the weight vector belonging to the ith subproblem with $\sum_{i=1}^{m} = 1$ and $\lambda^j_i \geq 0$ for all i $\in \{1, \ldots, m\}$. Then, using these weights, an objective function is assigned to every subproblem. There are several approaches to do this. Two well-known are the Weighted Sum and Tchebycheff approaches [27].

Weighted Sum

When using the weighted sum approach, the following objective function has to be optimized for the jth subproblem:

$$g^{ws}(x|\lambda^j) = \sum_{i=1}^{m} \lambda^j_i f_i(x) \tag{4}$$

Depending on whether the problem has to be minimized or maximized, the objective functions have to be minimized or maximized as well, respectively.

Tchebycheff

When using the Tchebycheff approach, the following objective function has to be minimized (regardless of the type of problem) for the jth subproblem :

$$g^{te}(x|\lambda^j, z^*) = \max_{1 \leq i \leq m} (\lambda_i^j |f_i(x) - z_i^*|) \tag{5}$$

where $z^* = (z_1^*, \ldots, z_m^*)$ is the reference point. During the search, z_i^* corresponds to the best value foud so far for objective f_i.

The weighted sum and the Tchebycheff approach are methods that are widely used in decomposition based MOEAs. However, they each have their own situations in which they work well. The weighted sum usually gives good results when the Pareto front of the problem is convex[4]. However, when the Pareto front is concave, it will face a problem. To illustrate this, lets take a look at Figure 3, which has a concave PF. The algorithm has generated a solution x (the blue dot) that is on or close to the PF. However, when using the weighted sum approach, every solution in the yellow area is an improvement over solution x. This means that eventually all N solutions in the population will converge to points that lay either on the intersection of the PF with the x-axis or the y-axis. This is undesirable behaviour, since one of the goals of an MOEA is to maximize the diversity of the solutions in the approximated Pareto set. With Tchebycheff, this problem will be avoided. Here, the distance to a reference point is used to determine the fitness of a solution. This way, the shape of the Pareto front does not matter.

Although tchebycheff is better when working with concave Pareto fronts, the weighted sum approach is a lot faster, partly because it does not have to take into account a reference point. This is why [28] came up with an approach that switches between these two. Here, the algorithm uses the weighted sum approach for the most part, since it is relatively fast, and switches to Tchebycheff when a non-convex region of the Pareto front is detected. The approach does not seem to have good results after all, because of the difficulty in developing an effective non-convex region detection

[4]The PF is called convex if the line segment of two points on the PF is in the feasible area. Otherwise it is called concave.

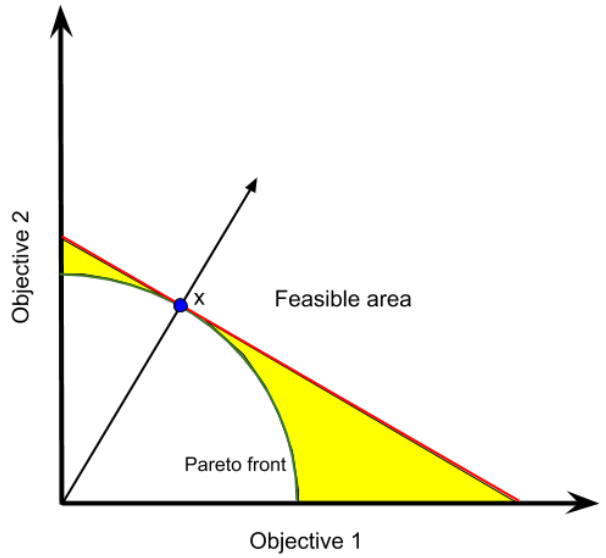


Figure 3: Showing the behaviour of the weighted sum approach in the case of a concave Pareto front. The yellow area indicating the points that are an improvement over solution x.

mechanism. However, this research did lead to other ideas. [13] for example, proposes an approach in which multiple decomposition methods are used simultaneously. They showed that this outperformed the use of a single decomposition method. New decomposition methods have been designed through the years as well. [29] proposes two new approaches that improve the performance of decomposition based MOEAs by adjusting the balance between diversity and convergence, two important goals of an MOEA.

2.2.2 MOEA/D algorithm

After the decomposition method is determined, the actual procedure can start. The problem that is used can be a minimization or a maximization problem. In this case, we are using a minimization problem. MOEA/D maintains the following information during the search:

1. A population of N solutions $x^1, \ldots, x^N \in \Omega$ where x^i is the current best solution to the ith subproblem
2. $FV^1, \ldots, FV^N$, where $FV^i = F(x^i)$ using equation (1)

3. $z = (z_1, \ldots, z_m)$, where z_i is the lowest value found so far for objective f_i
4. An external population (EP) where all non-dominated solutions are stored

MOEA/D is given the following information as input:

1. The MOP
2. A stopping criterion
3. N: the number of subproblems
4. $\lambda^1 \ldots \lambda^N$: N weight vectors
5. T: the number of weight vectors in the neighborhood of each weight vector

The algorithm is as follows:

1. **Initialization:**
   - 1.1. Initialize $z = (z_1, \ldots, z_m)$, where $z_i = min(f_i(x^1), f_i(x^2), ..., f_i(x^N))$
   - 1.2. Generate initial population $x = (x^1, \ldots, x^N)$
   - 1.3. Calculate $B(i)$, which are the indices of the weight vectors closest to vector λ^i
   - 1.4. Initialize $FV^i = F(x^i)$
2. **Update:**

   For i = 1,. . . , N:
   - 2.1. Generate a new solution y in the neighborhood of x^i using $B(i)$
   - 2.2. For every objective j: update z_j to $f_j(y)$ if $f_j(y)$ is an improvement of z_j
   - 2.3. Update neighboring solutions of x^i if y is an improvement of the objective function of the neighbour. In that case, also update the corresponding FV value.
   - 2.4. Update EP: remove all vectors dominated by $F(y)$ and add $F(y)$ if no vectors in EP dominate $F(y)$

3. **Stopping criteria**: if stopping criteria is met, then stop and return EP. Otherwise go to **Update**

The advantage of MOEA/D above dominance based MOEAs and indicator based MOEAs is that it can deal with many objects, due to the simplicity of the fitness evaluation. In fact, the fitness is just a single value as opposed to a set of objective values. Moreover, diversity along the PF is maintained automatically. Because of the weight vectors, the solutions evolve in approximately the same direction towards the PF. Choosing evenly spread weight vectors in the beginning, allows the approximated Pareto set the be diverse in the end (especially when using the Tchebycheff approach). MOEA/D also has a lower computational complexity in comparison with for example NSGA-II, since each subproblem is optimized using only information from its several neighboring subproblems. [26] showed that MOEA/D outperforms or performed similarly to alternative algorithms on most test instances.

MOEA/D brought new light in the world of multiobjective problem optimization with the use of evolutionary algorithms. The algorithm on its own already seemed to work quite well. But, more importantly, it showed room for improvement. That is why after the introduction of MOEA/D in 2007 many new algorithms arose based on this idea, mostly to mitigate the shortcomings. We have seen in section 2.2.1 that MOEA/D can be improved with the use of an appropriate decomposition method. Nevertheless, MOEA/D can be enhanced in many other ways. The next will give a summarization of some of the important algorithms that expanded on MOEA/D.

2.2.3 Improvements of MOEA/D

As we can see in the previous section, new solutions are generated for every subproblem every iteration. As a matter of fact, every subproblem is treated equally and given the same amount of computationally effort. However, they might have different computational difficulties. Therefore, it could make sense to put different amounts of energy into different problems. This is the idea behind MOEA/D with Dynamical Resource Allocation (MOEA/D-DRA) [30]. MOEA/D-DRA divides its computational efforts over the subproblems based on their usefulness. During the search process, it keeps track of an extra vector $\pi = (\pi_1, \ldots, \pi_N)$, where π_i is the utility of subproblem i. This measures how much improvement of the objective function of subproblem i has been caused by x^i (the solution belonging to the ith subproblem).

The computational efforts are then distributed over the subproblems based on their utilities. Due to this adaptation, the overall costs can be reduced, which improves MOEA/D.

[31] tries to improve the original algorithm using a whole different approach. They designed pMOEA/D, a thread-based parallel version of MOEA/D that allows the algorithm to be ran on multiple cores on a multi-core processor. To do this, the MOEA/D algorithm is adjusted a bit. The iterations from the main for-loop, i.e. the subproblems, are distributed among the threads. This way, each thread can work on a part of the population. As we know from multi-threading, the threads can interfere with each other, since they share the same memory. Therefore it is necessary that the parts where global variables are being updated are synchronized, so that the assignment from different threads does not intertwine, causing undesirable behavior. Results show that a notable time reduction can be achieved using this parallel version, while the solution quality does not significantly change, with a few exceptions.

To generate new solutions for a subproblem, MOEA/D uses only information from the subproblem's neighboring subproblems. The neighborhood of a subproblem is defined by the distance of the weight vectors. The neighborhood size T is important to the performance of the algorithm. In the original MOEA/D this parameter is fixed and determined from the beginning. However, the neighborhood size that is the best for one problem, might not be the best for another. Furthermore, within a problem, the optimal size might change over time as well. [32] addresses that a large T could be used for solutions that are trapped in locally optimal regions to get out, whereas a small T is useful for local exploitation. That is why they came up with ENS-MOEA/D, which uses a pool of different T-values. Every time the neighborhood of a subproblem is determined, a T-value from the pool will be selected with a probability based on their performance and used to create the neighborhood. This way, the neighborhood size can dynamically adapt to the most appropriate value at a particular moment. Experimental results showed that ENS-MOEA/D outperformed the original MOEA/D with different fixed Neighborhood sizes.

2.2.4 Differential Evolution (DE)

Another extension is MOEA/D-DE [18], which is based on differential evolution (DE). DE [33] is an evolutionary algorithm that instead of creating new

offspring from two parents, combines a parent with a weighted difference of several other individuals of the population to generate a new solution. First, a mutation occurs with the use of *DE mutation operators*. These operators add a weighted difference between two vectors (of the solutions) to the parent, a.k.a. the *target vector*, to create the *mutant vector*. The two vectors are in the neighborhood of the target vector. DE operators often outperform other genetic operators in single objective optimization. Since in MOEA/D, a set of single objective functions are being optimized, it make sense to use an DE operator in MOEA/D. Different DE operators exists, including these four commonly used ones:

1. **DE/rand/1**:
   $v^i = x^{r1} + F * (x^{r2} - x^{r3})$
2. **DE/rand/2**:
   $v^i = x^{r1} + F * (x^{r2} - x^{r3}) + F * (x^{r4} - x^{r5})$
3. **DE/current-to-rand/2**:
   $v^i = x^i + K * (x^{r1} - x^i) + F * (x^{r2} - x^{r3}) + F * (x^{r4} - x^{r5})$
4. **DE/current-to-rand/1**:
   $v^i = x^i + K * (x^{r1} - x^i) + F * (x^{r2} - x^{r3})$

where $x^i = (x^i_1, \ldots, x^i_n)$ is the ith target vector, v^i is the mutant vector and $x^{r1}, x^{r2}, x^{r3}, x^{r4}$ and x^{r5} are different solutions from the neighborhood of x^i.

Then, a *binomial crossover* is used to mix the target vector with the mutation vector the get the *trial vector*. Every variable u^i_j from the trial vector $u^i = (u^i_1, \ldots, u^i_n)$ is created as follows:

$$u^i_j = \begin{cases} v^i_j, & \text{with probability } CR \\ x^i_j, & \text{with probability } 1 - CR \end{cases} \tag{6}$$

In MOEA/D-DE, the DE-rand-1 is used, followed by the crossover from equation (6).

We can see that the procedure above involves three extra parameters: F, K and CR. F and K are used to control the influence of the difference between the two solutions on the mutant vector. CR controls the crossover, whereas a larger value for CR means that the trial vector is expected to inherit more information from the mutant vector and less from the target vector. To improve the performance of the algorithm, particularly in MOPs with

complicated Pareto fronts, MOEA/D-DE also applies a *polynomial mutation* to the trial vector with probability p_m. The resulting vector is then the new offspring.

The algorithm is compared with NSGA-II with the same reproduction operators. It turned out that MOEA/D-DE can outperform NSGA-II-DE. On top of that, MOEA/D-DE is less sensitive to the hyperparameters (F and CR), which makes the algorithm more robust.

Although all these extended MOEA/D algorithms reduce or completely remove some of the flaws that the regular MOEA/D entails, there is still an important issue that has not been addressed yet in this paper. It is about the application of variation operators in the evolutionary process. In previous discussed MOEA/D based algorithms, the new offspring is generated with the use of the same mutation operator during the optimization process, regardless of the problem or time step. However, this might be a very naive approach. As [34] addresses, different problems require different parameters, and therefore different operators, even for the same algorithm. So, it might be better not to stick to one operator for every problem, but choose an operator that is the best for the problem in question. However, using the same mutation operator across the whole process might also not be a good idea, since different operators might be good at different stages of the optimization process. For example, in the earlier stages, large mutation steps might be good for exploration in the search space, while later it might be better to apply a lot of small mutations in order to fine-tune the suboptimal solutions [35]. Therefore, it might be a good idea to choose amongst a set of operators. This can be done with the use of *parameter setting*. The next section will explain what this includes and goes deeper into different variants of parameter setting. The subsequent section will explain how it can be used to solve the issue described above.

2.3 Parameter setting

As discussed in the previous section, it is important to determine which operator to apply. This is referred to as *parameter setting* [30]. Parameter setting involves the determination of a parameter for an algorithm, such as the right operator. Parameter setting can be divided into two categories: *parameter tuning* and *parameter control*. Parameter tuning involves the determination of a parameter prior to running the algorithm. This is mostly done by doing a few practice runs and experimenting with different values to find suitable values. Then, the actual run can be performed with these values. However, this can be very time consuming and computationally expensive. Besides, this learning would be useless if the problem we are working with is in a non-stationary environment, meaning that the performance of the operators change during the optimization process. And since it is intuitively obvious and proven that in an EA different parameter values might be optimal at different stages of the process, making it a non-stationary process [35], we

cannot really use this type of parameter setting here. This is where parameter control comes in place. Instead of determining the values before the run and maintaining these values along the run, the values are controlled during the search process. This means that during the run, the parameters are learned and adjusted to the right values. Different kinds of parameter control exists:

- **Deterministic parameter control**: the parameter values are adjusted based on some predefined rules. It does not use any kind of feedback during the search.
- **Self-adaptive parameter control**: the parameters are part of the evolutionary process. As with the solution variables, the parameters are also altered with the use of variation operators.
- **Adaptive parameter control**: the parameters use feedback from the search to update their values.

We have already seen an example of an adaptive parameter control method in section 2.2.3, that is ENS-MOEA. Here, the neighborhood size is dynamically adapted during the search process. In this paper, we will focus on a special case of adaptive parameter control, called Adaptive Operator Selection (AOS). An AOS algorithm keeps track of the performance of an operator and makes a selection based on those performances. Later on in this paper, we will see that AOS can be incorporated into MOEA/D to improve its performance. First, we will elaborate more on AOS and see which AOS methods exists.

2.4 Adaptive Operator Selection

Adaptive Operator Selection (AOS) is a technique used to determine which variation operator should be used at a certain moment in a MOEA process. An AOS method can be broken down into two main tasks: *Credit assignment* and *Operator selection*. The former consists of assigning credit to operators based on their performance to determine their quality, while the latter uses this quality to select the right operator. An important issue that these two have to deal with is the so called *Exploration vs. Exploitation* (EvE) dilemma [3]. This dilemma implies that we want to exploit the operators that previously performed the best as much as possible, while also exploring some

worse performing operators as well. This is because the environment can be expected to be non-stationary. So, operators that perform poorly in the past, might get better over time. Therefore, it is necessary to try them from time to time. The balance between this exploitation and exploration is what the EvE dilemma is concerned with.

Different AOS algorithms exists and can vary in the way these two tasks are carried out. Some algorithms differ in the way they assign credit to the operators but have for example the same operator selection, or vice versa. In general, most credit assignment methods are either *fitness-based* or *rank-based*. In the former, the quality of an operator depends directly on the fitness improvement between child and parent, while the later defines a rank first, based on the fitness improvements, and uses these ranks to determine the credit. The operator selection of an AOS is usually either based on *multiarmed bandit* (MAB) or *probability based*. MAB based methods use the quality of an operator directly in a deterministic rule to choose the next operator to be applied. Probability based methods assign a probability based on these qualities first. Then, an operator is chosen through a roulette wheel-like process using these probabilities. In sections 2.4.1 and 2.4.4, we will go through various important and recognized credit assignment and operator selection methods and their associated characteristics.

2.4.1 Credit assignment

The credit assignment is concerned with how much credit must be assigned to an operator based on its performance. For every operator, a quality is kept track of. This value characterizes how "good" an operator is at a certain moment. In this paper, this value will be denoted with $q_{op,t}$, representing the quality of operator a at time t. After an operator has been applied to generate offspring, a reward is returned. This value represents the performance of the operator after application, usually the fitness improvement between parent and child it induced, and will be denoted as $r_{op,t}$, the reward induced by operator a at time t. After the reward is obtained, it will be used to update the quality of the operator. The credit assignment can thus again be broken down in the two parts: how to measure the impact of an operator application (the reward) and how to assign credit based on this impact assessments (the quality). The impact of the operator application is usually measured as the fitness improvement between the parent and the offspring. In the following sections, we use this fitness improvement as the reward, unless

states otherwise. The way credit is assigned, i.e. how the quality is updated can be done in various ways.

2.4.2 Fitness-based credit assignment

The most simple and naive design for a credit assignment method is to use the latest fitness improvement as the quality. However, as [36] also remarks, the nature of operators is stochastic. This means that operators can be lucky or unlucky on their last application, giving a bad indication of the overall performance of an operator. Therefore, only using the latest performance of an operator to base the quality on, would not be a smart approach.

A better, but still quite naive way is to see the quality of an operator as the empirical reward (fitness improvement), i.e. the average reward obtained during the search so far. Therefore, to obtain the current quality of an operator, the next equation can be used:

$$q_{op,t+1} = \frac{r_{op,t}, \ldots, r_{op,1}}{n_{op,t}} \tag{7}$$

where $n_{op,t}$ is the number of times that operator i is selected up to and including time step t. This methods is for example used in the *Upper Confidence Bound* (UCB) algorithm [37], which chooses the operator with the maximal quality. We will discuss this important algorithm later on in more detail. Like [3] explains, this method of using the empirical rewards works quite well in stationary environments[5], since the chances of choosing the wrong operator[6] decreases over time. However, it cannot deal well with non-stationary processes. We want the empirical reward of an operator to converge to the average of the real reward distribution as fast as possible to make a good operator selection afterwards. However, if the rewards distribution of the operator changes, the empirical reward cannot converge to this new value quick enough because of all the old obtained (and outdated) rewards from the past.

A way to reduce this problem is with the use of a *window* of size W, containing the last W rewards. The quality is then calculated by taking the average over the rewards in this window. The operators' qualities are now able to react quicker to changes in the environment, allowing an algorithm

[5]In a stationary environment, the reward distribution of the operators does not change.

[6]With "wrong operator" is meant an operator that is not optimal. The fact is that the goal is to eventually only select the optimal operator.

to make a better selection of the right operator. The performance really depends on the structure of the problem, so a general optimal size does not exists. However, a size too large prevents it from being able to quickly adapt to changes in the operator reward distribution, while a size too small leads to high loss of memory, giving a wrong estimation of the real reward distribution.

Another way to achieve quicker adjustments to changes in the environment, is to use the weighted average proposed in [38]. Here, the quality is updated as follows:

$$q_{op,t+1} = q_{op,t} * (1 - \alpha) + \alpha * r_{op,t} \tag{8}$$

As seen in [39], we can rewrite this equation to:

$$q_{op,t+1} = q_{op,t} + \alpha(r_{op,t} - q_{op,t}) \quad 0 < \alpha \leq 1 \tag{9}$$

where α controls the contribution of the new reward obtained to the value of the new calculated quality. This means that a larger value for α results in more importance of the recent rewards and makes it therefore more robust against abrupt changes in the operator reward distribution. However, a value too large would give a similar problem as a window size that is too small, since it would forget old data too fast. With an α too low, it cannot adjust fast enough, just like a window size too large. Therefore, the weighted average has a somewhat similar effect as using a window.

A whole different approach is proposed by [40]. It proposes the *Extreme Value Based* (EVB) Credit Assignment. The rationale for this approach is that rare but large fitness improvements obtained by the application of operators are more important than common and small improvements. Therefore, it is better to look out for extreme rewards and choose the operator that currently produces high fitness improvements from time to time. This cannot be done with the use of the empirical reward as the quality of an operator, described earlier in this section. Since the average of an operator that gives rare but large improvements is likely to be 0 after a few utilizations, it will seldom be selected in the future. In the case of using the instantaneous rewards (only the latest fitness improvement) as the quality of an operators, this problem becomes even worse. EVB however, can do this. For this purpose, it maintains a window of size W for every operator that stores the last W rewards (fitness improvements) of the operator. The quality of the operator at a certain time step is equal to the maximum reward in the window.

So, let's say that the the window $[r_o p^1, \ldots, r_o p^W]$ contain the last W fitness improvements of operator op, then the quality is calculated as follows:

$$q_{op,t} = \max([r_o p^1, \ldots, r_o p^W]) \tag{10}$$

By doing so, the algorithm tends to select operators that produce rare but high fitness improvements faster and therefore exploits these operators. It is important to determine the right value for the only parameter W in this method. If too large, the algorithm will stick to an operator that performed very well in the past for too long. If it is too small however, operators that perform very well but rarely, will be ignored or forgotten too fast.

Although these credit assignment methods already improve upon simply using the instantaneous rewards as the operator quality, there are still a few drawbacks. The reward of an operator is the raw fitness improvement between a parent and the offspring that is generated using that operator, which is then directly used in the calculation of the operator's quality. However, as [41] also says, the range of raw fitness improvements varies (1) among different problems and (2) even during the optimization process. This means that the hyperparameters (such as the window size) of the AOS method need to be tuned for every problem separately. An AOS algorithm using this fitness improvement directly in its credit assignment will not be very efficient when used for a problem that is not the same problem that is used in the offline tuning of the hyperparameters. And even if the algorithm has been tuned for the problem, its behavior might not be optimal during the search process, because the fitness range varies over time. In the beginning of the process can application of operators lead to much better offspring, compared to their parents, fitness-wise. This is because the solutions in the population at the earlier stages have a lot of room left for improvement, whereas later, the solutions are closer to optimal. This way, the same gain might have different weights at different moments. Therefore, the use of raw fitness improvements to determine the credit might lead to undesirable behavior and deteriorate the algorithm's robustness.

To make the credit assignment more robust when using it for different problems, [42] suggests a normalization scheme. Here, the reward that is returned is first divided by the maximal reward that has been gathered by an operator in the current window. Then, the quality is updated according the credit assignment in question. This way, the rewards will always be

between 0 and 1, where a reward of 1 reflects the maximum reward in the current window.

[43] uses a somewhat similar approach. they determine the reward with the use of a so-called *relative fitness improvement*:

$$\eta_i = \frac{\delta * |pf_i - cf_i|}{cf_i} \tag{11}$$

where δ is the fitness of the best solution in the current population, cf_i the fitness of the offspring solution and pf_i the fitness of the parent solution obtained by operator i. They then investigate how using different methods described above (average, extreme, normalization) behave in a combination with an operator selection mechanism.

Even though these normalization methods reduces the problem-dependency of the algorithm, it does not remove the problem. Furthermore, the algorithm is not invariant with respect to monotonous transformations of the fitness function, which is an important property if a robust algorithm wanted according to [44]. This invariance includes that the algorithm does not behaves differently if the problem is transformed in such a way that the fitness function maintains its ordinal properties (so there is no change in order). An example would be a problem containing a certain currency that is transformed to the same problem, but with another currency. When the raw fitness improvements are used directly to update the quality, this invariance does not hold.

In order to address these issues, several approaches were proposed that assign credit to the operator based on their ranks, instead of their raw rewards (fitness improvements). In the next section, a few credit assignment methods that use this rank-based approach are discussed.

2.4.3 Rank-based credit assignment

As we have seen in the previous section, the credit assignment can update the operator qualities using the raw fitness improvements directly. This however, has a large impact on the robustness of the algorithm with respect to fitness transformations. Therefore, it might be better to use another type of credit assignment, namely rank-based credit assignment. Here, the reward obtained from the application of an operator is used the determine a rank. These ranks reflects how many operator applications lead to a higher fitness improvement and how much lead to a lower one, usually also within a window. This way,

the mechanism relies less on the actual fitness values, but focuses more on the mutual position. This will in turn eliminate the problem of varying fitness ranges among different problems and along the search process.

[44] came up with two credit assignments that uses these ranks, namely the *Sum of Ranks* (SR) and the *Area Under Curve* (AUC) methods. They use windows of size W as well here, corresponding to the last W time steps that the algorithm looks back. Each slot of the window contains the operator that is applied, together with its reward. Every slot is then given a rank r based on this reward. The highest reward gets an r-value of 1, the second highest an r-value of 2, etc. Then the corresponding rank-value to each operator application is calculated as follows:

$$D^r(W - r) \tag{12}$$

where D is a decaying factor. D allows the higher-ranked rewards to have a bigger impact on the operators' quality.

In the SR credit assignment, the quality of an operator i at time step t is calculated by adding up all the rank-values of i and normalize it by the sum of all rank-values in the window:

$$q_{op,t} = \frac{\sum_{op_r=i} D^r(W - r)}{\sum_{r=1}^{W} D^r(W - r)} \tag{13}$$

AUC is based on the *Area Under the Curve* paradigm. It starts of by drawing the *Receiving Operator Curve* (ROC). This is done as follows. Consider a list, which is the window W, sorted by the rewards. Then, starting from the origin, a graph is drawn by moving through the list and adding a vertical line each time the particular operator is found in the list, a horizontal one if not, and a diagonal in case of ties. As the name suggests, the operator's quality is equal to the area under the curve, normalized by the sum of all operators' AUCs.

The length of each segment is dependent of D, that is $D^r(W - r)$. This makes the value of D very crucial here. The smaller D, the more influence the higher-ranked rewards have on the operators' quality. Of course, the algorithm is also sensitive to the window size W. A large value for W will lead to a conservative operator selection where the algorithm will stick to the same operator for a long time, whereas a small value causes the past to be forgotten quickly. Due to the use of these ranks, both AUC-B and SR-B are known for their robustness and efficiency against different fitness landscapes

Quite recently, [41] proposes another credit assignment that uses ranking instead of raw fitness improvement to update the qualities, called *FRR*. Again, it uses a sliding window to allow it to be applied in a dynamic environment, such as an MOEA. On top of that, it applies a decaying mechanism to exploit the best operator. It does this by first defining the *fitness improvement rates* (FIR). When an operator i is used at time step t, the FIR is calculated as follows:

$$FIR_{i,t} = \frac{r_{op,t}}{pf_{i,t}} \tag{14}$$

where $pf_{i,t}$ is the fitness of the parent and $r_{op,t}$ the fitness improvement. The FIR value represents the improvement rate of the child, relative to the parent. Then, the total reward for every operator is calculated, which is the total of all the FIR values for the operator in the sliding window. This means that if an operator has not been selected in the current window yet, it receives a $TotalReward$ of 0. The operators are then ranked by their $TotalReward$ value. A decaying factor $D \in [0, 1]$ is used to calculate the decayed reward:

$$Decay_i = D^{rank_i} * TotalReward_i \tag{15}$$

where $rank_i$ is the rank of operator i. Here, the parameter D can be used to control the exploration/exploitation of the algorithm. We can see that a smaller D-value increases the chance of the best operators to be selected and therefore boosts the exploitation. Finally, the quality of operator i at time step t is calculated in a normalized manner:

$$q_{op,t} = \frac{Decay_i}{\sum_{j=1}^{K} Decay_j} \tag{16}$$

Overall are rank-based mechanisms, such as AUC, SR and FRR, considered to be robust algorithms with respect to its hyper-parameters, since they are not dependent directly on the raw fitness improvements. They have shown to be very efficient, while also being very robust to many situations and different problems [36].

According to [45], the fitness improvement is not the only indicator to the progress of evolution. Therefore, they proposed *Compass*, an AOS in which the credit assignment is done by taking into account both the fitness improvement and population diversity. Here, the quality is a weighted sum of these two. To do this, for every operator, the average fitness improvements (ΔQ) and average population diversity (ΔD) over the last τ applications is

recorded, along with the average execution time of the operator applications. The normalized values of these measures represent a point on a graph, one axis being the ΔQ and the other being the ΔD. Then a plane is defined by an angle Θ. The points/vectors (o^n_{*t}) and plane (c) are shown in Figure 4. We can divide this graph in four squares. Since the fitness and diversity are somewhat opposite goals, most of the points will lay in the top left square (an increase of the fitness and a decrease of the diversity) and in the bottom right square (and increase of the diversity and a decrease of the fitness). The quality of an operator is then calculated as the shortest distance between the point and the plane, divided by the average execution time. The latter is to reward faster operators.

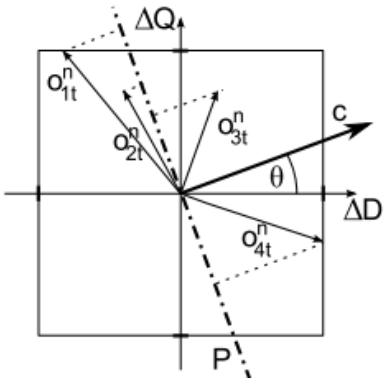


Figure 4: *Compass*: points o^n_{*t} and an angle Θ are used to define the quality of an operator. ΔQ denotes the average fitness improvements and Δ denotes the average population diversity [45].

These qualities are then mapped to a probability and, using these probabilities, an operator will be selected in a probabilistic way. This is called *Probability Matching* and will be discussed in section 2.4.6. The angle Θ can be used to attach more weight to the fitness improvement or the population diversity. If more weight is attached to the average fitness improvement, the algorithm will favor the operators that produce offspring with a higher fitness and therefore lead to more exploitation. On the other hand, if more weight is attached to the population diversity, more exploration will be induced. An interesting observation is done by [45]. Consider an angle which gives equal importance to both ΔQ and ΔD. In the early stages of the EA, the diversity is high, but the solutions' fitness is low. So, most operators tend to be located in quadrant *II*. After some time, the (local) optima will be approached,

which leads to lower fitness improvements. The points that are still in the II-quadrant correspond to the exploitative operators and will move towards 0 (the x-axis). Eventually, the fitness improvements become scarce, while the exploration operators still generate solutions that improve the diversity. So, the exploration operators will eventually be favored over the exploitative ones. This results in a search that shifts away from the optimum. This is something to keep in mind when one wants to understand the behaviour of the operators and algorithm.

Until now, we have talked about credit assignments which use the raw fitness improvements as the reward, and are used to update the quality of the operator either directly or by defining a rank. However, there are other ways to do this, like in *Self-adaptive Differential Evolution* (SADE) described in [46]. In SADE, the qualities are updated with the use of so called *success and failure memories*. It is called a success if an operator is applied and it leads to a fitness improvement, whereas a failure is when the application leads to a decrease of fitness. At every time step t, for every operator op the number of successes and failures are recorded, resp. denoted as $ns_{op,t}$ and $nf_{op,t}$. Again, a window of size W is used to let the algorithm forget old data. The quality of an operator op at time step t is calculated as follows:

$$q_{op,t} = \frac{\sum_{t=t-W}^{t} ns_{op,t}}{\sum_{t=t-W}^{t} ns_{op,g} + \sum_{t=t-W}^{t} nf_{op,g}} + \varepsilon \tag{17}$$

Basically, the quality here can be regarded as the success rate of the operator in terms of how often the operator generates offspring that are an improvement over their parents. Epsilon ϵ is used to avoid qualities of 0. This way, an operator that has not been successful in the previous W time steps, will still receive a small quality, allowing it to have a chance of being selected in the future. Afterwards, an operator's probability is computed as the proportion of the operator's quality to the sum of all qualities:

$$p_{op,t} = \frac{q_{op,t}}{\sum_{\mathrm{J}=1}^{K} q_{op,t}} \tag{18}$$

The larger the success rate relative to the others, the larger the probability of the operator being used to generate new solutions. [46] incorporated this approach in a differential evolution (DE) algorithm, where every iteration for every solution an mutation operator is selected among a pool of four DE mutation operators.

We have now seen various ways of assigning credit to operators. In these methods, different ways of measuring the operators' performance and different ways of using this measure to update the operator's current quality are used. After these updates, the next part of the AOS comes in action: the operator selection. In the next section, we will go through several operator selection methods and their properties.

2.4.4 Operator selection

After the qualities of the operators are updated by the credit assignment, the operator selection will select an operator based on these values. In the literature, this is done in various ways, but these methods can be divided into two categories. One is based on the so-called *Multi-Armed Bandit* (MAB) framework and uses the quality directly together with and exploration term to deterministically choose amongst a set of operators. AOS methods that use this type of operator selection will be referred to as bandit-based methods and will be explained further in section 2.4.5. The other way is to use the qualities to attach a probability to the operators and choose amongst them by a roulette wheel-like process. These approaches will be referred to as probability-based methods and we will go deeper into this subject in section 2.4.6.

2.4.5 Bandit-based AOS methods

Bandit-based AOS methods are based on the *Multi-Armed Bandit* (MAB) Problem paradigm. Here, the operators are regarded as a set of arms of slot machines, each having an unknown reward distribution. The goal is to maximize the cumulative reward during the process. By modelling the rewards that the arms give during the process, the best arm (operator) can be chosen at each moment in time. Many algorithms based on this idea have been invented. One of the first attempts is the *Upper Confidence Bound* (UCB) algorithm [37]. The credit assignment credits its operators by the empirical reward, i.e. the average reward obtained during the search so far. At time t, UCB selects the operator with the maximal quality q. Since this would fully exploit the best performing operator so far without any exploration of other operators, UCB adds a *confidence interval* (the square root term):

$$\underset{op=1...K}{\arg\max} \left(q_{op,t} + C\sqrt{\frac{2\log\sum_{j=1}^{K} n_{j,t}}{n_{op,t}}} \right) \tag{19}$$

where K is the number of operators and $n_{op,t}$ the number of times that operator i is selected up to and including time step t. The confidence interval depends on $n_{op,t}$ in such a way that the operator selection will favor an operator with the same quality but has been selected less often. Parameter C is used to determine the weight of the confidence interval and therefore controls the exploration. A larger value of C will lead to more exploration and a smaller value to more exploitation. Unfortunately, UCB is not usable in a dynamic environment, since the qualities, which are the empirical rewards, have difficulty adjusting to a shifting reward distribution. To make UCB suitable for dynamic processes, such as MOEAs, several extensions of UCB have been proposed.

One of the first contributions that build upon UCB is the *Dynamic Multi-Armed Bandit* (DMAB) algorithm [47]. DMAB adds a change-point test to UCB. This test, called the *Page-Hinkley* (PH) test [48], detects when a change in the operator reward distribution has taken place. If so, DMAB will restart UCB. This will reset the empirical reward of the operators (as well as $n_{i,t}$), making it possible again for the empirical rewards to converge rapidly to the right values again. It uses two extra hyperparameters (on top of the hyperparameter C of the UCB algorithm):

- $\boldsymbol{\lambda}$: controls the trade-off between false alarms and unnoticed changes.
- $\boldsymbol{\delta}$: makes the test more robust against slowly varying environments.

The algorithm was tested against two other state of the art AOS methods at that time, *Adaptive Pursuit* and *Probability Matching*, which we will discussed in the next section. It performed very well against these and performed even better when using boolean rewards instead of real value rewards.

[34] builds on this idea. This algorithm, called *Ex-DMAB*, also uses the PH test to restart UCB when a change in the reward distribution of an operator has been spotted. However, the credit assignment is different. It uses the *Extreme Value Based* (EVB) Credit Assignment described in section 2.4.1 to assign the right credit to the operators.

Although these algorithms give good results in some ad hoc scenarios, there are drawbacks. First of all, the HP test will only detect a change if the

change is sudden and abrupt. However, the reward of the operators often swifts slowly, letting the change go unnoticed. Moreover, when the test is triggered, the memory of the algorithm will be reset and has to be recreated by the exploration again. Useful information could be lost this way.

This leads to another method that uses UCB as a basis, called the *Sliding Multi-Armed Bandit* (SlMAB) algorithm [3]. To avoid the problems DMAB is facing, SlMAB uses only a part of the history to calculate the operator's quality in the first place. It makes uses of a window size W, referring to the number of steps it looks back. The quality of an operator i at time step t is calculated as follows:

$$q_{op,t+1} = q_{op,t}\frac{W}{W + (t - t_i)} + r_{op,t}\frac{1}{n_{op,t} + 1} \tag{20}$$

where t_i is the last time step that operator i has been applied and $n_{i,t}$ is the frequency of the application of operator i up to time step t.

The application frequency of operator i is updated as follows:

$$n_{op,t+1} = n_{op,t}\left(\frac{W}{W + (t - t_i)} + \frac{1}{n_{op,t} + 1}\right) \tag{21}$$

If we look at equation 20, we can see that, besides the reward, the quality of an operator depends on three extra values: the window size W, t_i and $n_{i,t}$. We already know that the window size is used to make sure that the algorithm does not use data that is outdated. t_i is used to determine the number of steps since the last application of the operator $(t - t_i)$. It makes sure that if an operator has not been applied for a long time, the new quality moves slightly more to the direction of the new obtained reward then if the operator has just been applied a couple of time steps ago. This way, the qualities can adjust more rapidly to the real reward distributions. Lastly, the frequency $n_{i,t}$ is used to preserve the exploration and exploitation trade-off. If an operator i is applied often, $n_{i,t}$ will be large, which means that the new obtained reward will have a lower contribution to the calculation of the new quality. However, if an operator hasn't been applied much in the last W time steps, the quality will be increased more. After the new quality is calculated, the operator selection from regular UCB is used (equation 19). SlMAB makes sure that if an operator is applied very often, $n_{i,t}$ converges quickly to W and the qualities accurately reflect the real reward distribution. An additional

advantage of SlMAB over DMAB (and Ex-DMAB) is that whereas DMAB has three hyperparameters (C, λ and δ) , SlMAB only has two (C and W), making it easier to tune the algorithm.

Although SlMAB already improved on a few aspects as described above, the bandit-based methods discussed so far still contain a major drawback. This has to do with the credit assignment. As discussed earlier in 2.4.1, using the raw fitness improvement between the parent and offspring can be disadvantageous for the robustness of the algorithm. Instead, it would be better to use rank-based credit assignment methods. This is why in [41] a bandit-based AOS algorithm is proposed which uses the FRR credit assignment, called *fitness-rate-rank-based multiarmed bandit* (FRRMAB). To select an operator, the UCB selection is used once more. A small difference here is that $n_{i,t}$ is now the number of times operator i is selected in the sliding window at time step t, instead of during the whole run so far.

Two new AOS methods where proposed based on the idea of FRRMAB, called *UCB-tuned* and *UCB-V* [49]. They use the exact same credit assignment as FRRMAB. For the operator selection, the UCB selection procedure is used with an addition. It also uses the quality and a confidence interval to select an operator. However, the confidence interval does not only depend on the number of times the operator is used, but also on the variance of the operators' rewards. This way, the confidence interval will get tighter in order to reduce the application of suboptimal operators. For UCB-tuned, at time t, the operator is selected that maximizes this equation:

$$\underset{op=1...K}{\arg\max} \left(q_{op,t} + C\sqrt{\frac{2\log\sum_{j=1}^{K} n_{j,t}}{n_{op,t}} * \min(\frac{1}{4}, V_{op})} \right) \tag{22}$$

with

$$V_{op} = \sigma_{op}^2 + C\sqrt{\frac{2\log\sum_{j=1}^{K} n_{j,t}}{n_{op,t}}} \tag{23}$$

where σ_{op}^2 is the variance and $n_{i,t}$ is the number of times that operator op has been applied. For UCB-V, this equation has to be maximized:

$$\underset{op=1...K}{\arg\max} \left(q_{op,t} + C\sqrt{\frac{2\log\sum_{j=1}^{K} n_{j,t} * \sigma_{op}^2}{n_{op,t}}} + 3 * \frac{\sum_{j=1}^{K} n_{j,t}}{n_{op,t}} \right) \tag{24}$$

The use of the variance of the operators' rewards would lead to a better exploration versus exploitation trade-off, which makes it more robust than the original FRRMAB, according to [49].

The bandit-based methods described above all use the same operator selection or a variant of this as the one used in the original UCB algorithm. They choose the operator with the best quality currently, including some confidence interval. Therefore, they all have a deterministic way of choosing an operator. Another way is to attach a probability to every operator based on the qualities and use these probabilities to chose amongst a set of operators in a stochastic way. In the next section, we will examine these methods further.

2.4.6 Probability-based AOS methods

In the previous section we have seen how AOS methods based on the Multi-Bandit paradigm work and what their properties are. A whole different type of AOS is probability based AOS. In a probability based AOS method, a probability is attached to each operator and a roulette wheel-like process is used to select an operator to be applied. This is different from bandit-based algorithms, since they select an operator based on a deterministic rule (equation 19), whereas in probability based AOS algorithms the selection process is stochastic. Therefore the goal is to maximize the expected value of the cumulative reward. The probabilities are based on the operators' qualities and the qualities are calculated using a credit assignment method described in section 2.4.1. Now, two well-known probability based AOS methods will be discussed: *Probability Matching* and *Adaptive Pursuit*.

Probability Matching

[50] introduces *Probability Matching* (PM), a way to select an operator every iteration by assigning probabilities to the them. After calculating the quality using the weighted average (equation 8), bandit-based AOS algorithms would use the calculated qualities directly in the operator selection from UCB (equation 19). However, probability based methods assign a probability to the operators first and hence does PM. [50] calculates this probability as the proportion of the operator's quality to the sum of all qualities:

$$p_{op,t} = \frac{q_{op,t}}{\sum_{j=1}^{K} q_{j,t}} \tag{25}$$

It might seem that when the right value for α in equation 8 is chosen, the algorithm is able to adjust its operator probabilities in an appropriate manner. However, there is a danger here. If an operator's quality approaches 0, its probability will approach 0 as well. It will therefore hardly ever be chosen, preventing the probabilities to adapt fast enough in a non-stationary environment. Let alone if the probability reaches 0, whereupon the operator will never be selected again in the future. This is why [39] proposed an adapted version for this probability assessment. It uses a minimal value p_{min}:

$$p_{op,t+1} = p_{min} + (1 - K * p_{min}) \frac{q_{op,t}}{\sum_{j=1}^{K} q_{j,t}} \quad 0 < p_{min} < 1 \tag{26}$$

where K is the number of operators. The p_{min} ensures that the operators retain a minimal probability. This way, when an operator does not receive (much) reward for a long time, its quality converges to 0 and its probability of being selected converges to p_{min}, allowing the operator to be chosen still in the future.

Adaptive Pursuit

In the previous section, we have seen how PM can adapt to changes in a non-stationary environment. Still, this method leads to some behavior that is not optimal. As [39] shows, the closer the operators' rewards are, the closer the probabilities are as well. However, our goal is to maximize the expected value of the cumulative reward, which will be obtained by maximizing the probability of the operator with the best quality, while applying the others with a probability of p_{min}. This is not the case for PM. Therefore, *Adaptive Pursuit* (AP) is proposed by [39], which should be better at maximizing this expected value of the cumulative reward.

The idea of AP is that is pursued the operator with the best quality at the moment. This is done by increasing the probability of this operator and decrease the probability of the others. Initially, the probabilities of all operators are set to $1/K$. Just like PM, it uses the weighted average credit assignment to update the qualities. Then, the operator with the best quality is selected, let's call this operator op^*. The probability of operator op^* is updated as follows:

$$p_{op^*,t+1} = p_{op^*,t} + \beta(p_{max} - p_{op^*,t}) \quad 0 < \beta \leq 1 \tag{27}$$

while the other probabilities are updated as follows:

$$\forall op \neq op: \; p_{op,t+1} = p_{op,t} + \beta(p_{min} - p_{op,t}) \quad 0 < \beta \leq 1 \tag{28}$$

where $p_{max} = 1 - p_{min} * (K - 1)$ and β the learning rate. As we can see, if an operator is the best one repeatedly, the corresponding probability converges to p_{max}. On the other hand, the other operators will converge to p_{min}. The learning rate β determines how fast this convergence takes place. Because the current best operator is now pursued at a certain rate, we can on the one hand exploit the operator that is performing the best, while also being able to quickly adapt if another operator becomes the superior.

2.5 Combining MOEA and AOS

The previous sections showed how MOEAs, in particular MOEA/Ds, can be used to solve a MOP and what research is done in this field. Next, we talked about how AOS works and which different approaches there are. We have seen that these AOS methods can, in some way, adapt to changes in the reward distribution. Thus, it can be used in non-stationary environments, such as MOEAs. This is also what we are going to investigate in this paper. In this paper, MOEA/D will be enhanced by incorporating an AOS method. Section 3 will explain in detail how this is done. The use of adaptive operator selection mechanisms in multiobjective optimization is not new. Several AOS methods have been integrated in an MOEA, usually to see what the impact is on the performance. In this section we will give a few examples of literature where algorithms are proposed that use this combination.

The SaDE algorithm discussed in the previous section uses the fitness improvements to determine whether the application of an operator is a success or a failure. Since in a problem with multiple objectives, solutions cannot be compared by a single fitness value, SaDE is only suitable for SOPs. This is fine if we want to use it in combination with MOEA/D, since MOEA/D consists of a bunch of single-objective functions to be optimized. However, if we want to make it fit into a dominated-based MOEA for example, the evaluation criteria for inferior solutions needs to be changed. Because of that, SaDE is extended to deal with multi-objective problems, i.e. MOSaDE [51]. In MOSaDE two solutions are compared by their pareto dominance, and in case they do not dominate each other, the least crowded solution is superior. Then, if the offspring is better than its parent, the operator probabilities are updated just like in SaDE. MOSaDE is even further extended to

OW-MOSaDE [52]. They claim that different objective functions may possess different properties. Therefore, instead of learning one set of operator probabilities, it keeps track of a set of probabilities for every objective.

As explained earlier, an AOS can easily be combined with MOEA/D, because of the set of single objective functions that it is decomposed into. These single objective functions are defined by a single value (fitness) which can be used be the AOS to determine the impact of an operator. There are broadly three main literature where AOSs are combined with MOEA/D ([41], [49], [53]) and we will focus on these in this section. First, we will see how these algorithms perform with and without AOS. Then, we will compare the different AOS methods used in combination with MOEA/D. And lastly, the influence of different problems and parameter settings will be discussed.

2.5.1 Comparing MOEA/D and AOS combinations

An example of a successful algorithm that combined an AOS method with MOEA/D is MOEA/D-FRRMAB [41]. Here, the AOS method FRRMAB is incorporated into MOEA/D-DRA. From section 2.4.5, we know that FRRMAB uses a credit assignment that converts fitness improvements to a rank and subsequently transforms this rank to a quality using a decaying mechanism. This quality is then used by the UCB operator selection (equation 19) to select the operator that will be applied. MOEA/D-FRRMAB was compared with other well-known MOEA/D based algorithms that do not use an AOS: MOEA/D-DE, MOEA/D-DRA & ENS-MOEA/D. It turns out that MOEA/D-FFRMAB outperforms these algorithms in most testcases based on the IGD and I_H performance indicators. ENS-MOEA/D uses just like MOEA/D-FRRMAB an adaptive parameter approach. Instead of recording the performances of the operators and in this way adaptively choose among the operators, it makes the neighborhood size flexible during the search process. We could therefore conclude that it is more profitable to improve MOEA/D on its operator selection then putting effort into letting the algorithm find the right neighborhood size during the search. Of course, this is the case for the problems and parameter setting that where used in this study. We don't know how this will differ if we try other parameter values and use the algorithm for other problems, for example problems with many objectives (more than just three). Moreover, the CPU time of ENS-MOEA/D is also much higher than MOEA/D-FRRMAB. Since both MOEA/D-DE and MOEA/D-DRA are faster than MOEA/D-FRRMAB, it is clear that

the good performance of the latter comes at the expense of the computation time. This makes sense, since it needs to do significantly more calculations in its adaptive operator selection.

Besides comparing MOEA/D-FRRMAB with famous MOEA/D algorithms that do not use any adaptive operator selection, it is also compared with MOEA/D algorithms that use other AOS methods. To do this, they replaced the operator selection method from FRRMAB with the operator selection of Probability Matching (PM) and Adaptive Pursuit (AP). These two algorithms still use the same credit assignment as FRRMAB, namely FRR, but selects an operator using either PM or AP. They also tested the combination of MOEA/D with *SRMAB* as AOS. In SRMAB, the Sum of Rank credit assignment is used, together with the operator selection from UCB (eq. 19). Again, in an extensive experiment, it turned out that MOEA/D-FRRMAB outperforms its fellow algorithms. SRMAB also performed quite well, ending up second in this pool of algorithms. From these results, it seems that bandit-based methods are better overall than probability based methods. As mentioned by [41], this would be caused by the theoretically sound way of balancing exploration and exploitation, the main dilemma of AOS algorithms.

The probability based AOS methods AP and PM are also studied in combination with MOEA/D by [53]. However, their focus was on the influence of using different ways of measuring the impact of an operator, i.e. ways of constructing the reward. This was done using the relative fitness improvements from [43] (eq. 11). Four different ways are investigated:

- AvgAbs: the average relative fitness improvement
- AvgNorm: the average normalized relative fitness improvement
- ExtAbs: the extreme relative fitness improvement
- ExtNorm: the extreme normalized relative fitness improvement

In all these methods, the window size is exactly one generation, i.e. the population size N.

Then, these four are combined with AP and PM, resulting in eight different algorithms and see which one performed the best. It turned out that, with the investigated parameters, most algorithms using Probability Matching scored higher than when Adaptive pursuit is used. One strong argument

that this paper gives is that this is because of the fact that with AP, it is not possible to apply multiple operators frequently in a short period of the search process, since AP can only exploit one operator at the time. However, in a MOEA/D, different subproblems have different characteristics, which means that one operator performs very well for one subproblem and another for another subproblem. We might therefore want to attach a high probability to multiple operators. Since PM selects its operators using probabilities proportional to their qualities, it is able to select multiple operators with a relative high probability compared to operators with a probability P_{min}. Amongst the four different algorithms using PM, the one using the extreme absolute reward approach performed best. This combination of ExtAbs with PM and MOEA/D, called *ADEMO/D*, is compare with other powerful multiobjective optimization algorithms, including MOEA/D-DRA, NSGA-II and ENS-MOEA/D. ADEMO/D was by far the best performing algorithm if we measure the quality of Pareto sets with Unary Epsilon, Hypervolume and the *R indicator*. Also, the IGD value is investigated. It turned out that ADEMO/D performed well when applied to two objective problems when looking at the IGD. However, the problems with three objectives seem to be difficult for ADEMO/D. In most cases, the other algorithms perform better. They argue that this makes sense, since the other algorithms (except for NSGA-II) are able put its computational effort into subproblems that are more profitable, with the use of Dynamical Resource Allocation (DRA), explained in 2.2.3. ADEMO/D lacks this ability and therefore wastes more effort on solutions that are less improvable.

[49] also tested the FRRMAB variations, *UCB-tuned* and *UCB-V* (section 2.4.5), in combination with MOEA/D. UCB-tuned and UCB-V use the UCB operator selection, just like FRRMAB. However, it also integrates the variance of the operators' rewards into the confidence interval of this calculation. It turned out that UCB-tuned was the best, among UCB-V and FRRMAB when combined with MOEA/D and the metrics IGD and Hypervolume are considered. The better performance of MOEA/D-UCB-tuned can be explained by the fact that it focuses more on the exploitation than the other algorithms, thanks to its unlimited influence of the variance. We could therefore conclude that the use of variance in an bandit-based algorithm is advantageous and that it is better to keep this variance influence unlimited w.r.t. the performance.

2.5.2 Parameter influence

In the previous section, we have summarized important research in the field of combining MOEAs and AOS methods. It is proven that in many cases, the use of an AOS can significantly improve the performance of MOEA/D. In order to make an AOS method work well in these environments, it is important that the right values for the parameters used in these methods are chosen. This is why [53] also studies the influence of these parameters to the performance of the eight different algorithms that were tested. The parameters of PM and AP (p_{min}, α and β) are studied, as well as the parameters that are used by the Differential Evolution (CR and F). Again, the IGD performance indicator is used to do the comparison. It turned out that for both PM and AP with the use of the extreme absolute reward approach, they are indifferent of what parameter values for p_{min}, α & β are used. The only exception is when the p_{min} is set to 0.00. This is because when $p_{min} = 0.00$, the operator will not be selected and therefore not be selected ever in the future. This is clearly a detrimental behaviour when applied in a non-stationary environment. Looking at the parameters for the DE mutation operators (CR and F), it turned out that default values given by [18] are the optimal values in most test instances. Exceptions are when problems that contain parabolic or convex properties. Here, it showed that a low value for CR leads to better IGD values.

[41] also studied the sensitivity of the parameter values for their algorithm MOEA/D-FRRMAB. They combined different values for the scaling factor C, sliding window W and decaying factor D. All these configurations where then tested on two problems UF3 and UF7 [54] and two performance metrics were measured: the hypervolume and IGD value. Then, for each problem and performance metric, the best configuration was selected and compared with the other configurations using the Wilcoxon rank sum test. As it turned out that, in these four cases, about 50 % of the configurations were significantly worse than the best configuration. Only on UF7, in terms of the hypervolume, the best configuration only outperforms a small percentage of the other configurations. However, in the other cases, it significantly matters what the parameter values are, which makes MOEA/D-FRRMAB sensitive to the parameters.

Despite the fact that AOS mechanisms can easily be integrated into MOEA/D, little research has been done in the field combining these two.

Of the research that exists, [41] provided a significant part of it, as described earlier on in this section. Among other things, they tested their new credit assignment mechanism, FRR, with both the Adaptive Pursuit and Probability Matching operator selection mechanisms. However, both of these face drawbacks. AP can only pursue one operator at the time. However, in a MOEA/D it might be optimal to exploit more than one operator by applying multiple operators frequently in a certain time period. PM is the other extreme, which selects an operator purely in a proportional manner. This may cause too little room for exploitation of the best operator. It might be better to see if a method exists whose behaviour is somewhat in between. Therefore, we propose a new operator selection mechanism based on *Tournament Selection*. This mechanism will be explained further in section 3.

3 Methods

The method section is structured as follows. First, the new proposed algorithm will be explained, including the pseudo-code. Then, we will shortly comment on the implementation for the experiments. Lastly, we will elaborate on the experiments that were done.

3.1 New algorithm: *MOEA/D-FRR-TS*

As explained in the previous chapter, we have implemented a new algorithm based on Tournament Selection, which we call *MOEA/D-FRR-TS*. We have used MOEA/D-DRA [30], an improved version of MOEA/D that won the CEC 2009 MOEA competition. This version has two differences with normal MOEA/D, which we explained in section 2.2. Firstly, it divides its computational efforts in a more clever manner in order to improve the performance. This is explained in more detail in section 2.2.3. Additionally, instead of using a external population to keep track of the non-dominated solutions, the current population after termination is used as the approximated pareto set.

[41] build their AOS method, FRRMAB, on top of MOEA/D-DRA to improve the algorithm's performance. This AOS method consists of a fitness-rate-rank-based (FRR) credit assignment method that uses the fitness improvement rates obtained by the operators together with a decaying method (this is explained in section 2.4). For the operator selection, the default UCB [37] algorithm is used. They also combine their FRR credit assignment method with the Adaptive Pursuit [39] and Probability Matching [55] operator selections. We have also used this FRR credit assignment, but instead use a different approach for the operator selection. This approach is based on *Tournament Selection* (TS). Tournament selection is normally used in the selection process in evolutionary algorithms. Here, we have used it to select an operator. First, a number of operators are selected from the pool of operators at random. Then, the operator from this selection with the best quality will be used for application. The number of operators that are selected from the pool is known as the *tournament size* and can be used to control the exploitation and exploration of the operators. This is also known as the *selection pressure*. A tournament size of 1 is equal to choosing an operator at random from the operator pool without taking into account the quality. Therefore, it will fully explore without any exploitation. If the tournament

size is equal to the operator pool size, it will always select the operator with the best quality. This means that there is no exploration and that it will always fully exploit the best operator. If a value is used that is somewhere in between, it balances the exploration and exploitation. Where AP focuses more on the exploitation and PM more on the exploration of operators, TS does something in between. In this investigation, we have tried different tournament sizes and see how it compares to the implementations using AP and PM. Section 3.4 will elaborate on this.

3.2 Algorithms pseudo-code

The pseudo-code that is used in this research can be found in Algorithms 1, 2 and 3 in [41]. However, the operator selection algorithm (Algorithm 2 in [41]) is different here. The new procedures for Adaptive Pursuit (AP), Probability Matching (PM) & Tournament Selection (TS) can be found in Algorithms 1, 2 and 3, respectively.

Algorithm 1 The AP operator selection mechanism

/ Before MOEA/D-FRR-AP starts, the probability p_{op} is set to 1/K for every operator op, where K is the nr. of operators*/*

if Not all operators have been selected yet **then**
    op = uniformly randomly select an operator from the operators pool
else
    $op^* = \underset{op \in pool}{\operatorname{argmax}}(q_{op})$
    $p_{max} = 1 - (K-1)p_{min}$
    $p_{op^*} = p_{op^*} + \beta(p_{max} - p_{op^*})$
    $\forall op \neq op^* : p_{op} = p_{op} + \beta(p_{min} - p_{op})$
    op = choose operator using roulette wheel-like process using probabilities p
end if

Algorithm 2 The PM operator selection mechanism

if Not all operators have been selected yet **then**
    op = uniformly randomly select an operator from the operators pool
else
    for each op in $pool$ **do**
        $p_{op} = p_{min} + (1 - K \times p_{min}) \times \frac{q_{op}}{\sum_{i=1}^{K} q_i}$ (K = nr. of operators)
    end for
    op = choose operator using roulette wheel-like process using probabilities p
end if

Algorithm 3 The TS operator selection mechanism

if Not all operators have been selected yet **then**
    op = uniformly randomly select an operator from the operators pool
else
    ops = select k (tournament size) operators from the pool at random
    $op = \underset{op \in ops}{\operatorname{argmax}}(q_{op})$
end if

The pool consists of these four DE mutation operators:

1. **DE/rand/1**:
   $v^i = x^{r1} + F * (x^{r2} - x^{r3})$

2. **DE/rand/2**:
   $v^i = x^{r1} + F * (x^{r2} - x^{r3}) + F * (x^{r4} - x^{r5})$

3. **DE/current-to-rand/1**:
   $v^i = x^i + K * (x^{r1} - x^i) + F * (x^{r2} - x^{r3})$

4. **DE/current-to-rand/2**:
   $v^i = x^i + K * (x^{r1} - x^i) + F * (x^{r2} - x^{r3}) + F * (x^{r4} - x^{r5})$

Here $x^i = (x_1^i, \ldots, x_n^i)$ is the ith target vector, which is the solution that is currently being mutated. v^i is the mutant vector and $x^{r1}, x^{r2}, x^{r3}, x^{r4}$ and x^{r5} are random solutions from the neighborhood of x^i.

3.3 Software & Framework

For this research, we have used the jMetalPy framework (python) as a basis. This was convenient, as the MOEA/D-DRA algorithm was already implemented here, which is used to add the AOS mechanisms to. We then implemented the FRR credit assignment and the operator selection mechanisms: Adaptive Pursuit, Probability Matching & Tournament Selection. In jMetalPy, experiments could be run within this framework using the *Experiment* class. We also added some more code that could help to get more insight, such as the ability the keep track of the operators that were used by the AOS algorithms. The problems that were used to test the algorithms on, we have implemented ourselves. Lastly, we have written multiple unit tests for the algorithms and problems to minimize the risk of errors in the implementations.

3.4 Experiments

In order to see the impact of this new algorithm, we have done a number of experimental studies. First, we have compared the three algorithms (MOEA/D-FRR-TS, MOEA/D-FRR-AP and MOEA/D-FRR-PM) with each other using default parameter values. Then, a parameter analysis on MOEA/D-FRR-TS is performed to see if the algorithm is sensitive to different parameter values. Lastly, the use of a pool operator is compared with using a single operator. The results of the experiments can be found in the appendices at the end of this document.

3.4.1 Experiment 1: Comparison with Adaptive Pursuit and Probability Matching

We started off with comparing the new MOEA/D-FRR-TS with MOEA/D-FRR-AP and MOEA/D-FRR-PM from [41]. To do this, we ran these three algorithms 30 times on four unconstrained MOP problems: UF2, UF4, UF5 and UF8 from [54]. We have chosen these problems, since they have different characteristics that we can investigate, such as varying Pareto Fronts and a different number of objectives. These characteristics are given in table 1. Due to the fact that they are different in nature, we could see whether the impact of different algorithms is also dependent of the kind of problems that it is given. For this study, we use the same parameter values as [41]

Problem	Nr. of variables	Nr. of objectives	Pareto front
UF2	30	2	Convex
UF4	30	2	Concave
UF5	30	2	Linear, 21 points
UF8	30	3	Concave

Table 1: Characteristics of the investigated problems

uses for their experimental studies. Additionally, in MOEA/D-FRR-TS the tournament size will be set to 2. The results of this study will be discussed in section 4.1.

3.4.2 Experiment 2: Parameter analysis

We also wanted to see what happens if we change the parameters used in the adaptive operator selection part of the algorithm. The three control parameters of the AOS are the window size W, the tournament size S and the decaying factor D. In the last subsection of this method section, we explain that we are interested in the performance of the algorithms, but also in the dynamics of the operator selection. For the latter, a single run of MOEA/D-FRR-TS is performed using all different combinations of the following values for the three control parameters:

1. The tournament size S: 2 & 3
2. The window size W: 0.05*N, 2*N & 4*N [7]
3. The decaying factor D: 0.1 & 1.0

We used values that are far apart from each other on purpose, so that we can see how the operator selection is affected. Hereby, we will focus on the UF2 problem. The operator selection dynamics might differ from problem to problem, but the points that will be discussed in this experiment, will apply for every problem.

As will be explained in the results section, it turned out that MOEA/D-FRR-TS is unaffected by the decaying factor. Therefore, to measure the

[7]Note that the window size depends on the population size N. Since N is 600 for two-objective problems, the window sizes are 30, 1200 and 3000 here.

performance of using different parameter values, a parameter analysis is performed using varying values for S: 2 & 3 and W: $0.1*N$, $0.5*N$ & N [8], where we consider (S=2, W=$0.5*N$) as the default setting, The decaying factor remains 1.0 and the rest of the parameter values stay the same as in [41]. The algorithms are than run 30 times on the UF2 and UF4 problems. The results of this study will be discussed in section 4.2.

3.4.3 Experiment 3: Pool of operators vs. single operator

Lastly, it might be interesting to see if it is advantageous to use a pool of operators instead of just a single one. Therefore, we compared MOEA/D-FRR-TS using the pool of four DE mutation operators with four equivalent algorithms that uses only one of those four operators. Again, we ran those five algorithms 30 times on the UF2, UF4, UF5 and UF8 problems. The results of this study will be discussed in section 4.3.

3.4.4 Measurements

To be able to compare the algorithms, we used two performance indicators: the *hypervolume* (HV) and the *inverted generational distance* (IGD). These are both explained further in section 2.1.7. Briefly, the HV indicates the area that is dominated by the approximated pareto set (PS). The larger this value is, the better the PS. The IGD corresponds to how close the approximated PS is to the real PS. The closer this value to 0, the closer the approximated PS is to the real PS. Additionally, we also looked at the running time of the algorithms to compare their efficiency. To check whether of not one algorithm is actually significantly different from another, we have used an unpaired two samples t-test at a significance level of 0.05. In the first experiment, we have checked whether MOEA/D-FRR-AP and MOEA/D-FRR-PM perform significantly worse, equal or better than the new MOEA/D-FRR-TS algorithm. In the second experiment, the variants using other parameter values are tested on their significance with MOEA/D-FRR-TS using the default values. In the last experiment, we have investigated the significance of the single operator variants with using a pool of operators. These are all tested on both the HV and IGD.

[8] Note that the window size depends on the population size N. Since N is 600 for two-objective problems, the window sizes are 60, 300 and 600 here.

Besides comparing the algorithms on their performance in terms of the hypervolume or the IGD, we were also interested in the dynamics of the operator selection. To get a better understanding of how the AOS mechanisms select the operators during the search process, we kept track of the chosen operators. Since the algorithm terminated after 300.000 evaluations, the number of times that an operator is selected during the whole process is 300.000 as well. We divide this process into 50 phases, with each phase containing $300.000/50 = 6.000$ evaluations/operator selections. The number of times every operator is selected during a phase is plotted against the phases. We call this the *trajectory* of the AOS of a specific algorithm. It might be interesting to see how the trajectories differ for different algorithms. In experiment 3, the algorithms using a single operator select the same operator every time. Therefore, we are not interested in the operator selection trajectories and will not discuss them in that experiment. Instead, we have plotted the approximated PF and the real PF. This way, we can examine if different algorithms generate different kind of Pareto fronts and how this differs among various problems.

4 Results

4.1 Experiment 1: Comparison with Adaptive Pursuit and Probability Matching

4.1.1 Operator Dynamics

In Figure 9 the trajectories of UF2, UF4, UF5 and UF8 are plotted for MOEA/D-FRR-AP, MOEA/D-FRR-PM, MOEA/D-FRR-TS. First of all, AP seems to have some clear alternating phases in which one operator is dominating. In Figure 9j for example, in the first few phases, DE/current-to-rand/2 is mostly used. Then it is followed up by a number of phases where DE/current-to-rand/1 dominates, with occasionally DE/rand/1 in between. This is not really the case for PM and TS. Here, it is mostly one operator that is used the most during the whole process. Especially on UF2 and UF8, where this is the case for DE/current-to-rand/1.

Also, AP has constant minima and maxima, whereas for PM and TS, this varies from problem to problem. This can be explained by the fact that PM maps the qualities of the operators to a probability in a proportional manner. So, the higher the quality of an operator, the higher the probability of it being selected (and vice versa). Therefore, depending on the problem, the trajectories of the operators might be close together or further away. AP however, pushes the best operator towards a maximum probability (p_{max}) and the other operators towards a minimum probability (p_{min}). In this experiment, these values for p_{min} and p_{max} are 0.1 and 0.7, respectively. Since β in equations 27 and 28 is quite high (0.8), the probabilities are quite close to 0.1 and 0.7 as well. These probabilities corresponds to 0.1*6000 = 600 and 0.7*6000 = 4200 usages per phase. If we look at the minima and maxima of AP in figure 9, we can see that they are indeed close to 600 and 4200.

For UF2 and UF8, we can see that the trajectories of TS also tend to stick to certain values. The best operator fluctuates between 2000 and 2500, while the worst one ranges between about 500 and 1000. The others fluctuate around 1500 and they take turns in being the most used. This has to do with the fact that in MOEA/D-FRR-TS, the probability of choosing an operator is independent from the actual value of the quality. We will further explain this in section 3.4.2 (Parameter analysis).

Lastly, it is noticeable that for UF4 (and for UF5 a little bit as well), the trajectories of the graphs of all three algorithms are more intertwined than

for the other problems. Whereas with the other problems, DE/current-to-rand/1 seemed to be the best most of time, this is not that case for UF4. This indicates that this problem has certain characteristics such that during its optimization, less strong preferences are given to certain operators.

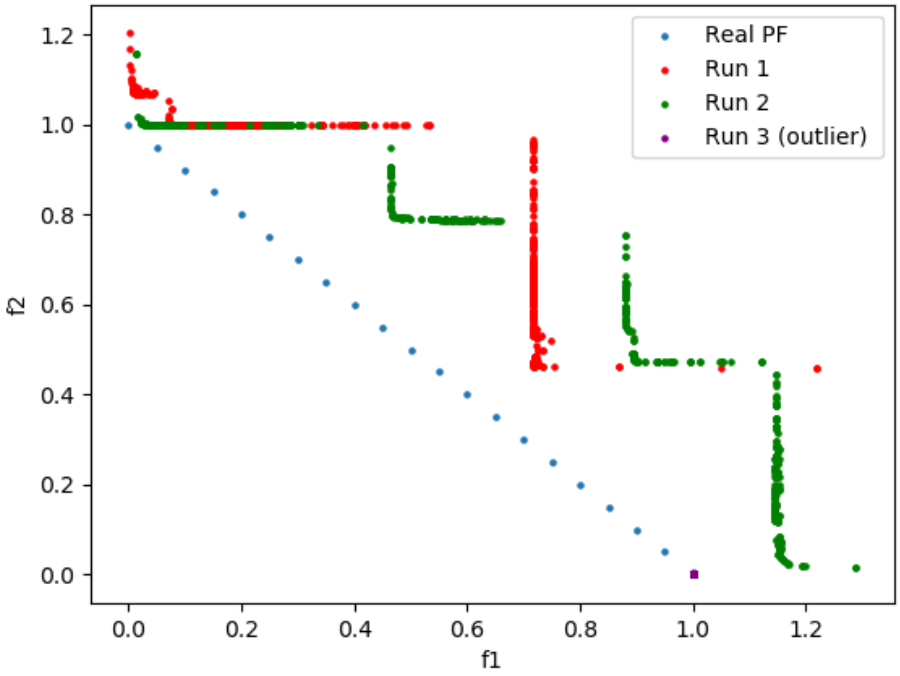
4.1.2 Performance

The box plots of the peformance of AP, PM and TS are given in Figures 10 (based on HV) and 11 (based on IGD). From these results, we can quickly conclude that, for the four concerned problems, there is no significant difference between AP and TS or PM and TS on the IGD and HV performance indicators. This indicates that these problems are not sensitive to the three tested probability based AOS types. In the next section, we will investigate whether or not they are sensitive to different parameter values.

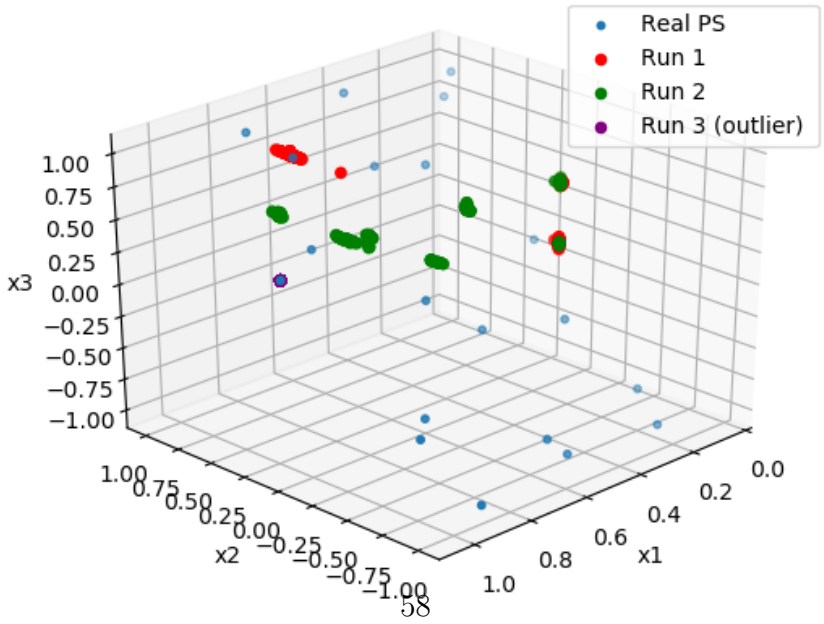
However, there is something noteworthy about UF5. Looking at the IGD (Figure 11c), we can see a couple of outliers around 0.70. To get a better insight of what is happening here, we plotted the approximated PFs and PSs of this outlier together with that of 2 other runs in Figure 5. It turns out that the solutions of this outlier (run 3) have converged to a single point in the variable space (Figure 5b). This point correspond to (1,0) in the objective space (Figure 5a), which is also a point on the real PF. The other two runs consist of chunks of points that each either vary in x_2 (and not in x_3) or in x_3 (and not in x_2). We know that x_2 is used in the calculation of f2 and x_3 in the calculation of f3. This is why the PF consists of a couple of vertical and horizontal strips of points.

It sounds pleasant that run 3 was able to produce solutions that were very close to optimal. However, all points of the approximated PS converged to this single optimal solution. So, there is absolutely no diversity. Even though runs 1 and 2 contain solutions that are not as close to a solution from the real PS as run 3 has, they do have a much better diversity. This is why runs 1 and 2 have a better IGD and HV, making them more desirable.

The average running times of the three algorithms for UF2, UF4, UF5 and UF8 are also plotted in Table 3. No significance is found, except for UF8, where AP is significantly faster than both PM and TS. In all cases, AP is faster than PM. This indicates that the probability updating approach of PM takes up more time than AP, at least in this implementation. The actual values of these running times should be taken with a grain of salt, since they are really dependent on the implementation and programming language.



(a)



(b)

Figure 5: The approximated PFs (a) and PSs (b) of three independent runs

4.2 Experiment 2: Parameter analysis

4.2.1 Operator Dynamics

As we said earlier, we are interested to see how the operator selection dynamics looks like. Especially in a parameter analysis, where we are evaluating different parameter values, we can expect the operator selection to be different among the parameter settings. The trajectories of MOEA/D-FRR-TS using different combinations of parameter values on problem UF2 are plotted in Figure 12. The influence of the three control parameters will now be discussed.

The window size

First of all, we will take a look at the difference of using various window sizes. To start off, the smaller the window size, the closer together the trajectories of the operators are. Take Figure 12c for example. In the first few phases, the best operator (DE/current-to-rand/1) has usages of about 1800 while the worst operator (DE/rand/2) has about 1200, a difference of 600. On the other hand, in Figure 12g this is resp. 2500 and 400, a difference of 2100. Presumably, this is because the smaller the window size, the closer the algorithm is to choosing its operators at random. To illustrate this, you first need to be aware of fact that with tournament selection, the probabilities of selecting a particular operator is solely based on the tournament size and its rank (in terms of quality). They are independent of the actual value of the quality. Therefore, we have created table 2 that shows the probabilities given a tournament size and rank.

Tournament Size	Rank	Probability	Nr. of usages (phase = 6000)
2	1	7/16	2625
2	2	5/16	1875
2	3	3/16	1125
2	4	1/16	375
3	1	37/64	3469
3	2	18/64	1687
3	3	8/64	750
3	4	1/64	94

Table 2: The probabilities and number of usages of selecting an operator in a pool of four operators given a rank and tournament size. An operator with the best quality has a rank of 1, the second best a rank of 2, etc.

Now consider MOEA/D-FRR-TS using a pool of operators with window size 1 and tournament size 2. At every moment in time, the operator selection will be based solely on the previous operator application, because of the window size of 1. Therefore, the corresponding operator will automatically have the best quality. From Table 2 we can tell that this operator has a probability of 7/16 to be selected next. The other operators have an equal probability of being chosen, namely $(9/16)/3 = 3/16$. These probabilities hold for every operator. This means that there is no competition between operators at any point; the one that is applied in the previous time slot, will be applied with a fixed probability for the next one. To prove this, we have run the MOEA/D-FRR-TS algorithm using a window size of 1 and plotted the trajectories in Figure 6 on UF2. It is clear that they have more or less the same amount of usages.

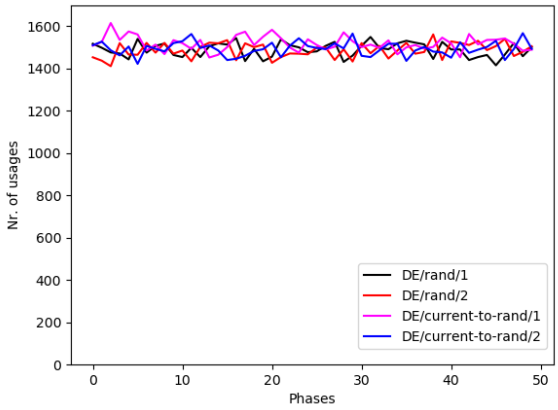


Figure 6: The operator usage trajectory when a window size of 1 is used.

If a large window is used however, the operator's quality is based on many observations. Therefore, it is less likely for operators to catch up with each other, and if they do, the new obtained order of qualities remain for a longer time.

Also, you can see that the trajectories when using large window sizes tend to stick to certain values. In Figure 12k for example, these values are at about 2600, 1800, 1200 and 400. These correspond with the values from table 2 (tournament size of 2). So. the smaller the window size, the more it resembles choosing operators at random. If the windows gets bigger, the more operator usages will stick to fixed values.

In addition to this, the trajectories belonging to implementations that use smaller windows tend to converge to the same value. This might be explained by the fact that in the beginning of the evolutionary process, there is more room for improvement. So, the difference between better and worse performing operators gets larger, resulting in more widely separated usages. Later on in the process, many new generated solutions will be worse than the solution it is compared with. The applications will often receive no credit whereby good operators cannot excel as much. To check whether or not the trajectories indeed converge more towards the same value, we have run the algorithm for 4.000.000 evaluations and plotted this in Figure 7. As we can see from the figure, the graphs indeed converge to the same value, namely 20.000, which makes sense. Since, the number of phases are 50, the number of operator selections per phase is $4.000.000/50 = 80.000$. And since there are

4 operators used here, the number of usages per operator would be 80.000/4 = 20.000. We can notice that there are some bumps here and then where the number of usages of the best operator suddenly starts increasing for a few phases. The usages of the worst one decreases here. Presumably, the algorithm got out of some local optimum at those points of time and found a bunch of new improved solutions with the use of the DE/current-to-rand/1 operator.

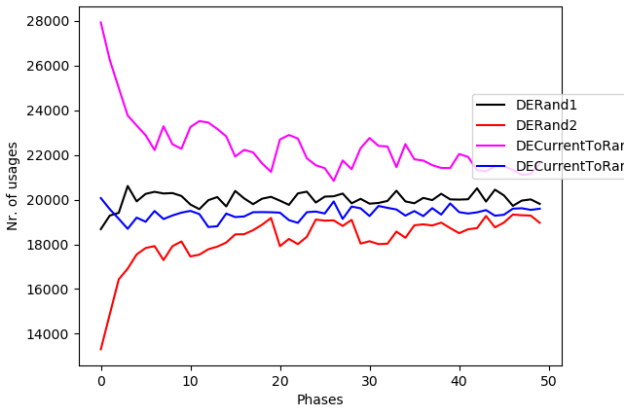


Figure 7: The operator usage trajectory after running FRR-TS on UF2 for 4 million evaluations

Tournament size

Just like with using different window sizes, the tournament size influences the range between the trajectories. Whereas in the first few phases of Figure 12c the difference is about 600, in Figure 12d this is equal to circa 1100. This can be explained by the fact that a larger tournament size leads to more extreme probabilities. Let's look at Table 2 again. When changing the tournament size from 2 to 3, the probabilities of the best operator (rank 1) is increased by more than 30 percent, while the other operators have a decreasing probability. Especially the worst performing operator's probability drops down dramatically, namely from 1/16 to 1/64.

Decaying factor

We also analyzed the effect of using different decaying factors towards the operator usages. However, by looking at the graphs in Figure 12, we did not found any significant differences. If we think about it, this actually makes sense. The tournament selection that is used in MOEA/D-FRR-TS selects its operators based solely on their rankings, so the actual difference in qualities between the operators does not matter. If operator a has a better quality than operator b, no matter how much decaying is applied, a's quality will always remain better than b's. For AP and PM, the decaying factor would matter. The probability of selecting an operator is (partly) proportional to the quality. Thus, increasing the decaying factor would increase the selection probability and therefore the exploitance of the best operator.

4.2.2 Performance

The parameter settings are also compared on their performance. As we concluded from the previous section, the decaying factor has no influence on the actual behaviour of the algorithm. Therefore, we will ignore this parameter in this section and leave it at the default value of 1.0. We consider the following parameter setting as default: W=300, S=2. The parameter values were tested on problems UF2 and UF4 and the results are plotted as box plots in Figure 13. From these results, we can conclude that there is no significant difference between any parameter setting that is tested and the default parameter setting, except for (S=3, W=600), which is significantly worse on the IGD. We can see that the IGD is already quite close to 0, indicating that an improvement might be difficult to achieve. On the other hand, MOEA/D-FRR-TS using other parameter settings do not perform much worse either. Apparently, UF2 and UF4 are not that sensitive to these parameters when using MOEA/D-FRR-TS.

Table 4 shows us the running times of the different parameter values, tested on UF2 and UF4. It is clear that a larger window size leads to an increase of running time. This is due to the fact that every time it calculates the quality of an operator, it needs to loop through the whole window. Of course, this depends on the implementation and it might be possible to do this in a more elegant way, such that the computational time is independent of the window size. It seems that a larger tournament size also raises the running time. Except for a few extra insignificant executions the algorithm

has to perform, we could not find a clear reason for this.

4.3 Experiment 3: Pool of operators vs. single operator

4.3.1 Performance

In Figures 14 and 15, box plots are given for the comparison of MOEA/D-FRR-TS using a pool of operators and MOEA/D using only a single operator from the pool. We can see that for problems UF2, UF4 and UF5, DE/current-to-rand/1 is significantly worse compared to using a pool in terms of both the HV and IGD. It is also the worst among all the single-operator algorithms. For the three-objective problem UF8, this is the case for DE/rand/2. These findings correspond with the results from Figure 6 and 7 in [41]. For UF8, DE/rand/1 is significantly worse than the pool variant as well for both IGD and HV. When looking only at the IGD on UF2, the pool variant also significantly outperforms DE/rand/1 and DE/rand/2. Lastly, we can see that there are cases in which a single operator performs significantly better than the pool variant. This applies to DE/rand/1 in UF5 and DE/rand/2 in UF4 and UF5.

4.3.2 Pareto Front approximation

We also plotted the approximated PFs of the 30 runs of each algorithm that were performed in Figure 16. Starting with UF2, they all perform quite well in terms of PF approximation; we cannot even see the real PF (the blue line) anymore. However, all algorithms seem to have some difficulty in approximating the real PF at the bottom right. This could be caused by what [18] addresses about the F5 problem (which is the same as UF2) they tested their algorithms on. According to them, this could have to do with the weight vectors. The weight vectors that are used (and in our experiments as well) are evenly spread along the PF. This means that the optimal solutions to each subproblem belonging to two neighboring weight vectors will be close on the PF (the objective space). However, this does not always mean that they are close in the variable space as well. Figure 8 shows the optimal solutions of 50 subproblems that are evenly spread along the PF. As you can see, as x_1 approaches 1, the solutions are not that close anymore. This means that combining two of these neighboring solutions to generate a new solutions is

not that useful and as [18] describes it: "*makes little sense*". As they suggest, another weight vector initialization could improve the approximation of PFs in problems like UF2.

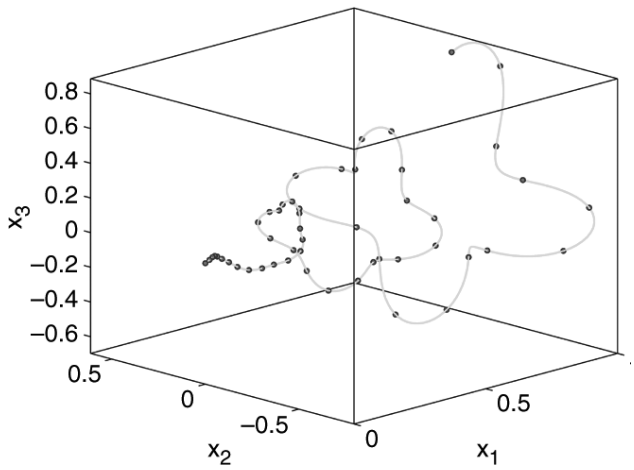


Figure 8: Distribution of the optimal solutions of 50 subproblems with evenly spread weight vectors in x_1 - x_2 - x_3 space [18]

Going back to our results, we can conclude that not every algorithm is affected just as much by this problem. Especially DE/current-to-rand/1 and DE/rand/1 seem to have struggles, whereas the others are doing a lot better. As [41] points out, DE/*/2 operators have two random-to-random terms, while DE/*/1 operators only have one. This means that DE/*/2 operators can do more exploration, which could be how these operators partially avoid the problem described above.

In Fig. 16 we can also see that for UF4 and UF5, DE/current-to-rand/1 has points that are further from the real PF than the others. Furthermore, for UF4, the points of DE/rand/2 are clearly the closest to the real PF, indicating that this operator performed the best. These findings for UF2 and UF4 match the results from earlier (Figs. 14 and 15).

Lastly, if we take UF8, it is noticeable that all the plots have a gap in top right corner that contain few points, indicating a flaw in the diversity of the approximated PF. However, the gap of DE/rand/2 is significantly larger than that of the others and contains more points bunched up in the direction of the corners of the PF. This means that this operator is less diverse than the others. This matches the fact that DE/rand/2 also performs the worst in

terms of IGD, as discussed earlier. DE/rand/1 also performed significantly worse, however this is not that clear from the plots in Figure 16. This is likely because of the fact that the differences are less significant.

We also wanted to know if there is a link between the performance of the pool variant and the amount of improvement that is left on average of all the approximated PFs. Therefore, table 5 shows per problem which single operator variants perform better, worse, or equal to MOEA/D-FRR-TS using a pool of operators. The problems are sorted on how much IGD improvement is left on average, with UF5 having the most improvement left. We can see straight away that less improvement leads to a better performance of the pool compared to the others. A reason for this could be that for a problem like UF2, for which the algorithms can produce PFs close to optimal, it needs different operators to reach those last small improvements. An algorithm using a single operator is perhaps not able to reach certain solutions which a set of multiple operators can.

To conclude, these results indicate that MOEA/D-FRR-TS using a pool of operators is not superior on all test cases. Also, there is no single operator variant that performs the best on all problems. For example, DE/rand/2 does perform significantly better on UF4, but is the worst on UF8. Furthermore, DE/current-to-rand/2 is the only single operator variant that performed equally to the pool for all test cases.

5 Discussion & Conclusion

5.1 Discussion & Future Work

We have now seen the results of three experiments regarding the performance of MOEA/D-FRR-TS. These lead to a few points of discussion. In this section, we will work these out, as well as providing possible improvements and future work.

To start off, from the trajectories of the first experiment we have concluded that one operator (DE/current-to-rand/1) dominates during the processes of UF2 and UF8. This leads to the expectation that in Experiment 3, MOEA/D using this operator is also superior to the others for these two problems. The results from Experiment 3 show that for UF8 DE/current-to-rand/1 indeed performs better than the other single operator variants. However, this is not the case for UF2. Here, it is even significantly worse than all the other ones. The fact that it is outperformed by the pool variant, can be explained. It could be that during the process DE/current-to-rand/1 is constantly the better operator, but other operators might be necessary to find solutions that could not be reached by using only DE/current-to-rand/1. However, it is hard to say why all the single operator variants perform better.

Furthermore, we have concluded that all four tested problems are not sensitive to what type of operator selection is used (AP, PM or TS). Among the problems, UF2 and UF4 are also not sensitive to the parameter settings that were tested for MOEA/D-FRR-TS. For UF2, we have seen that little improvement is left, indicating the easiness of this problem. One could argue that therefore, it does not really matter which operator selection is used, since it finds a good PF approximation anyways. However, the approximations for the other problems are much further from optimal. And yet, the operator selection makes no difference. It could be that for other problems (or using other operator selections) there is significance in the type of operator selection that is used.

Nevertheless, our results show that it does matter which operators are used. It could therefore be interesting to do more research in the use of different operators. Perhaps trying different combinations of certain operators, or experiment with other types of operators. For example, operators that mutate a solution using the best operator from the current population, such as DE/best/* and DE/current-to-best/* [56]. According to [46], these types of operators have a higher chance of getting stuck at local optima, but also

have a higher convergence speed. So, instead of using a maximum amount of evaluations as termination criteria, we could check when the algorithms using different operators reach a certain hypervolume. We could then investigate how fast they actually converge.

Overall, the research in this paper can be pursued by more extensive testing, such as using more problems. Problems with other characteristics could be used, such as those having more objectives, different amounts of variables or other Pareto fronts. Also, constrained problems could be tested as well, such as those from [54]. As [57] denotes, constrained problems introduce other types of Pareto fronts, such as disconnected Pareto fronts. This leads to difficulties in the PF approximation and might give a better picture of the strength of an algorithm, which opens up more possibilities to determine the difference between various algorithms. Furthermore, the parameter analysis could be more exhaustive. Instead of only trying out different parameter settings for the control parameters of the operator selection, parameters of MOEA/D and the credit assignment (FRR) could be alternated too. On top of that, if more operators are added to the pool, larger values for the window size and tournament size could be tried as well.

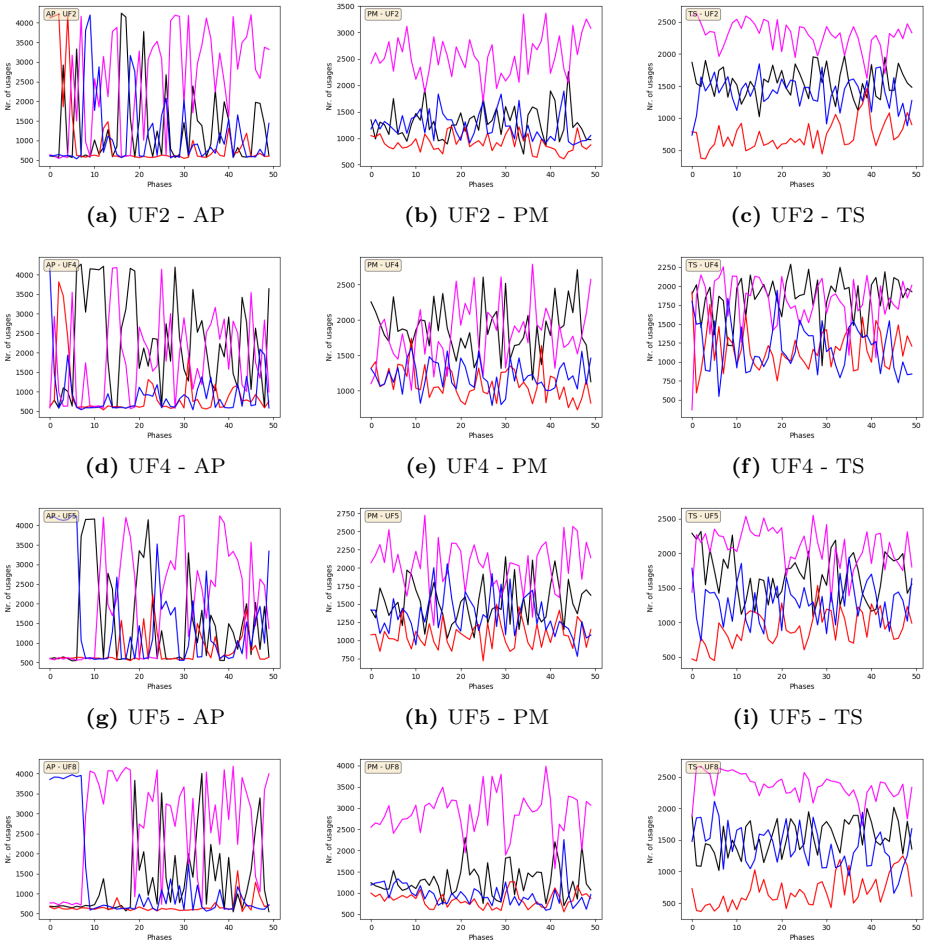
5.2 Conclusion

This paper has given an overview of existing Adaptive Operator Selection (AOS) mechanisms and how these are applied to Multi-Objective Evolutionary Algorithms. Also, a new algorithm is proposed, called MOEA/D-FRR-TS. This algorithm chooses mutation operators from a pool of operators using an operator selection that is based on tournament selection. First, this new algorithm was compared to two other algorithms using another operator selection mechanism based on Adaptive Pursuit (MOEA/D-FRR-AP) and Probability Matching (MOEA/D-FRR-PM). Then, a parameter analysis of the control parameters was performed. Finally, the algorithm was compared with MOEA/D that uses just a single operator for the reproduction. Results clearly show different behaviour in the operator selection among the various algorithms. Nevertheless, no significance was found between MOEA/D-FRR-AP or MOEA/D-FRR-PM and MOEA/D-FRR-TS, nor between the different parameter settings. The results did show that using MOEA/D-FRR-TS with a pool of operators can improve MOEA/D that uses just a single operator, depending on the problem. This indicates that more research in using other types of operators might be worth it. Furthermore, the studies in this

paper include a limited set of problems that were used to test the algorithms. Therefore, it could be interesting to see what happens if other problems were used, especially with different characteristics.

Appendices

A Experiment 1



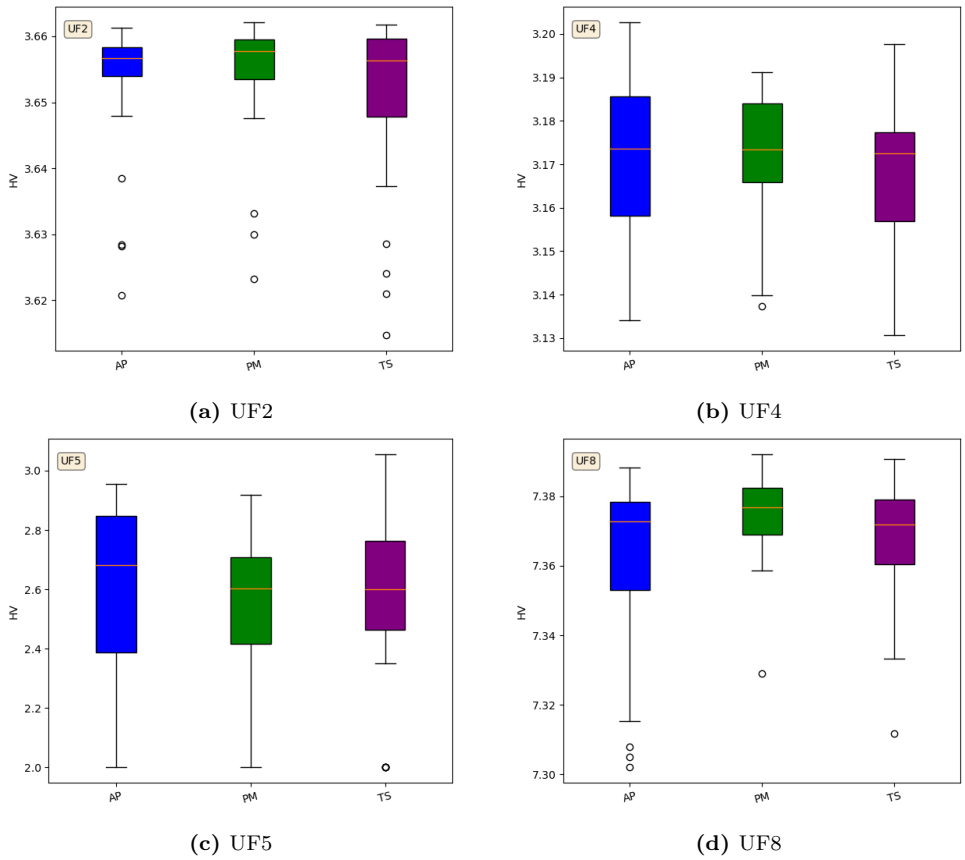
(a) UF2 - AP **(b)** UF2 - PM **(c)** UF2 - TS

(d) UF4 - AP **(e)** UF4 - PM **(f)** UF4 - TS

(g) UF5 - AP **(h)** UF5 - PM **(i)** UF5 - TS

(j) UF8 - AP **(k)** UF8 - PM **(l)** UF8 - TS

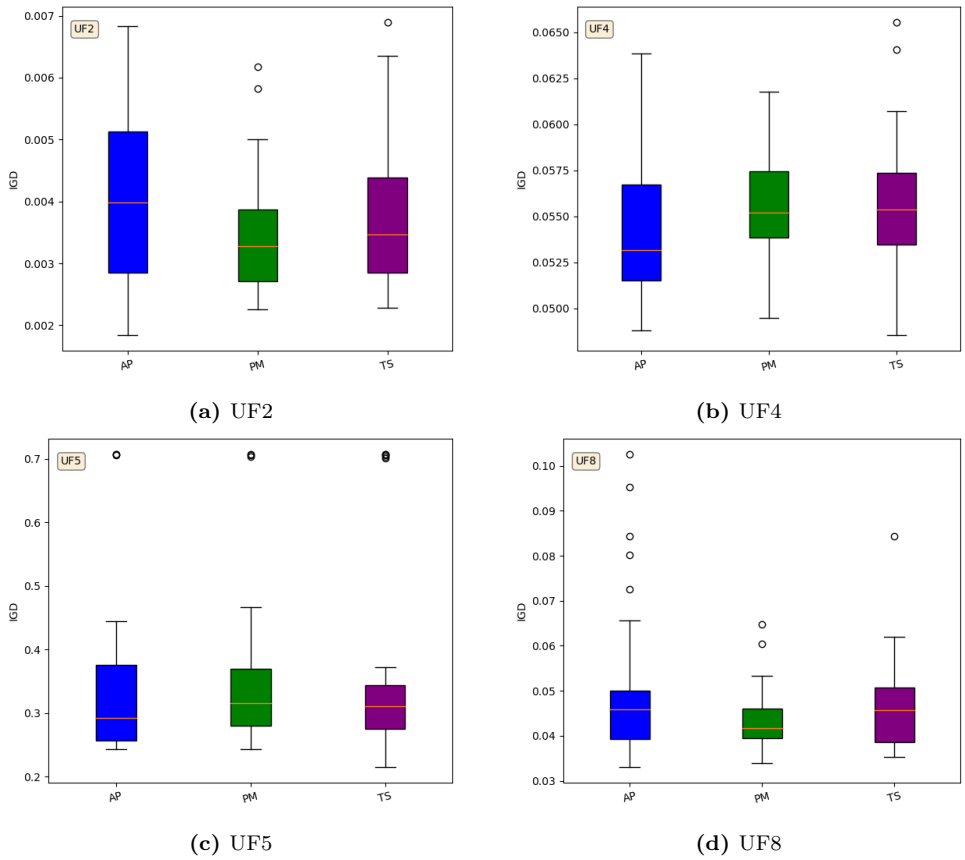
Figure 9: The operator selection trajectories of MOEA/D-FRR-AP, MOEA/D-FRR-PM & MOEA/D-FRR-TS. DE/rand/1 (——), DE/rand/2 (——), DE/current-to-rand/1 (——), DE/current-to-rand/2 (——).



(a) UF2 **(b)** UF4

(c) UF5 **(d)** UF8

Figure 10: Box plots of the performances of MOEA/D-FRR-AP, MOEA/D-FRR-PM & MOEA/D-FRR-TS based on the HV. Both MOEA/D-FRR-AP and MOEA/D-FRR-PM are compared with MOEA/D-FRR-TS using an unpaired two samples t-test with a significance level of 0.05. Significance is notated with a '+' or a '-' above the algorithm if it is significantly better or worse than MOEA/D-FRR-TS, resp.



(a) UF2 **(b)** UF4

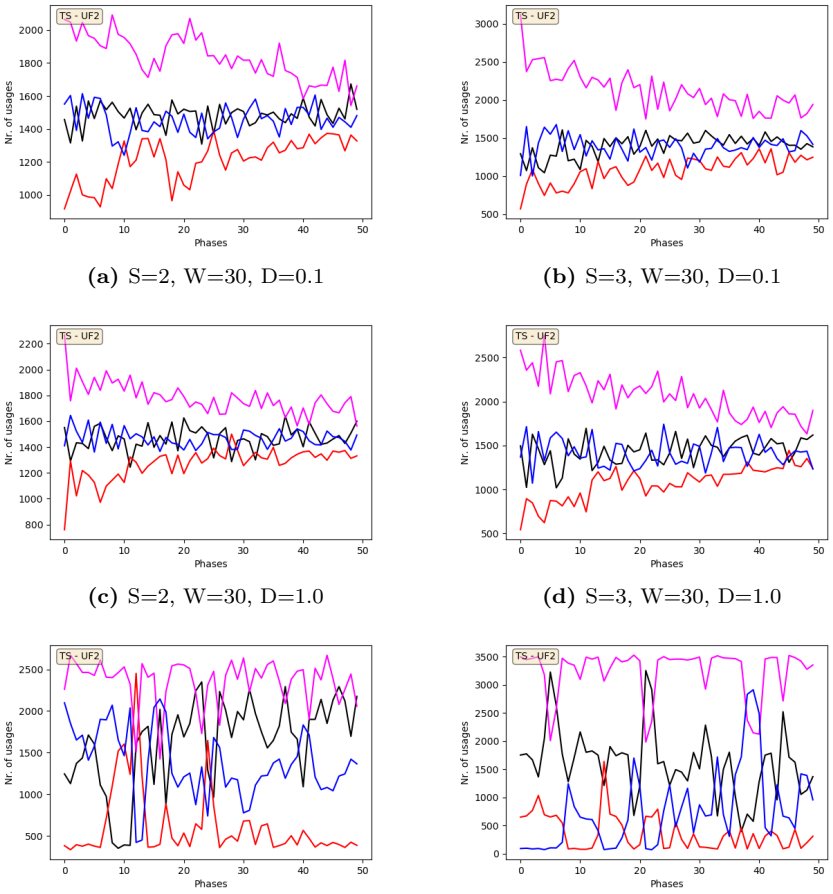
(c) UF5 **(d)** UF8

Figure 11: Box plots of the performances of MOEA/D-FRR-AP, MOEA/D-FRR-PM & MOEA/D-FRR-TS based on the IGD. Both MOEA/D-FRR-AP and MOEA/D-FRR-PM are compared with MOEA/D-FRR-TS using an unpaired two samples t-test with a significance level of 0.05. Significance is notated with a '+' or a '-' above the algorithm if it is significantly better or worse than MOEA/D-FRR-TS, resp.

Algorithm	**UF2**	**UF4**	**UF5**	**UF8**
MOEA/D-FRR-AP	523	521	525	831
MOEA/D-FRR-PM	532	529	539	856
MOEA/D-FRR-TS	529	525	522	853

Table 3: The average running time of MOEA/D-FRR-TS, MOEA/D-FRR-AP & MOEA/D-FRR-PM

B Experiment 2



(a) S=2, W=30, D=0.1

(b) S=3, W=30, D=0.1

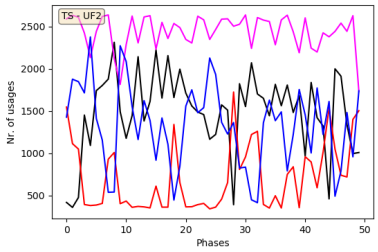
(c) S=2, W=30, D=1.0

(d) S=3, W=30, D=1.0

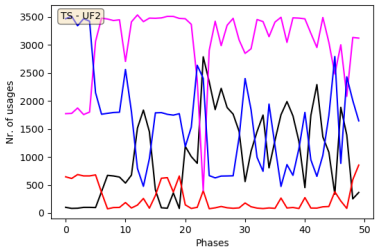
(e) S=2, W=1200, D=0.1

(f) S=3, W=1200, D=0.1

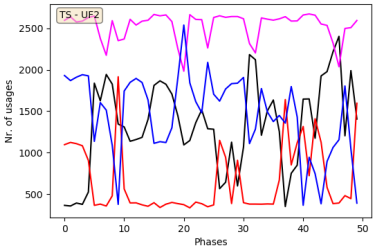
Figure 12: The operator selection trajectories of MOEA/D-FRR-TS using different parameter settings for the tournament size (S), the window size (W) and the decaying factor (D) on UF2. DE/rand/1 (——), DE/rand/2 (——), DE/current-to-rand/1 (——), DE/current-to-rand/2 (——).



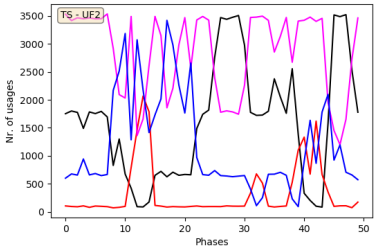
(g) S=2, W=1200, D=1.0



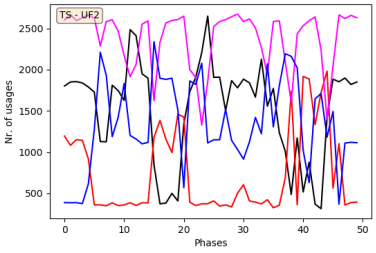
(h) S=3, W=1200, D=1.0



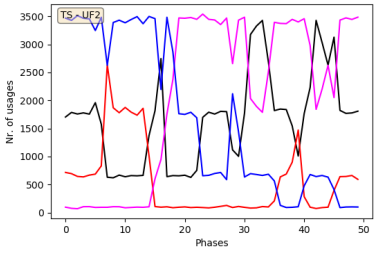
(i) S=2, W=3000, D=0.1



(j) S=3, W=3000, D=0.1

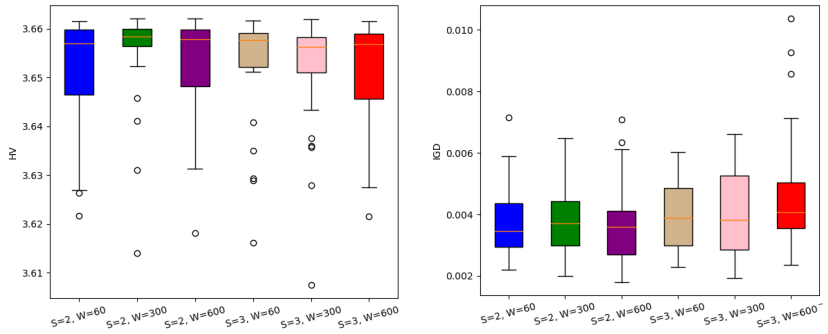


(k) S=2, W=3000, D=1.0



(l) S=3, W=3000, D=1.0

Figure 12: The operator selection trajectories of MOEA/D-FRR-TS using different parameter settings for the tournament size (S), the window size (W) and the decaying factor (D) on UF2. DE/rand/1 (—), DE/rand/2 (—), DE/current-to-rand/1 (—), DE/current-to-rand/2 (—) (**cont.**)



(a) UF2 - HV **(b)** UF2 - IGD

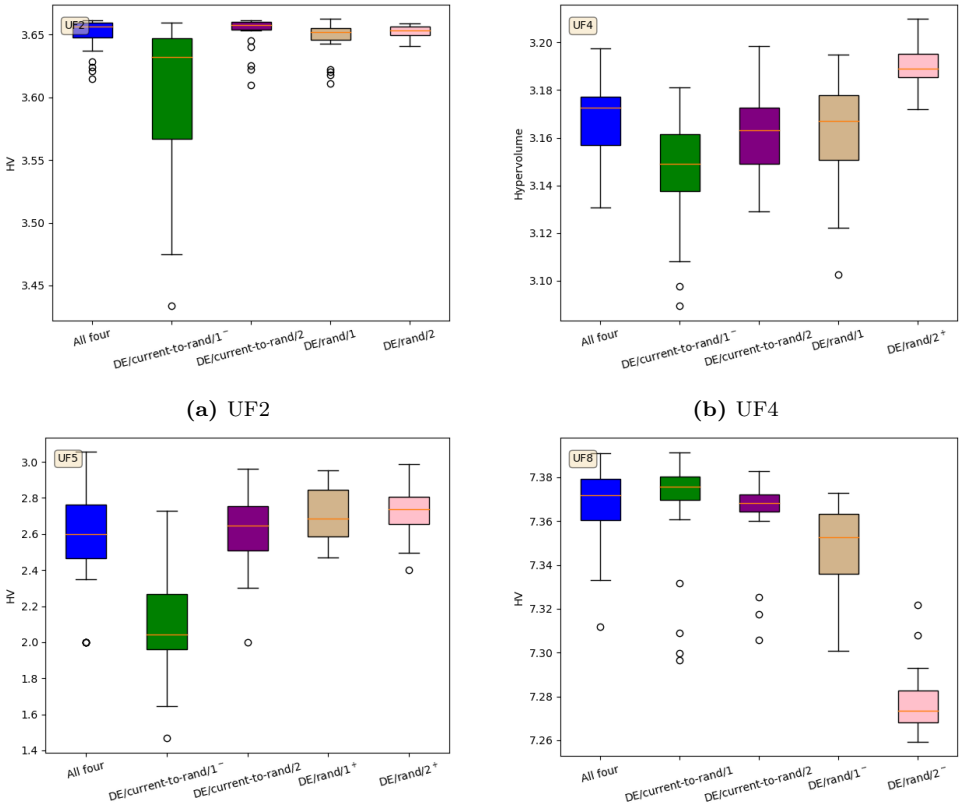
(c) UF4 - HV **(d)** UF4 - IGD

Figure 13: Box plots of the performance of MOEA/D-FRR-TS using different values for the tournament size (S) and the window size (W). Tested on UF2 & UF4, using the Hypervolume (a & c) and the IGD (b & d). An unpaired two samples t-test is performed using a significance level of 0.05 between MOEA/D-FRR-TS using different parameter values and the default parameter values (S=2, W=300). Significance is notated with a '+' or a '-' above the algorithm if it is significantly better or worse than the default, resp.

Tournament size	**Window size**	**Time UF2**	**Time UF4**
2	60	488	478
2	300	544	532
2	600	614	601
3	60	507	474
3	300	557	530
3	600	631	595

Table 4: The average running time of MOEA/D-FRR-TS with different parameter values for the Window Size and Tournament Size on UF2 and UF4

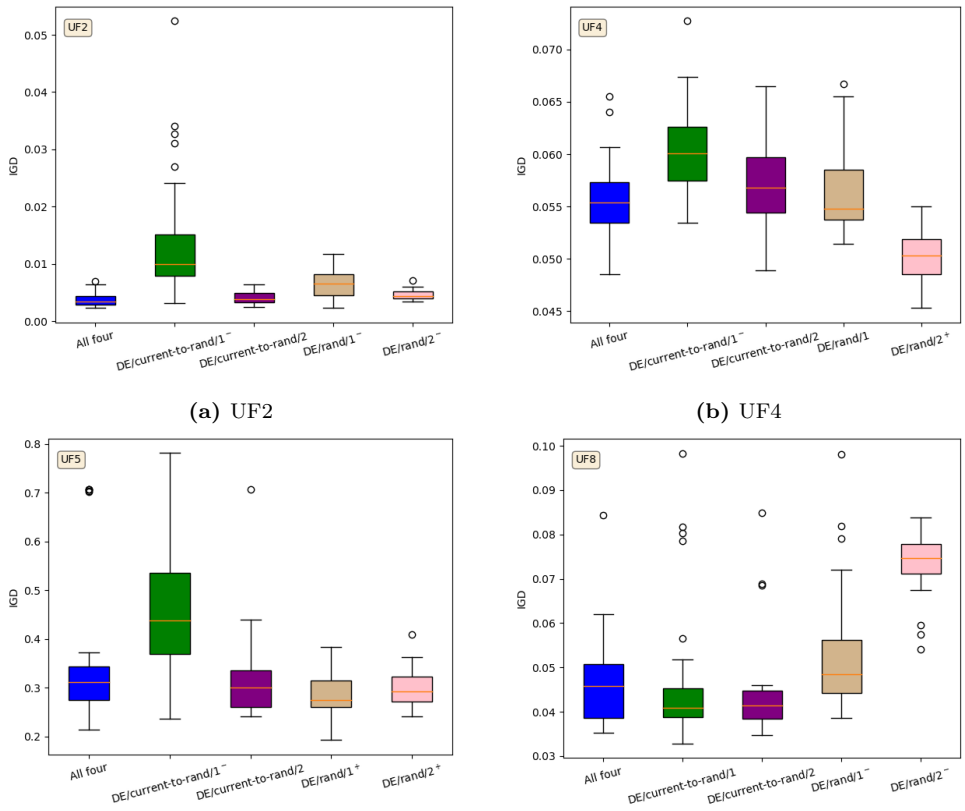
C Experiment 3



(a) UF2 **(b)** UF4

(c) UF5 **(d)** UF8

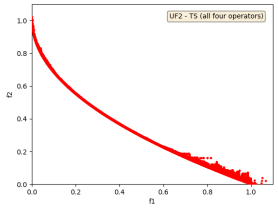
Figure 14: Box plots of the performance of MOEA/D-FRR-TS using single operators vs. a pool with all four based on the HV. An unpaired two samples t-test is performed using a significance level of 0.05 between each of the single variants and MOEA/D-FRR-TS using a pool. Significance is notated with a '+' or a '-' above the algorithm if it is significantly better or worse than MOEA/D-FRR-TS, resp.



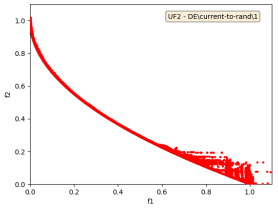
(a) UF2 **(b)** UF4

(c) UF5 **(d)** UF8

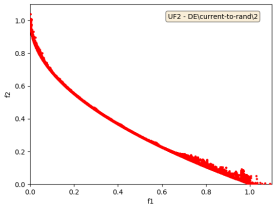
Figure 15: Box plots of the performance of MOEA/D-FRR-TS using single operators vs. a pool with all four based on the IGD. An unpaired two samples t-test is performed using a significance level of 0.05 between each of the single variants and MOEA/D-FRR-TS using a pool. Significance is notated with a '+' or a '-' above the algorithm if it is significantly better or worse than MOEA/D-FRR-TS, resp.



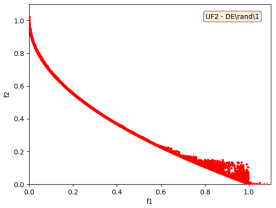
(a) UF2 - All four



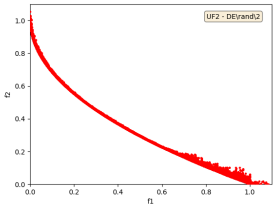
(b) UF2 - DE/current-to-rand/1



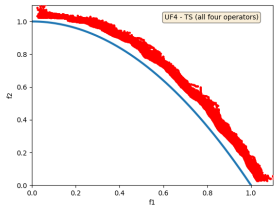
(c) UF2 - DE/current-to-rand/2



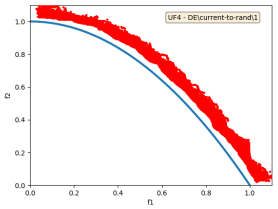
(d) UF2 - DE/rand/1



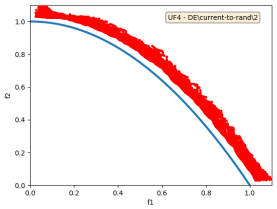
(e) UF2 - DE/rand/2



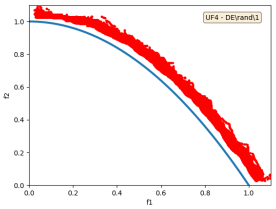
(f) UF4 - All four



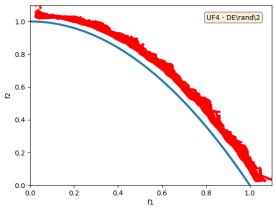
(g) UF4 - DE/current-to-rand/1



(h) UF4 - DE/current-to-rand/2

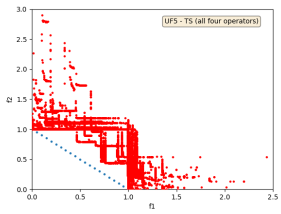


(i) UF4 - DE/rand/1

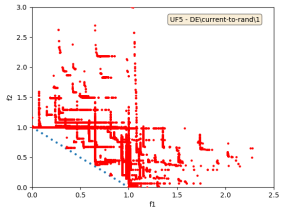


(j) UF4 - DE/rand/2

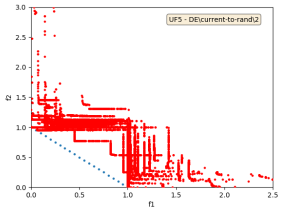
Figure 16: All the 30 approximated PFs of using a pool of operators and each of the single operators for problem UF2 & UF4.



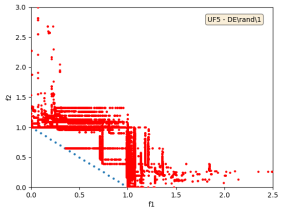
(k) UF5 - All four



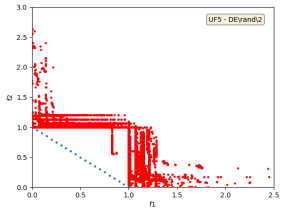
(l) UF5 - DE/current-to-rand/1



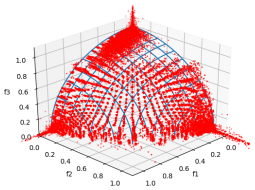
(m) UF5 - DE/current-to-rand/2



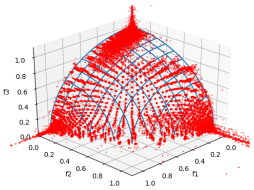
(n) UF5 - DE/rand/1



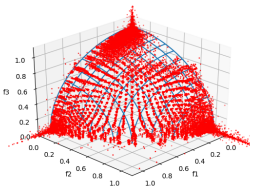
(o) UF5 - DE/rand/2



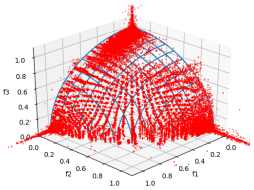
(p) UF8 - All four



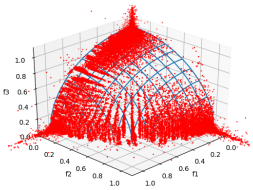
(q) UF8 - DE/current-to-rand/1



(r) UF8 - DE/current-to-rand/2



(s) UF8 - DE/rand/1



(t) UF8 - DE/rand/2

Figure 16: All the 30 approximated PFs of using a pool of operators and each of the single operators for problem UF5 & UF8 (**cont.**)

	Better than pool	Not significant	Worse than pool
UF5	DE/rand/1 DE/rand/2	DE/current-to-rand/2	DE/current-to-rand/1
UF4	DE/rand/2	DE/current-to-rand/2 DE/rand/1	DE/current-to-rand/1
UF8	-	DE/current-to-rand/1 DE/current-to-rand/2	DE/rand/1 DE/rand/2
UF2	-	DE/current-to-rand/2	DE/current-to-rand/1 DE/rand/1 DE/rand/2

Table 5: The problems sorted on how much improvement is left in terms of IGD (UF5 has the most improvement left) and which single operator variants perform worse, better or equal than MOEA/D-FRR-TS using a pool of operators.

References

[1] K. Deb, "Multi-Objective Optimization Using Evolutionary Algorithms : An Introduction", in *Multi-objective evolutionary optimisation for product design and manufacturing*, 2011, pp. 3–34.

[2] A. Trivedi, D. Srinivasan, K. Sanyal and A. Ghosh, "A survey of multiobjective evolutionary algorithms based on decomposition", *IEEE Transactions on Evolutionary Computation*, vol. 21, no. 3, pp. 440–462, Jun. 2016.

[3] Á. Fialho, L. da Costa, M. Schoenauer and M. Sebag, "Analyzing bandit-based adaptive operator selection mechanisms", *Annals of Mathematics and Artificial Intelligence*, vol. 60, no. 1-2, pp. 25–64, 2010.

[4] A. Zhou, B.-y. Qu, H. Li, S.-z. Zhao and P. Nagaratnam, "Multiobjective evolutionary algorithms : A survey of the state of the art", *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011.

[5] D. Savic, "Single-objective vs. multiobjective optimisation for integrated decision support", *Proceedings of the First Biennial Meeting of the International Environmental Modelling and Software Society*, vol. 1, pp. 7–12, 2002.

[6] B. Suman and P. Kumar, "A survey of simulated annealing as a tool for single and multiobjective optimization", *Journal of the Operational Research Society*, vol. 57, no. 10, pp. 1143–1160, 2006.

[7] C. A. Coello and M. Reyes-Sierra, "Multi-Objective Particle Swarm Optimizers: A Survey of the State-of-the-Art", *International Journal of Computational Intelligence Research*, vol. 2, no. 3, pp. 287–308, 2006.

[8] J. D. Schaffer, "Multiple Objective Optimization with Vector Evaluated Genetic Algorithms.", *Genetic Algorithms and Their Applications: Proceedings of the First International Conference on Genetic Algorithms*, pp. 93–100, 1985.

[9] J. D. Knowles and D. W. Corne, "Approximating the nondominated front using the Pareto Archived Evolution Strategy.", *Evolutionary computation*, vol. 8, no. 2, pp. 149–172, 2000.

[10] E. Zitzler, M. Laumanns and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm", *TIK-report*, vol. 103, 2001.

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

[12] K. Ikeda, H. Kita and S. Kobayashi, "Failure of Pareto-based MOEAs: Does non-dominated really mean near to optimal?", *Proceedings of the 2001 Congress on Evolutionary Computation*, vol. 2, pp. 957–962, 2001.

[13] H. Lshibuchi, Y. Sakane, N. Tsukamoto and Y. Nojima, "Simultaneous use of different scalarizing functions in MOEA/D", *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10*, pp. 519–526, 2010.

[14] E. Zitzler and S. Künzli, "Indicator-based selection in multiobjective search", *International Conference on Parallel Problem Solving from Nature*, pp. 832–842, 2004.

[15] M. Emmerich, N. Beume and B. Naujoks, "An EMO algorithm using the hypervolume measure as selection criterion", *International Conference on Evolutionary Multi-Criterion Optimization*, vol. 3410, pp. 62–76, 2005.

[16] D. Brockhoff and E. Zitzler, "Improving Hypervolume-based Multiobjective Evolutionary", *Evolutionary Computation*, pp. 2086–2093, 2007.

[17] J. Bader and E. Zitzler, "HypE: An algorithm for fast hypervolume-based many-objective optimization", *Evolutionary Computation*, vol. 19, no. 1, pp. 45–76, 2011.

[18] H. Li and and Q. Zhang, "Multiobjective Optimization Problems With Complicated Pareto Sets, MOEA/D and NSGA-II", *IEEE Transactions on Evolutionary Computatione*, vol. 13, no. 2, p. 1138, 2009.

[19] Y. Tian, X. Zhang, R. Cheng and Y. Jin, "A multi-objective evolutionary algorithm based on an enhanced inverted generational distance metric", in *2016 IEEE Congress on Evolutionary Computation (CEC)*, IEEE, 2016, pp. 5222–5229.

[20] M. P. Hansen and A. Jaszkiewicz, "Evaluating the quality of approximations to the non-dominated set", *IMM Technical Report IMM-REP-1998-7*, p. 31, 1998.

[21] Y. Tian, R. Cheng, X. Zhang, F. Cheng and Y. Jin, "An Indicator-Based Multiobjective Evolutionary Algorithm with Reference Point Adaptation for Better Versatility", *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 4, pp. 609–622, 2018.

[22] D. H. Phan and J. Suzuki, *R2-IBEA: R2 indicator based evolutionary algorithm for multiobjective optimization*. 2013, pp. 1836–1845.

[23] E. Zitzler and L. Thiele, "Multiobjective Evolutionary Algorithms : A Comparative Case Study", *IEEE Transactions on Evolutionary Computation*, vol. 3, no. September, pp. 257–271, 1998.

[24] D. Van Veldhuizen, "Multiobjective evolutionary algorithms: classifications, analyses, and new innovations", *AIR FORCE INST OF TECH WRIGHT-PATTERSONAFB OH SCHOOL OF ENGINEERING*, 1999.

[25] P. A. Bosman and D. Thierens, "The balance between proximity and diversity in multiobjective evolutionary algorithms", *IEEE Transactions on Evolutionary Computation*, vol. 7, no. 2, pp. 174–188, 2003.

[26] Q. Zhang and H. Li, "MOEA/D: A multiobjective evolutionary algorithm based on decomposition", *IEEE Transactions on Evolutionary Computation*, vol. 11, no. 6, pp. 712–731, Dec. 2007.

[27] K. Miettinen, *Nonlinear multiobjective optimization*. Springer Science & Business Media, 2012, vol. 12.

[28] H. Ishibuchi, Y. Sakane, N. Tsukamoto and Y. Nojima, "Adaptation of scalarizing functions in MOEA/D: An adaptive scalarizing function-based multiobjective evolutionary algorithm", *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 438–452, 2009.

[29] S. Jiang, S. Yang, Y. Wang and X. Liu, "Scalarizing Functions in Decomposition-Based Multiobjective Evolutionary Algorithms", *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 296–313, 2018.

[30] Q. Zhang, W. Liu and H. Li, "The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances", in *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, 2009, pp. 203–208.

[31] A. Nebro and J. Durillo, "A Study of the Parallelization of the Multi-Objective Metaheuristic MOEA/D", *International Conference on Learning and Intelligent Optimization*, vol. 6073, no. January, pp. 303–317, 2010.

[32] S. Z. Zhao, P. N. Suganthan and Q. Zhang, "Decomposition-based multiobjective evolutionary algorithm with an ensemble of neighborhood sizes", *IEEE Transactions on Evolutionary Computation*, vol. 16, no. 3, pp. 442–446, 2012.

[33] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces", *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[34] Á. Fialho, L. Da Costa, M. Schoenauer and M. Sebag, "Dynamic multi-armed bandits and extreme value-based rewards for adaptive operator selection in evolutionary algorithms", *International Conference on Learning and Intelligent Optimization*, pp. 176–190, 2009.

[35] F. G. Lobo, C. F. Lima and Z. Michalewicz, *Parameter Setting in Evolutionary Algorithms*. 2007, vol. 54.

[36] Á. Fialho, "Adaptive Operator Selection for Optimization", *Université Paris Sud*, 2010.

[37] P. Auer, "Finite-time Analysis of the Multiarmed Bandit Problem", *Machine learning*, vol. 47, no. 2-3, pp. 235–256, 2002.

[38] F. G. Lobo and D. E. Goldberg, "Decision making in a hybrid genetic algorithm", in *Proceedings of the IEEE Conference on Evolutionary Computation, ICEC*, 1997, pp. 121–125.

[39] D. Thierens, "An adaptive pursuit strategy for allocating operator probabilities", *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pp. 1539–1546, 2005.

[40] Á. Fialho, L. Da Costa, M. Schoenauer and M. Sebag, "Extreme value based adaptive operator selection", in *International Conference on Parallel Problem Solving from Nature*, 2008, pp. 175–184.

[41] K. Li, A. Fialho, S. Kwong and Q. Zhang, "Adaptive operator selection with bandits for a multiobjective evolutionary algorithm based on decomposition", *IEEE Transactions on Evolutionary Computation*, vol. 18, no. 1, pp. 114–130, Feb. 2014.

[42] Á. Fialho, M. Schoenauer and M. Sebag, "Analysis of adaptive operator selection techniques on the Royal Road and Long K-Path problems", *Proceedings of the 11th Annual Genetic and Evolutionary Computation Conference, GECCO-2009*, pp. 779–786, 2009.

[43] W. Gong, A. Fialho and Z. Cai, "Adaptive strategy selection in differential evolution", in *Proceedings of the 12th Annual Genetic and Evolutionary Computation Conference, GECCO '10*, 2010, pp. 409–416.

[44] Á. Fialho, M. Schoenauer, M. Sebag, Á. Fialho, M. Schoenauer, M. Sebag, T. C.-b. Adaptive, Á. Fialho, M. Schoenauer and M. Sebag, "Toward Comparison-based Adaptive Operator Selection", *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pp. 767–774, 2010.

[45] J. Maturana and F. Saubion, "A compass to guide genetic algorithms", *International Conference on Parallel Problem Solving from Nature*, vol. 5199 LNCS, pp. 256–265, 2008.

[46] A. K. Qin, V. L. Huang and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization", *IEEE Transactions on Evolutionary Computation*, vol. 13, no. 2, pp. 398–417, 2009.

[47] L. D. Costa, Á. Fialho, M. Schoenauer, M. Sebag, L. D. Costa, Á. Fialho, M. Schoenauer, M. Sebag, A. Operator, L. Dacosta, Á. Fialho, M. Schoenauer, M. Sebag and O. Cedex, "Adaptive Operator Selection with Dynamic Multi-Armed Bandits", *Proceedings of the 10th annual conference on Genetic and evolutionary computation*, pp. 913–920, 2008.

[48] D. Hinkley, "Inference About The Change-point From Cumulative Sum Tests", *Biometrika*, vol. 58, no. 3, pp. 509–523, 1971.

[49] Goncalves, Almeida and Pozo, "Upper Confidence Bound (UCB) Algorithms for Adaptive Operator Selection in MOEA/D", *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 411–425, 2015.

[50] H. J. C. Barbosa and A. Medeiros-e-Sá, "On Adaptive Operator Probabilities in Real Coded Genetic Algorithms", *XX Intl. Conf. of the Chilean Computer Science Society*, 2000.

[51] V. Huang, A. Qin, P. Suganthan and M. Tasgetiren, "Multi-objective Optimization based on Self-adaptive Differential Evolution Algorithm", *2007 IEEE Congress on Evolutionary Computation*, pp. 3601–3608, 2007.

[52] V. L. Huang, S. Z. Zhao, R. Mallipeddi and P. N. Suganthan, "Multi-objective optimization using self-adaptive differential evolution algorithm", *2009 IEEE Congress on Evolutionary Computation, CEC 2009*, pp. 190–194, 2009.

[53] S. M. Venske, R. A. Gonçalves and M. R. Delgado, "ADEMO / D : Multiobjective optimization by an adaptive differential evolution algorithm", *Neurocomputing*, vol. 127, pp. 65–77, 2014.

[54] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan and W. Liu, "Multiobjective optimization Test Instances for the CEC 2009 Special Session and Competition", *University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report*, vol. 264, pp. 1–30, 2008.

[55] D. E. Goldberg, "Probability Matching, the Magnitude of Reinforcement, and Classifier System Bidding", *Machine Learning*, vol. 5, no. 4, pp. 407–425, 1990.

[56] R. Storn, "On the usage of differential evolution for function optimization", *Biennial Conference of the North American Fuzzy Information Processing Society - NAFIPS*, no. May, pp. 519–523, 1996.

[57] K. Li, K. Deb, Q. Zhang and S. Kwong, "An evolutionary many-objective optimization algorithm based on dominance and decomposition", *IEEE Transactions on Evolutionary Computation*, vol. 19, no. 5, pp. 694–716, 2015.