ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Semester Thesis at the Department of
Information Technology and Electrical Engineering

Spring Semester 2009

Optimization According to Interactively Quantified User Preferences

Simon Hügi & Björn Muntwyler

Professor:	Prof. Dr. Eckart Zitzler
Advisor:	Tamara Ulrich
Handout:	Thursday, 12th of March, 2009
Due:	Tuesday, 16th of June, 2009

Acknowledgment

First of all, we want to thank our advisor, Tamara Ulrich, for her support and making this thesis possible. Furthermore, we would like to thank Dimo Brockhoff and Johannes Bader for sharing their rich experience in the area of multiobjective optimization.

A special thank you goes to Kaisa Miettinen for the idea of the interaction cycle, pointing us in the right direction.

Zurich, June 16, 2009

Simon Hügi & Björn Muntwyler

Abstract

An interactive optimization method is developed extending the existing PISA framework to an interaction cycle. The user interaction is embedded into the EMO, executed after a predefined amount of generations, asking for the preferences of the decision maker. This allows an emphasized optimization and even focus changes during the search. A presentation and interaction method has been developed to capture the user preferences. Different proposals to process the received data into a weight distribution for the pareto front, which can be interpreted by the PISA framework to continue the optimization with new parameters, are discussed and applied. Additionally, the available framework was improved with a history archive to enable a speed up during focus changes. At the end, a dynamic behavior analysis completes this work.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

"Most real-world engineering optimization problems are multiobjective in nature, since they normally have several (possibly conflicting) objectives that must be satisfied at the same time" [1]. As we have multiple criteria, which can't be all optimized at the same time[1], the aim is not to find one single solution, but to find a set of compromise solutions, i.e. each of these compromise solutions is in at least one criteria better than any other, hence solutions which are trade-offs. There already exist such procedures for *Evolutionary Multiobjective Optimization* (EMO), which were first hinted by Rosenberg in the 1960s. But the big interest in such kind of optimization came in recent years [1].

There also exists the task of *Multi-Criteria Decision Making* (MCDM), where the know-how of an expert is used to choose the one, preferable solution out of the set of compromise solutions. Through interaction with the user, the MCDM procedure captures the preference information to determine regions of interest in the optimization space, to help the expert to make his / her decision of choosing a single solution in favor. The interaction step is necessary, because all solutions of the set of compromise solutions optimize the multiobjective problem in it's own way. Hence, they cannot be compared by any automated mechanism and therefore need to be evaluated by a human being.

Until now, these two steps above have strictly been separated. The use of those is illustrated in figure (1.1). The first step on the left of the figure, while looking for the set of solutions to the multiobjective problem, is the optimization step. EMO results in a set of compromise solutions, which all optimize the problem, and hence are incomparable. That's why in a second step, after the EMO has reached the termination criteria, the user interaction takes place, to capture the user's preference model. With this knowledge, the task of MCDM is performed on the set of incomparable solutions to select the one, which fits the user's preferences the most.

[1] This is because some objectives may be conflicting, i.e. optimizing one of these will lead to a degradation of the other.

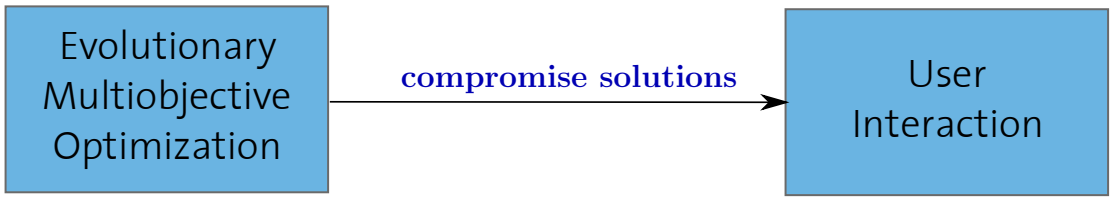

Figure 1.1: Before: The two step procedure. After the EMO process, the user interaction helped to find the one solution.

In this thesis, we try to combine these two steps from above. We include the user interaction step into the search procedure in order to allow a focused optimization. Hence, we can halt the EMO after a predefined number of generations to perform a user interaction. This allows concentrating the search on a specific region of the pareto front, which is in favor of the user. With this focus, the optimization procedure is continued only in this selected region, and the whole loop starts over. This leads to the *interaction cycle* illustrated in figure (1.2). With this method, we hope to be able to lower the overall computing effort and time. It also allows us to use the whole population of the evolutionary algorithm (EA) in one narrowed region, providing more potential choices in favor for the user.

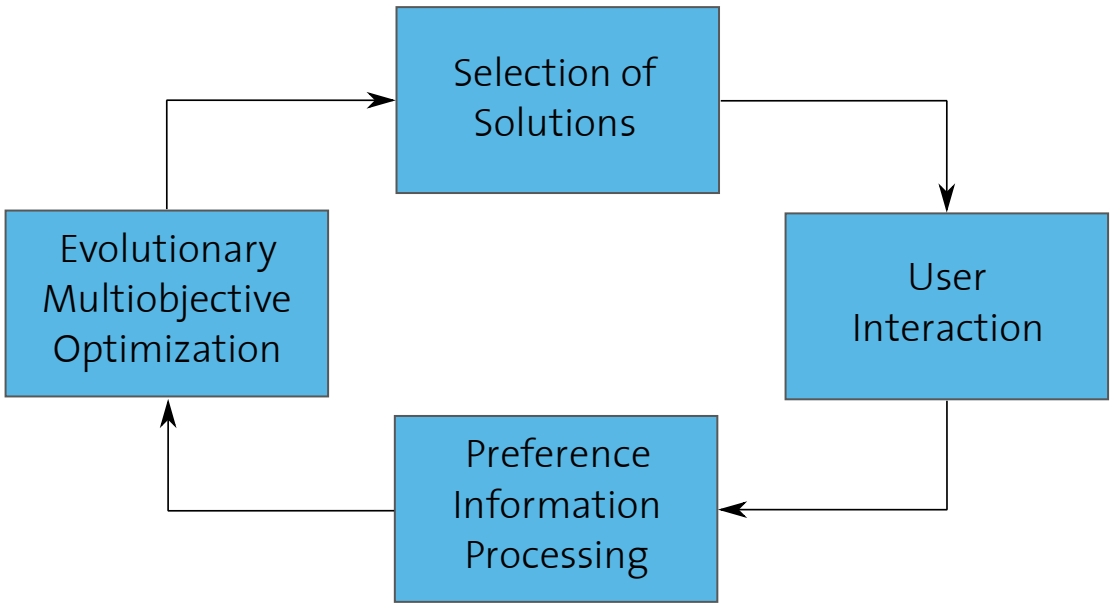

Figure 1.2: Interaction cycle

In the interaction cycle, the compromise solutions, which result from the EMO are used to interact with the user. The steps *selection of solutions* and *user interaction* will

be discussed in chapter 3. As well, the basic idea of how the user interaction takes place, how the current population of the EA is presented and how the preference model of the user is captured is explained. In chapter 4, the step *preference information processing* is explained. Here we develop and analyze different proposals of how to use the captured preference model to convert it into an input argument, which can be used by the EA to perform the focused optimization.

In chapter 5 we discuss how we adapt and use the existing PISA[2] framework to even enable this interaction cycle and speed up the EMO.

1.1 Task Description

In order to fulfill the above ambitions, the following tasks had to be completed:

- Extensive literature study
- Adapting the existing PISA framework to perform this interaction cycle and to include an archive solution to speed up focus transitions
- Identifying methods to select and present a representative choice of solutions
- Finding a suitable interaction method to capture the user's preference model
- Developing methods to process the user's preference model into a weight distribution, which is suitable as a parameter for the EMO procedure
- Analyze the dynamical behavior of the interaction cycle and the influence of the developed archive solution

[2]PISA is a EA framework developed by ETH Zurich, which will be explained later in section 2.1

Chapter 2

Background Knowledge

2.1 PISA - A Platform and Programming Language Independent Interface for Search Algorithms

The PISA framework was developed at the Computer Engineering and Networks Laboratory (TIK), ETH Zurich, by Stefan Bleuler, Marco Lausmanns, Lothar Thiele and Eckart Zitzler [2]. Its main idea is to divide evolutionary (multiobjective) algorithms into two separated parts:

- problem-specific: Representation of candidate solutions and variation operators.
- problem-independent: Selection operators.

2.1.1 Variator

The representation of candidate solutions is the way how individuals are defined in the context of an evolutionary optimization, i.e. a mapping from the original problem to the problem solving space [3], which is highly problem specific.

The variation of a population is a stochastic process, that aims to create new individuals (the offsprings) from old ones [3]. There are two operators: a unary operator (taking one individual of the population as input) is usually called *mutation*, whereas *recombination* is a binary operator (taking two individuals as input).

A program implementing the representation of solutions and variation operators is called *variator*.

2.1.2 Selector

The selection is rather a deterministic process, consisting of two steps: *Mating selection* tries to select better individuals as parents for the next generation [3]. *Environmental selection*, however, is applied on the resulting offspring generation, where solutions with higher quality have higher chances to survive.

A program implementation of the selection procedure of individuals is called *selector*.

2.1.3 The Framework

PISA defines an interface between the variator and the selector. This means, that both programs are able to interact with each other communicating with text files. The notation for their names used in this paper can be found in the appendix, A.1. The location where these files can be found is denoted as *communication directory*.

The different population sizes are defined in the configuration file `cfg`:

`alpha`	initial population size
`mu`	parent population size
`lambda`	offspring population size
`dim`	number of objectives

There are four different files for the exchange of the populations. The variator creates an initial population (with size `alpha`), which is saved in the file `ini` right at the beginning of the initialization. During the optimization phase, its offspring population (with size `lambda`) is written to the file `var`. Both files contain for each individual an identification number (index) and its objective values. The selector writes its selected parent population (with size `mu`) to the file `sel`, which contains only the indices of the individuals. The archive population written to `arc` comprises solutions the selector would like to keep.

To synchronize both processes, the state file `sta` is used. PISA defines the following states:

State	Set by	Action	Next State
0	V	Create initial population	1
1	V	Create sample	2
2	S	Variate sample	3
3	V	Create sample	2
4	V	Terminate variator	5
5	V	Variator terminated	-
6	S	Terminate selector	7
7	S	Selector terminated	-
8	V	Reset variator	9
9	V	Variator reset	
10	S	Reset selector	11
11	S	Selector reset	

2.1.4 Monitor

The monitor is an additional program that was also developed at the TIK by Lothar Thiele [4]. If the monitor is used, the variator and the selector do not communicate directly, but the monitor “sits in between” these programs and regulates the control

and data flow. Each program has its own communication directory in place of sharing the same. The monitor controls the states of both programs and moves the produced output files from one side to the other at the right time. This offers several possibilities to influence the optimization.[1]

2.2 The Hypervolume Indicator

The term "hypervolume" is basically a generalization of the 3-dimensional term "volume". It is used in the case of more than three dimensions. The hypervolume is always expressed in dependence of a reference point and a set of solutions.

The hypervolume indicator is a measure that can be used to redefine the pareto dominance: If the objective functions $f_i(\mathbf{z}),\ i \in \{1,\ldots,k\},\ \mathbf{z} \in \mathbb{R}^k$, are to be minimized, a selector preferably chooses those solutions that maximize the hypervolume indicator [5].

The hypervolume indicator can be weighted to give those solutions with a larger weight a higher change of been selected (see [6] for more details). As the major drawback of the hypervolume indicator is its high computation effort, the value can be approximated using the Monte Carlo sampling method [5]. This permits even more possibilities to articulate user preferences by sampling regions with different numbers of samples.

2.3 The WDFS Selector

This selector offers the possibility to articulate user preferences by sampling the weighted hypervolume [7]. The simple idea is to sample the objective space randomly according to the possible multivariate distributions in order to estimate the hypervolume indicator. The parameter file contains these distributions (as shown in appendix A.3), the seed and the number of total samples `nrOfSamples`.

Gaussian distribution

- Preference point $\boldsymbol{\mu} = (\mu_1,\ldots,\mu_k)^T \in \mathbb{R}^k$
- Direction $\mathbf{t} = (t_1,\ldots,t_k)^T \in \mathbb{R}^k,\ |\mathbf{t}| \neq 0$
- Standard deviation $\sigma_\epsilon \in \mathbb{R}$ describing the widespread of the solutions
- Standard deviation $\sigma_t \in \mathbb{R}$ in direction of $\pm\mathbf{t}$
- Covariance matrix $C = \sigma_\epsilon^2 \cdot \mathbb{I} + \sigma_t^2 \cdot \frac{\mathbf{t}\cdot\mathbf{t}^T}{|\mathbf{t}|}$

These parameters describe the multivariate Gaussian distribution

$$w(\mathbf{z}) = \frac{1}{\sqrt{(2\pi)^k \det(C)}} \cdot e^{-\frac{1}{2}(\mathbf{z}-\boldsymbol{\mu})^T C^{-1}(\mathbf{z}-\boldsymbol{\mu})}.$$

[1] This issue is explained latter in chapter 5.

Uniform distribution

- Lower bound $\mathbf{b_l} = (b_1^l, \ldots, b_k^l)^T \in \mathbb{R}^k$
- Upper bound $\mathbf{b_u} = (b_1^u, \ldots, b_k^u)^T \in \mathbb{R}^k$

This results in the multivariate uniform distribution

$$w(\mathbf{z}) = \begin{cases} \prod_i \frac{1}{b_i^u - b_i^l} & , \mathbf{z} \in \left[b_1^l, b_1^u\right] \times \ldots \times \left[b_k^l, b_k^u\right] \\ 0 & , \text{ else} \end{cases} .$$

Exponential distribution

Not used in our framework.

All distributions can be combined together. The user can give each one a different weight factor $\alpha \in [0, 1]$. This number determines how many samples are used for the specific distribution.

2.4 The WERA Selector

This selector weights the hypervolume according to weighted rays [8]. In our thesis, we limit ourselves to 2 objectives. As one can see in figure (4.1), the objective space is subdivided in rays. Each of these rays has a weight assigned, which will be used to weight the hypervolume in this part of the pareto front.

The WERA parameter file contains the seed and all data needed to construct these weighted rays. The line of interest in this file is the following.

$$\text{rays } 0\ 0\ \alpha_0\ w_1\ \alpha_1\ w_2\ \alpha_2\ w_3\ \alpha_3\ \cdots\ w_n\ \alpha_n$$

where to each ray, spanning from α_{i-1} to α_i, the weight w_i is assigned, $i \in \{1, ..., n\}$, while $n \geq 1, n \in \mathbb{N}$ denotes the number of rays. The angles α_i are measured in a positive mathematical direction. The first two zeros refer to the center of the rays.

Chapter 3

Interaction Methods

After performing the evolutionary multiobjective optimization step, one is always confronted with a set of compromise solutions[1].

This chapter refers to the modality of how to interact with the user, how to capture his / her preference model and how to present the current state to the user in order to even get some information containing his / her expertise. To be able to guide and focus the search on a specific region of the pareto front, this step is essential to even know where this special region of interest might be. Especially, we want to capture the know-how of the user without making any assumptions about the problem itself. The procedure should be totally problem-independent. There are various ways to perform an interactive articulation of preference information from the user [9]. In general, there are specific questions necessary to find the focus of interest.

While working through the technical literature we noticed that most of them do not make a problem-independent entry, or even make assumptions about the value function[2] of the user [9]. In [10] for example, they used a set of yes or no questions on a set of pairwise comparisons of solution vectors to establish the experts utility function[3]. Or in [12], where they use local trade-off ratios to reflect the user's favoritism between two criteria. But like many other papers too, throughout [10] and [12] it is assumed that the objective functions are either concave, or differentiable and concave.

But in the end, we needed something which is easy for the user to understand and predict, which does not make any assumptions about the value function or objective functions. The complexity of the user interaction method should remain low and especially should not exceed the capabilities of the user.

[1] Compromise solutions are solutions, which all minimize the multiobjective problem and for that reason are incomparable without human interaction.

[2] A value function is a function, describing the user's preference model.

[3] According to wiktionary [11], a utility function is a mathematical function, which assigns a real number to every element of the outcome space in order to collect the user's preference model.

3.1 Our Interaction Method

3.1.1 Basic Idea

After performing the evolutionary multiobjective optimization procedure for the specified amount of generations, the resulting population is passed on in the interaction cycle[4] to the next instance, the solution selection and user interaction.

The current solution received should be evaluated by the user, in order to be able to focus on a specific region of the pareto front[5]. But as the population size is way to big to be evaluated one-by-one, the amount of solutions to evaluate can be specified by the user. The interaction procedure should then select a representative choice of solutions from the current population, i.e. more solutions should be chosen in regions of high population density, than in regions of low density. The selection is then presented to the user, who can rate each of the selected solutions with a weight between 0 and 10. Hence, the user can submit his / her expertise in an easy way to the interaction procedure in order to focus the optimization on a region of interest, rather then optimizing over the entire front.

The selected solutions are presented in the objective space, like illustrated in figure (3.1). This only suits for problems with 3 criteria or less. For higher order multiobjective problems one might need new approaches to present the solutions. One possibility could be to evaluate each solution in multiple steps, giving partial weights with respect to one or two objectives at the time. Notice, that in this interaction method, it is simple to evaluate the solutions in the case of only two or possibly three objective functions. But as soon as the amount of criteria grows, this task will get rapidly more difficult.

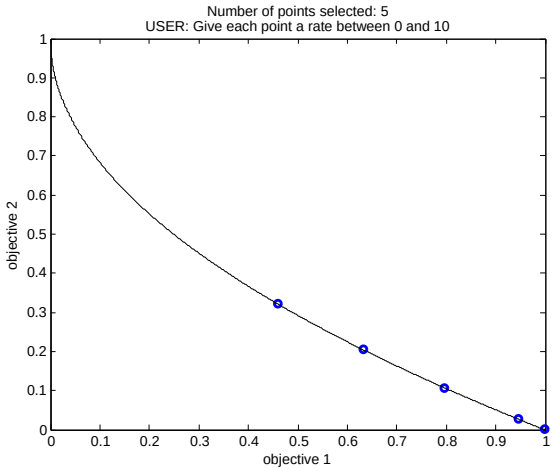

Figure 3.1: How the presentation of solutions will be displayed to the user during the user interaction step.

[4]Explanation of the interaction cycle, see chapter 1.

[5]As a part of the assembly of evolutionary multiobjective optimization and decision making.

As a remark, the presented interaction method is a trade-off between reasonability and precision. The more solutions are evaluated by the user, the more precise is the captured preference model. But with many evaluations, the effort and time needed to handle the interaction step grows too.

Here's an example of such a user interaction procedure using the WERA selector's interaction protocol. In the meantime, the plot (3.1) is shown for visualization of the selected solutions.

```
User Interaction #1
**********************
Interaction Method (wdfs | wera | unif | help | exit): wera
    Choose approach ( rabi | sifa | weli | help | skip ): rabi
    Chosen approach does not require a scalarizing function!
    Number of points to rate: 5
    Number of rays determined by number of evaluated points.
      #rays = 5
    Rate each point [0, 10]:

      weight for point: 0.99821 x 0.00089614: 0
      weight for point: 0.94607 x 0.027337: 1
      weight for point: 0.79627 x 0.10766: 8
      weight for point: 0.6324 x 0.20476: 7
      weight for point: 0.45908 x 0.32246: 0

    START execution of approach 1:
rays 0 0 0 0 0.83195 12.5 4.4304 100 12.3353 87.5 25.782 0 90
  Continue (s)earch or redo (w)era interaction: s
```

Additionally, one could enter `skip`, which would jump over the user interaction procedure and continue the optimization with the same parameters used in the previous WERA optimization round.

3.1.2 How to Select the Points

In order to be able to present m[6] solutions to the user, the procedure has to make a representative collection of population members from the current solution pool. For the following discussion, it is assumed that the population is sorted on the 2-dimensional pareto front from right to left.

Assuming the population size is `alpha`, the members of this population are stored in the array called `populationMember` and we need to select m of those to present to the user. The algorithm first computes a step size parameter called `stepSize`. Then the routine traverses through the whole population, selecting those solutions having an

[6] m denotes the amount of solutions to be evaluated. It is specified at the beginning of each interaction by the user

index which is divisible by `stepSize` without remainder. Additionally the first and last one within the population are chosen as well.

Generally speaking, the procedure steps through the whole population, and chooses m population members. It makes sure that the selected solutions are equidistant from each other, in terms of number of solutions in between two choices.

Here is the pseudo code of the selection algorithm:

1: `stepSize` $= \text{round}\left(\frac{\textbf{alpha}}{(m-1)}\right)$;
2: `selectedPoints`(1) = `populationMember`(first);
3: **for** i from 2 to $(m-1)$ **do**
4: `selectedPoints`(i) = `populationMember`($(i-1) \cdot$ `stepSize`);
5: **end for**
6: `selectedPoints`(m) = `populationMemeber`(last);

This algorithm will make sure that the presented solution points are a representative description of the whole population. This means that in regions of high population density, the algorithm's choices will be close to each other than in regions of low population densities. Figure (3.2) illustrates some examples of populations (black dots) and the corresponding points selected by the algorithm above (red dots).

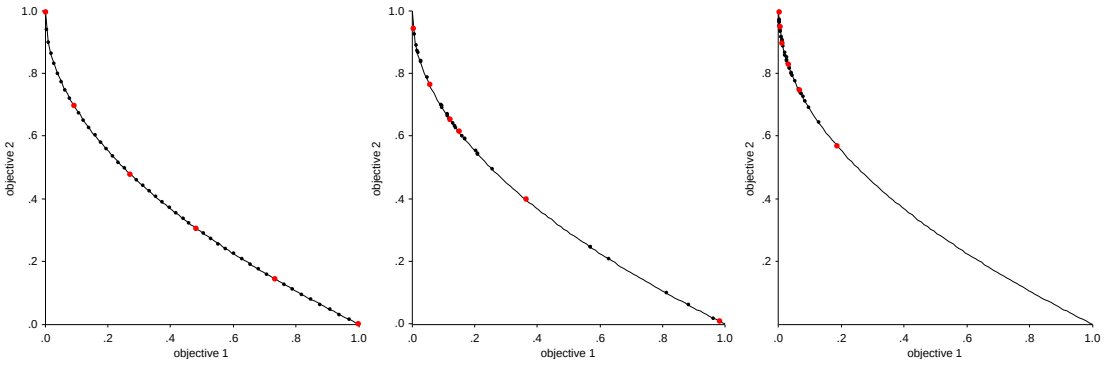

Figure 3.2: Plot of the current population (black dots) for 3 different test cases, one uniform and two with a focused region. In each case, there were 6 solutions selected (red dots).

Chapter 4

Processing of User Preferences

In this chapter we will introduce different proposals how the collected preference data is processed in order to fit the used selectors requirements. This is necessary to establish a focus on the region of the user's interest. The established approaches, which will be introduced in the next few chapters below, are all specified to be applied to the WERA selector. This selector was established and explained in [8]. The most important property, which should be known here is the input argument "rays", needed by WERA to build and weight the new rays. It is defined as a string containing the edges and weights of each ray. A general form of the parameter "rays" can be written as

$$\text{rays } 0\ 0\ \alpha_0\ w_1\ \alpha_1\ w_2\ \alpha_2\ w_3\ \alpha_3\ \cdots\ w_n\alpha_n$$

while the weights w_i are assigned to the ray r_i, spanning from the angles α_{i-1} to α_i, $i \in \{1, ..., n\}$, $n \geq 1, n \in \mathbb{N}$ being the number of rays. α_0 is mostly zero degrees, and α_n 90 degrees[1]. The angles are measured in a mathematical positive direction, i.e. going from 0 to 90 degrees is the same as traversing the front from right to left. The first two numbers, here 0 0, specify the intersection of the cones, but are not of further interest here. In figure (4.1) is an example of this selector and its weighted rays depicted, where the number of weighted rays n is 10 and the parameter used for this plot is:

rays 0 0 0 6.4615 9 21.5385 18 37.5 27 54.1739 36 71.3043 45 88.6957 54 92.6261 . . .
. . . 63 73 72 52.4857 81 29.2143 90

[1]For this thesis, this is always true.

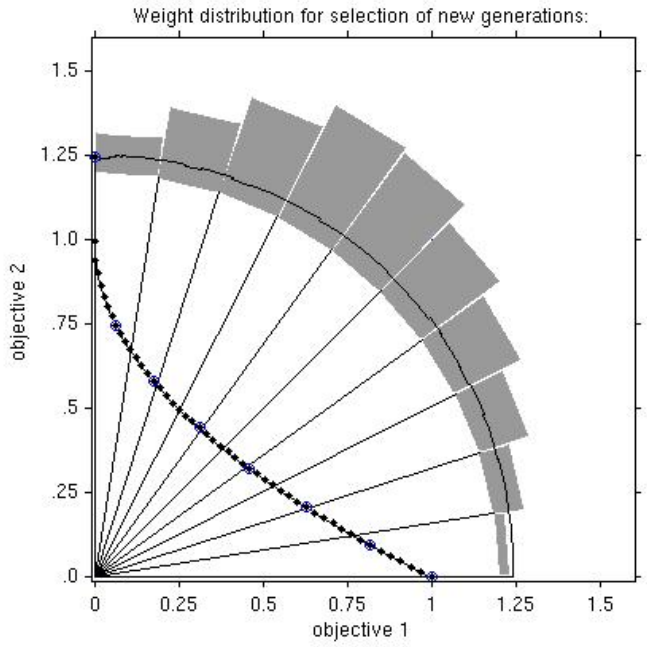

Figure 4.1: Plot of a uniform front with $n = 10$ weighted rays. This kind of weighted rays are used in the WERA selector to affect the density of the sampled solutions on the front.

But how can we assign the captured preference model to these weighted rays? How can we determine the angles and the corresponding weights? – In the following couple of subchapters, we discuss two different proposals with their properties and limitations. In the end, we briefly look at an approach using scalarizing functions, which actually is more of an idea for future work.

4.1 The Rays by Angle Bisector Approach (RABI)

4.1.1 Basic Idea

This is the simplest approach, which we implemented. At first, the user can specify the amount of points m from the current population[2]. For the evaluation of the selected points, the user assigns weights $w_i, i \in \{1, ..., m\}$ between 0 and 10 to each of these points p_i according to his preference model, as discussed in chapter (3.1.1).

The basic idea of the RABI approach is to assign one ray to each of the points p_i, which is weighted proportional to the ratings given by the user. That for, the algorithm will compute the angle bisector between each of the selected points, which determines the boarders between each of the rays. Then, the weights w_i assigned to p_i are normalized

[2]This population was determined by the evolutionary multiobjective optimization procedure before this user interaction, as mentioned in chapter 3.1.

to w_{ni}, such that they sum to one. The resulting normalized weight is then assigned to the ray containing that particular point p_i to which w_i was assigned to by the user in the first place.

As a last step, the weights are stretched, such that they span from 0 to 100. This is necessary to fit the requirements of the WERA selector.

In figure (4.2) the whole process of the RABI approach is depicted. For the sake of simplicity, the population from which we start is uniform distributed, which leads to a more or less uniform distribution of the selected points[3] as well. In this case, the user defined 4 solutions to be evaluated. The weights $(1, 3, 9, 4)$ are assigned to them, parsing the front from right to left. On the right hand side is the resulting population after optimizing with respect to the just assigned weighted rays. As one can see, the WERA selector used the given weights quite well to influence the population density.

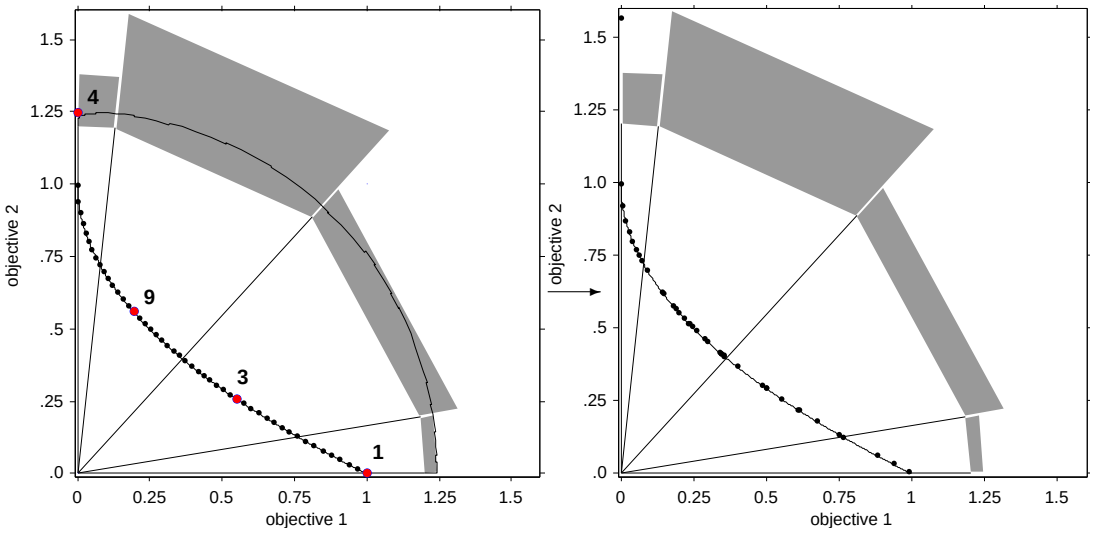

Figure 4.2: On the left side is the user rating depicted to each evaluated solution (blue) on the front, and the resulting weighted rays. On the right side, there is additionally the corresponding population after one interaction round of only 20 generations.

4.1.2 Properties and Limitations

A main property of the RABI approach is that to each evaluated solution, there is one ray assigned to, while the angle bisector between two points is limiting the width of each ray. This fact is very important to understand the dynamics of this approach over multiple interaction rounds. The simplicity of this approach also allows the user to easily predict the weighted rays, which will result from the user's input.

[3] See section 3.1 for how these solutions are selected.

When the current population is uniform distributed over the front, the selected points are uniform distributed too. Hence, all rays, except for the first and last one, are all of the same size[4], as can be seen in figure (4.2). This leads to a first beam focusing according to the user's preferences.

This approach delivers the possibility to switch between focus regions in at least two steps. Now lets assume the population is concentrated in one region, like it is shown in figure (4.3), from the users previous preference model. But this time, the user changes his mind in the following interaction step and wants to switch the region of interest to an other part of the pareto front. So one needs to specify, like depicted on the left of figure (4.3), in which direction the user likes to go. Hence he / she assigns the most right selected point the highest rating. The resulting weight distribution assigned to the rays will look like the one given in figure (4.3).

After a next interaction round, the most of the population's solutions will be concentrated within the most right ray of figure 4.3). In a second step, the focus can then be narrowed to the region of interest by weighting the new solutions accordingly. See figure (4.4) for illustration. On the left is the resulting population and a new possible weight distribution which would narrow down the region of interest. After the one more EMO step, the population will look like illustrated on the right of figure (4.4).

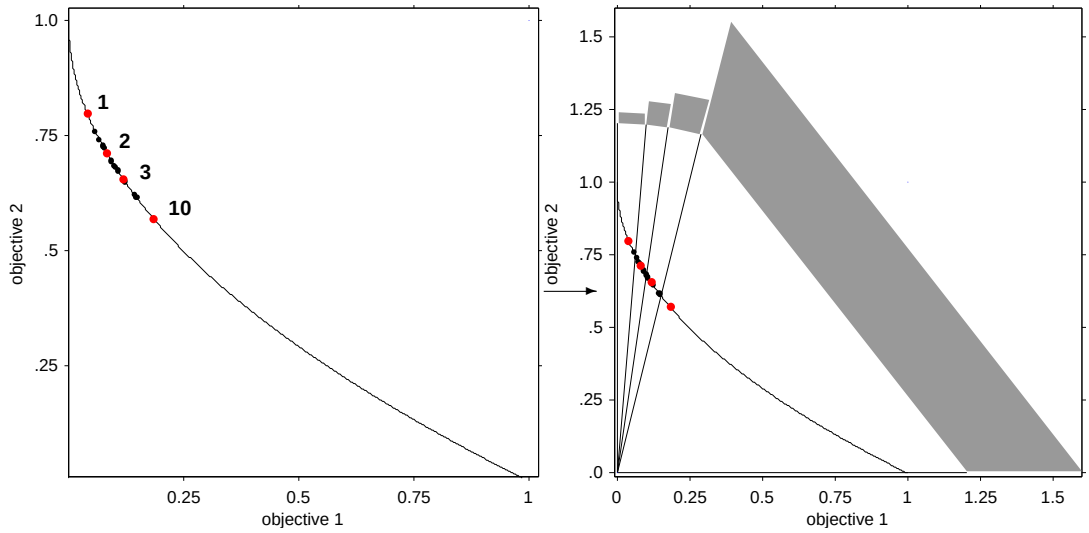

Figure 4.3: On the left side is the user rating depicted to each evaluated solution (blue) on the front and the resulting weighted rays on the right.

[4]This ray size also depends on the shape of the front.

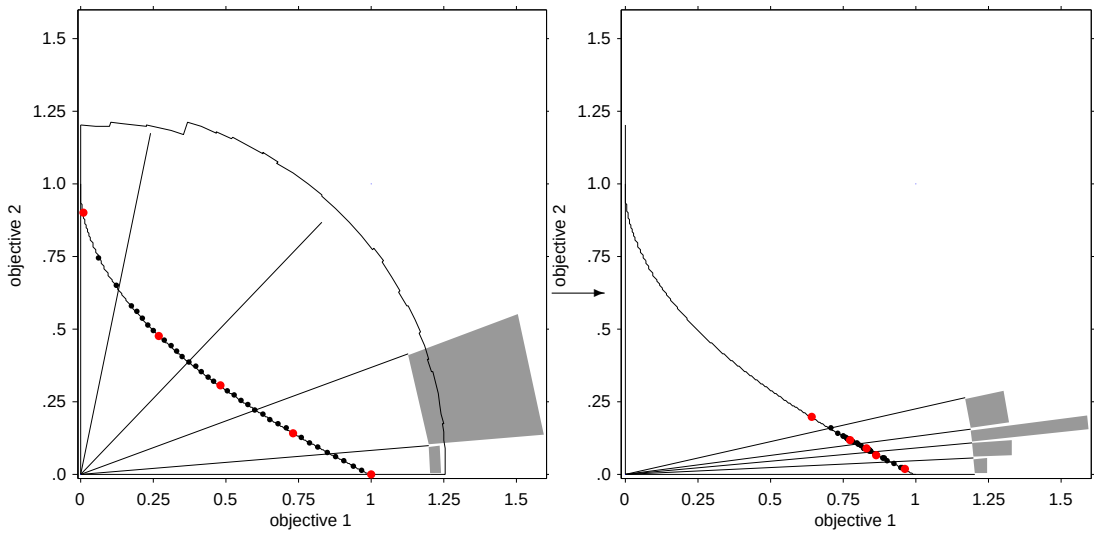

Figure 4.4: Getting from one region of interest to another, starting in figure (4.3).

A major limitation of the RABI method is that the resolution of the weight distribution over the rays is completely determined from the amount of evaluated solutions. Additionally, the big jumps between two successive rays is a bit of a problem. If we consider the test problem of figure (4.2), where the two middle solutions got the weights 3 and 9, assigned by the user. In this RABI approach, the solutions in between get either a 9 or a 3, instead of a weight between 9 and 3, like one would expect.
This will be considered in the following approach.

4.2 The Weights Linearly Interpolated Approach (WELI)

4.2.1 Basic Idea

This procedure is a renovation of the previous one. Similar to the RABI approach, the user conveys the amount m of points $p_i, i \in \{1, ..., m\}$ he / she would like to evaluate. In the next step, the user rates these points with weights w_i between 0 and 10. In contrast to RABI, the resulting weights will here be squared[5]. This will assure that the difference between solutions in favor of the expert and such rather uninteresting for further analysis gets more significant for the computations within the WELI approach. The resulting weights get normalized to w_{ni}, such that they sum to one just like before.

Additionally, the user needs to specify n, the amount of rays. This will determine in how many parts the weight-distribution should be subdivided. The bigger n, the smoother it will get. But this will be mentioned in more detail later on.

[5]This can be applied to all the other approaches too, if desired.

The WELI starts with the computation of the angles a_i of each point p_i. These angles will be used to describe the expert's preference model in the position versus weight domain. This step is necessary to construct a continuous weight-distribution function. In other words, one can imagine that the approach is drawing a plot with the x-axis labeled with the angles between 0 and 90 degrees, characterizing the angular position of p_i in the objective space, and the y-axis labeled with the user ratings w_{ni}. Hence, for each point p_i we get a dot at the position (a_i, w_{ni}).

This step is illustrated on the left side of figure (4.5). We assumed the same test problem as in figure (4.2) and the same ratings $(1, 3, 9, 4)$. Consider the fact, that in the figure below, the weights are already normalized to w_{ni}.

In order to finally obtain the continuous weight-distribution function, the key feature of WELI is to connect these dots (a_i, w_{ni}) with each other in terms of a linear interpolation[6]. This step can be seen proceeding from left to right in figure (4.5).

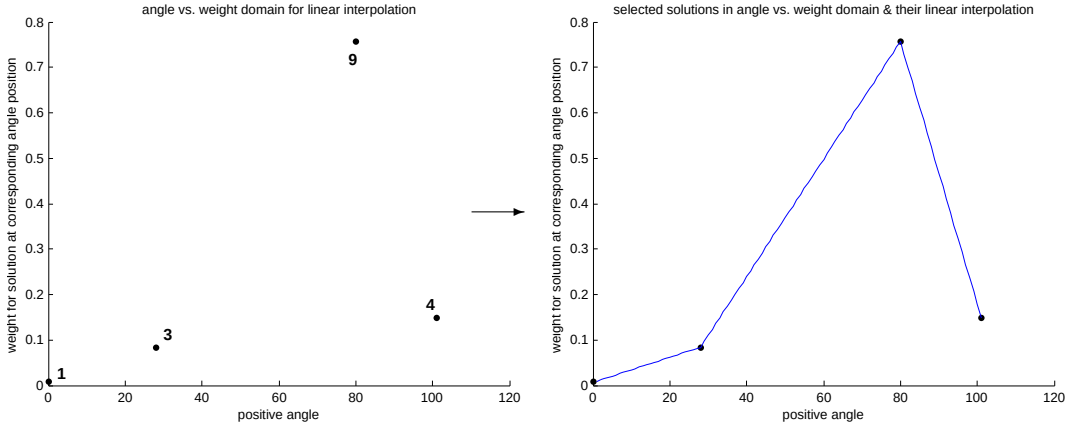

Figure 4.5: On the left side are the evaluated points with the corresponding user ratings. Note that the y-axis is the normalized weight and the x-axis denotes the angle position of the solutions. On the right side is the continuous weight distribution function resulting from the linear interpolation of the dots.

As a last step, the determined function is subdivided in n parts, n being the number of rays. Within each of these parts, the weight distribution function is averaged to a constant value, which then will be assigned to the corresponding ray. This step can be verified in figure (4.6).

[6]For further analysis of the WELI approach, one could try to use different kinds of interpolation methods, e.g. a polynomial interpolation.

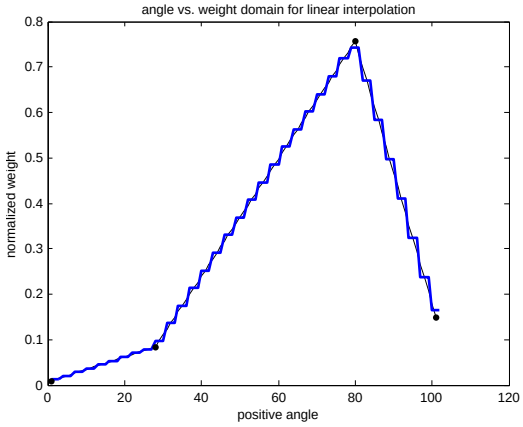

Figure 4.6: The depiction of the step, where the wdf is averaged over each ray.

To match the requirements of the WERA selector, the determined discrete weight distribution function is stretched, such that it spans from 0 to 100.

The resulting weighted rays of the WELI approach is depicted in figure (4.7). On the left side are the selected points, to which the user assigned the weights $(1, 3, 9, 4)$ and the resulting weighted rays. On the right side of figure (4.7) is the resulting population after one further interaction round. Same as in figure (4.2), the population density on the right hand side of figure (4.7) adapted the weighted rays well.

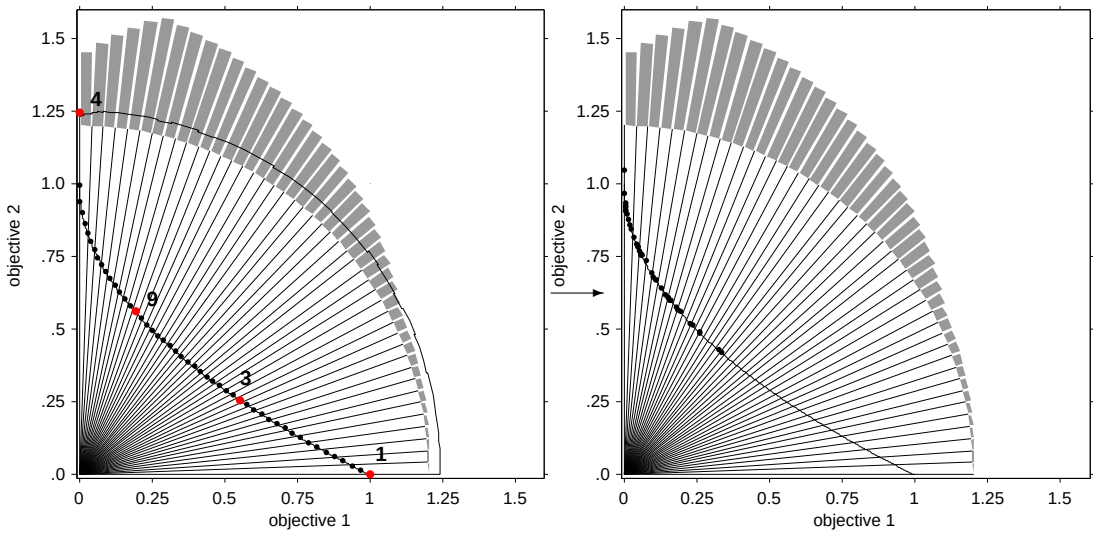

Figure 4.7: On the left side is the user rating depicted to each evaluated solution (blue) on the front and the resulting weighted rays. On the right side, there is additionally the corresponding population after one interaction round of only 20 generations.

4.2.2 Properties and Limitations

The WELI approach assures that the areas between two successive solutions with significantly different weights has a continuous weight transition. Lets consider an example. If the user assigns a weight of magnitude 1 to a selected solution p_1 and to a subsequent solution p_2, he / she assigns a weight of magnitude 5, all possible solutions among these two will get a weight somewhere between 1 and 5. So this seems quite reasonable.

Now what happens, if all the evaluated solutions are in the center of the pareto front? How are the edges of the weight distribution handled during the linear interpolation? - This can be done in various ways, one could for example decay the weights towards zero. But this would complicate our goal to be able to move our region of interest from one area of the pareto front to another, even though we don't have any solutions from the new, desired region available. Hence, the weights at the edges of the weight distributions of the available data is held constant, as illustrated on the left hand side of figure (4.8).

Even though this approach seems to enable an unlimited resolution, in reality it doesn't. First of all, we cannot have more than 100 rays. This follows from the implementation of WELI. If it is desired to have more rays, this is no big deal, but it has to be edited within the source code, as the used weight array currently has at most 101 slots.

Further, if two points within one ray have two significantly different weights, the algorithm will not be able to reflect exactly the same weight distribution as the user's preference model. This is because the algorithm will average the weights within each ray. This can happen if we have a lot of points within a small region of focus, and n, the number of rays, is too small. On the right hand side of figure (4.8) is an illustration of this limitation[7]. Here $m = 5$, $n = 6$ and the weights assigned by the user to the selected points (blue) are $(1, 5, 10, 3, 1)$, parsing the pareto front from right to left. One can circumvent this by just increase the number of rays, such that no evaluated solutions with significantly different weights are in the same ray.

An additional question came up concerning the reliability of the WELI approach. Due to the fact that the weights are normalized to w_{ni}, such that $\sum_{\forall i} w_{ni} = 1$, and the possible scenario when many low weighted solutions may be very close to each other at one end of the pareto front, and one very high weighted solution may be located at the opposite end of the pareto front, if this could lead to an unexpected weight distribution function. But this is actually no problem, as long as the weights of those solutions, which are very close to each other, don't vary to much. The density of evaluated solutions doesn't change the weights within the rays parameter, neither before nor after the normalization.

[7] Note that the first two points on the right hand side of the pareto front are very close to each other.

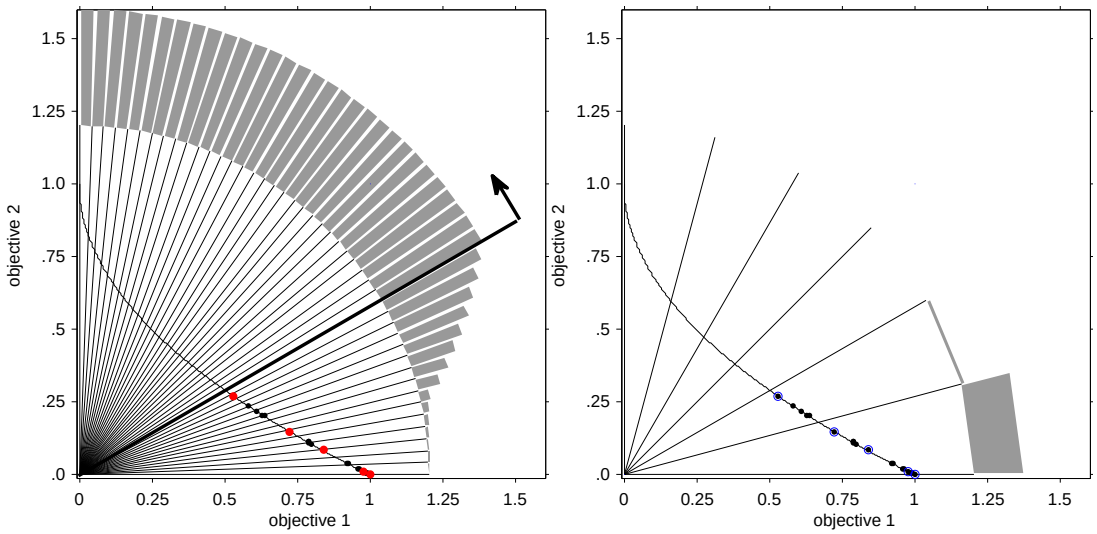

Figure 4.8: Left: Illustration of the linear interpolation if all the data is in a small angle interval. Here, the user evaluated 5 solutions according to (0, 0, 1, 2, 8). Right: Visualization of the limitation due to m and n on the resolution of the weight distribution function due to averaging.

4.3 The Scalarizing Function Approach (SIFA)

We also made the effort to develop a third approach, called the SIFA. Here, we tried to use scalarizing functions to reconstruct the preference model of the user in form of a continuous weight distribution function.

According to [9], a scalarizing function in general can be used to transfer a multiobjective problem into a single objective problem, for which there exist many optimization methods. 'In most scalarizing functions, preference information of the decision maker is taken into consideration." [13]

We thought of two different scalarizing functions. The first one was the "Weighted Tchebycheff scalarizing function" which is defined as

$$s_\infty(\mathbf{z}, \mathbf{f}(x_j), \mathbf{w}) = \max_{1 \leq k \leq \dim} \{w_k \cdot (f_k(x_j) - z_k)\}$$

where $\mathbf{z}$ is the vector of the preference point[8]. In our case, as our multiobjective optimization problem is a minimization problem, i.e. $\min\{f_1(\mathbf{x}), f_2(\mathbf{x})\}$, and we want our front to move towards the origin of the search space, we choose $\mathbf{z} = (0, 0)$. The vector $\mathbf{f}(x_j)$ is the objective vector of the point x_j, and $\mathbf{w} = (w_1, w_2)$ is a weight vector, with $w_i \geq 0, \forall i$ and $\sum_{\forall i} w_i = 1$.

The second function we tried, is the weighted-sum scalarizing function, defined as

[8] A preference point refers to a location in the objective space, towards which we try to optimize.

$$s_l(\mathbf{z}, \mathbf{f}(x_j), \mathbf{w}) = \sum_{k=1}^{\dim} \{w_k \cdot (f_k(x_j) - z_k)\}$$

with $\mathbf{z}, \mathbf{f}$ and $\mathbf{w}$ as above.

4.3.1 Basic Idea

This approach is less intuitive than the ones described above. It starts the same way as the WELI approach. The expert specifies an amount n of points to select from the current population and a quantity of rays.

The SIFA applies a scalarizing functions to the objective vectors of the selected points. Then, we need a transition from these scalarizing functions to the weighting of the hypervolume.

The weight w_1 is chosen as an array of weights with values spanning from 0 to 1 and $w_2 = 1 - w_1$. Each value of w_1 together with $w_2 = 1 - w_1$ represents a straight line in the objective space, from which we initially wanted to read off equipotential weighted solutions.

A possible algorithm to user scalarizing functions in this context could be the following:

1: compute $s(\mathbf{z}, \mathbf{f}(x_j), \mathbf{w}), \forall j$
2: normalize $s(\mathbf{z}, \mathbf{f}(x_j), \mathbf{w})$ to $s_n(\mathbf{z}, \mathbf{f}(x_j), \mathbf{w}), \forall j$
3: $\text{diff}(x_j) = \|s_n(\mathbf{z}, \mathbf{f}(x_j), \mathbf{w})\|^2 - \|weights(x_j)\|^2, \forall j$
4: $(w_{c,1}, w_{c,2}) = \min_{w_1,w_2} \text{diff}(x_j)$

while s denotes the chosen scalarizing function, *weights* the weight vector captured from the user interaction and diff refers to the difference taken from the weights and the normalized scalarizing function s_n. In this algorithm, we look for the weight pair $w_{c,1}, w_{c,2}$[9], which minimizes the diff(x_j) function. Then one could use the scalarizing function with $w_{c,1}, w_{c,2}$ to weight the hypervolume directly[10].

We actually had no success with the above functions and there are no scalarizing functions known which would fit our requirements to communicate the preference model over scalarizing functions to a weighted hypervolume. The problem is that depending on the user ratings, diff(x_j) will get very big, even if we use $(w_{c,1}, w_{c,2})$, which actually is minimizing it. One would need a scalarizing function, having more degrees of freedom, which would increase the complexity of the whole procedure. An alternative method could be to present fewer solutions to the user, such that all possible preference models can be characterized by the scalarizing function. But in our case, e.g. using the "Weighted Tchebycheff scalarizing function", we could only present two solutions to evaluate.

[9] The c in the indices refers to the fact that this weight pair has been chosen.

[10] This approach was no further considered in this thesis, as there was no such selector available at that time

To get a better understanding of why the proposed scalarizing functions can't work, see figure (4.9). As one can see on the left side, where only two solutions have been presented, the plot shows all possible permutations[11]. But if more than two solutions are rated just like on the right hand side of figure (4.9), where 4 solutions have been chosen, not all possible constellations of preference models are possible to characterize using the available $s(\mathbf{z}, \mathbf{f}(x_j), \mathbf{w})$. That's why a scalarizing function having more degrees of freedom is necessary to use this kind of proposal.

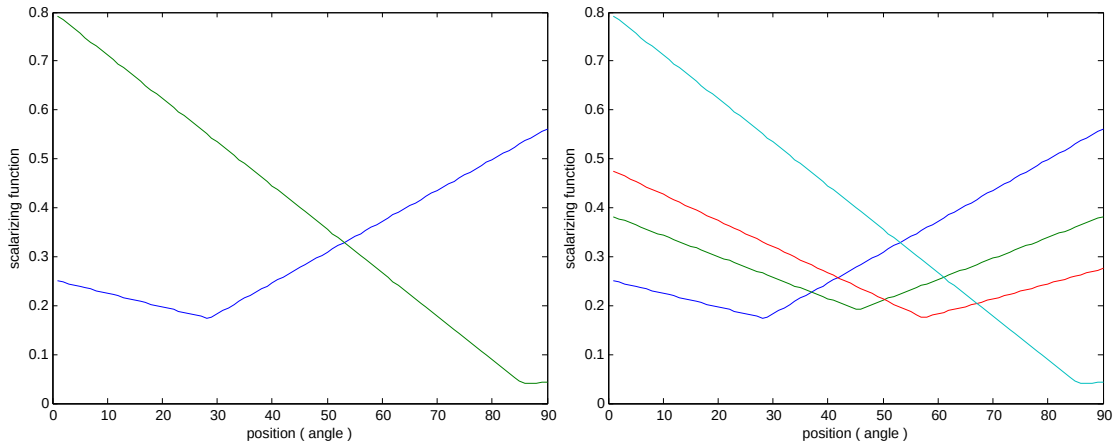

Figure 4.9: Illustration of problematic property of the used scalarizing function. Left: 2 evaluated solutions, Right: 4 evaluated solutions.

4.4 Comparison of the RABI and the WELI Approach

First of all let us mention that both approaches work just fine with almost any kind of input from the user. The difference between the two is especially with regards to number of rays and the resolution of the weight distribution.

Both approaches allow a transition from one region of interest to another in at most two steps, like illustrated in chapter 4.1.2. One should note that if the user want's to exclusively optimize in a local area of interest, the edges of the evaluated solutions around this area have to get a weight of magnitude zero. This is necessary in order to make sure that the selector will weight the hypervolume in those regions of the pareto front accordingly. If the user applies the RABI approach for this step, a more or less uniform distribution in this area of interest is achieved[12], while with the WELI approach, the user will receive a population density which is tapered towards the periphery of this region of interest. In figure (4.10) one can see such examples. On the left side, we used

[11] Here, a permutation refers to a possible preference model as one solution has a higher rating than another.

[12] depending on the shape of the front.

the RABI approach evaluating five solutions according to $(0, 0, 10, 0, 0)$, which leads to a heavyside shaped weight distribution. In the middle and on the right of figure (4.10), we used the WELI approach. First we also evaluated five solutions with the same preference model as we used in the RABI approach. Then we evaluated ten solutions in order to imitate the shape of the heavyside function. This illustrates what happens on the edges of the focused regions. The preference information for the far right plot was $(0, 0, 0, 0, 0, 10, 10, 0, 0, 0)$.

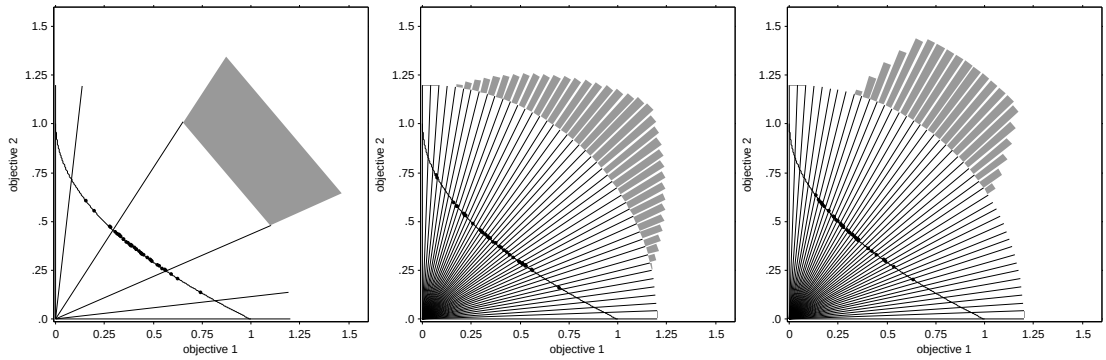

Figure 4.10: Focusing on a local area of the pareto front. Right: RABI approach, middle and left: WELI approach.

This is due to the fact of linear interpolation used in the WELI approach, like mentioned before in chapter 4.1.2 and 4.2.2.

Another difference is that the RABI approach, in contrast to the WELI, can adopt significantly different user ratings, no matter how close to each other the evaluated solutions are. In RABI, the width of the rays is just adjusted, as it uses the angle bisector to define them. But the WELI approach has equidistant rays, in which the solution ratings would get averaged, if the amount of rays is too small to account those separately.

4.5 The Influence of the Amount of Selected Points

The specified amount m of selected solutions, to which the user is supposed to assign weights to, determines what kind of shapes the preference model can have. The possible shapes for different m will be visualized using weighted rays just like those used by the WERA selector. It just makes it much easier to understand

If $\mathbf{m = 1}$, there are not that many possibilities for a weight assignment. As only one solution is evaluated, the direct consequence is that the whole front will get the same weight, leading to a uniform distribution.

If $\mathbf{m = 2}$, one can only highlight the edges of the front, or make a uniform weight assignment just like above.

If $\mathbf{m = 3}$, the possibilities increase significantly. All possibilities we had for $m < 3$

still remain. Additionally, we can highlight either both edges or only a middle part of the front. The precise shape, like the gradient of the slope, depends on how far or close the evaluated solutions are to each other and on the position of those solutions on the pareto front.

If $\mathbf{m} \geq \mathbf{4}$, again, the possibilities from above remain. The bigger m, the more shapes are possible. But the most important feature of choosing a high value for m is that the user can submit a even more precise image of his preference model. On the other hand the expenditure of time grows enormously.

In figure (4.11) is an illustration of the shape variations for different values of m.

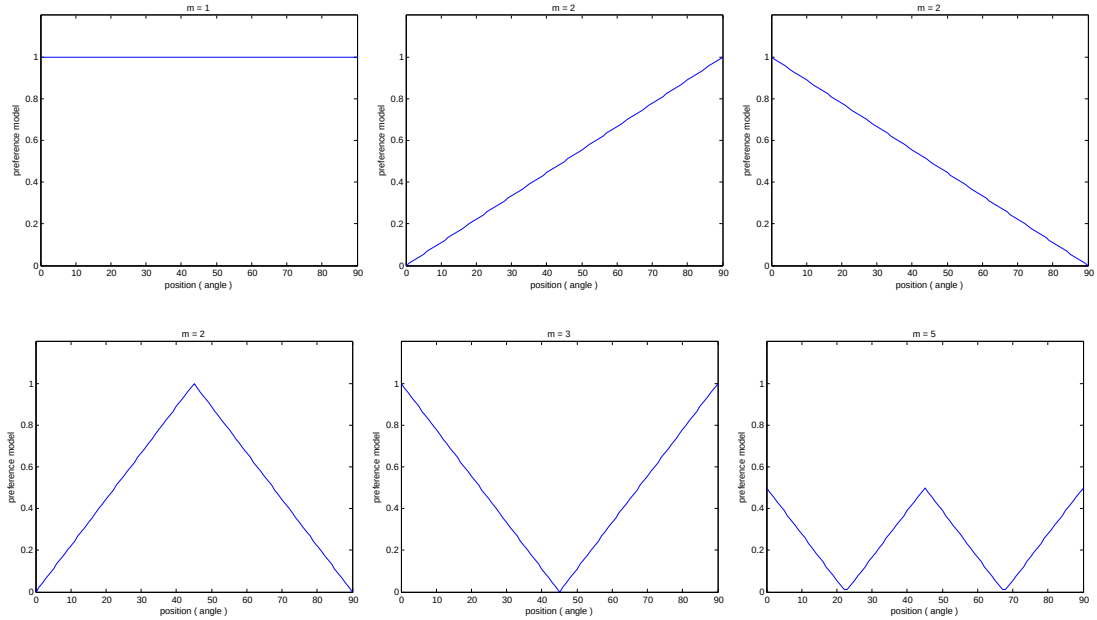

Figure 4.11: Possible shape variations of the preference model for different values of m. From top left to bottom right: $m = 1, m = 2, m = 2, m = 3, m = 3, m = 5$.

Chapter 5

Integration into the Monitor Module of PISA

The monitor module [4] of PISA [2] is the starting point for the integration of the user interaction into the search procedure. In a usual setup a variator and a selector would co-operate directly by sharing the same PISA files. The advantage of the monitor is that it acts as an interface between the two processes and it is therefore able to control the whole optimization procedure.[1]

The existing module has to be adapted in order to fulfill the desired functionalities. Namely, these are:

- The user should be able to choose between the two supplied selectors WERA [8] and WDFS [7].
- The user interaction should gather the preferences of the interacting user.
- The currently achieved front should be presented to the user.
- A history archive should keep non-dominated solutions from previous rounds. If the user decides to change the region in the objective space, the new area should be reachable in a small number of generations.
- The monitor will loose some of its original functionalities which are not needed for the new implementation:
  - There is just one round, i.e. the variator will be reset only at the beginning.
  - The user states[2] will be replaced.
  - The archive handling[3] will be managed in a new way.
  - The output handling[4] will be different.

[1] As mentioned in section 2.1.

[2] `monitor_user.c`: `state{1,2,3}_user()` removed.

[3] `monitor_user.c`: `paretoset_join()`, `dominates()` removed.

[4] `monitor_user.c`: `appendOutput()`, `outputAll()`, `outputOnline()`, `outputOffline()` removed.

5.1 The New Parameter File

There are several new features which require their own parameters. Thus, the parameter file of the monitor looks quite different. Instead of calling the monitor process with the specified arguments for one variator and one selector, these parameters are stored within the parameter file too. So the execution command for the monitor changes to:

```
./monitor monParamFile monOutputDir poll
```

The first argument `monParamFile` is the file name for the parameter file of the monitor. The second argument `monOutputDir` is the output directory, to where text files containing populations and plots will be saved. The polling interval `poll` in seconds determines how long the process (or its children variator, selector WERA, selector WDFS) waits until it checks the state files again.

The new parameter file `monParamFile` consists of following entries, which are all required:

seed The pseudo-random number generator of the monitor is initialized with this value. As the variator is reset only once, the first random value[5] is used for the seed of the variator.

numberOfRuns removed

numberOfGenerations The maximal number of generations n_G for the whole optimization procedure. If this number is reached, the process will exit. If the variator has a similar parameter, one has to make sure that the value of the variator is at least as large as the value of the monitor. Otherwise, the variator would terminate too early.

userInteraction This integer value $n_{ui} \in (0, n_G]$ specifies after how many generations the user interaction should take place.

interactionType The interaction type can be either `stdin` or `file`. The former type reads the new preferences from standard input, the latter from a text file.

interactionFile The file name of the input file in the case of file-interaction.

maxHistorySize The maximum size of the history archive. If the value is negative, the size is not limited.

savePlots If set to 1, the plot of the current solutions will be saved to the output directory `monOutputDir`.

outputType removed

[5] calling `srand()`.

outputSet This parameter is used in a little bit different way as originally intended. Its value $n_{os} \in (0, n_g]$ determines after how many generations the program should write the current generated offspring population file to the output directory of the monitor, which is `monOutputDir`.

debug If set to 1, debug information is written to the standard output.

varProblem The problem of the variator. This parameter is required in order to draw the pareto front properly. If the problem is unknown, no pareto front will be plotted.

varExecPath Path to the executable of the variator.[6]

varParamFile Path to the parameter file of the variator.

varPisaPath Path to the PISA communication directory of the variator.

selWdfsExecPath Path to the executable of the WDFS selector.

selWdfsParamFile Path to the parameter file of the WDFS selector.

selWdfsPisaPath Path to the PISA communication directory of the WDFS selector.

selWeraExecPath Path to the executable of the WERA selector.

selWeraParamFile Path to the parameter file of the WERA selector.

selWeraPisaPath Path to the PISA communication directory of the WERA selector.

selWeraIAExec Path to the interaction executable of the WERA selector.

plotResultsExec Path to the plotResult executable.

5.2 Adapting the Protocol

5.2.1 Initialization

At the beginning of the program, the parameter file of the monitor and the configuration files of the variator and the selectors are read in. For a clean start up, the states are initialized with their reset values.

As it is easier to start just one program, the current process creates new child processes for the variator and both selectors as illustrated in figure (5.1).[7] That's the reason why their execution paths have to be declared in the parameter file.

Next, the parameter files of the selectors are stored temporarily in order to restore them during the termination phase.

[6] All file paths can be expressed relatively.

[7] See appendix B.2 for further advantages.

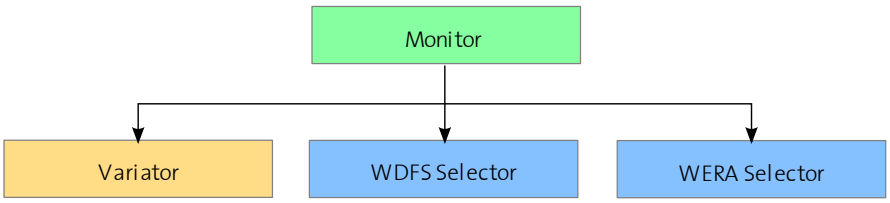

Figure 5.1: The monitor process creates 3 child processes.

Furthermore, a first user interaction is processed. The program forces the user to employ one or more uniform distributions using the WDFS selector in order to find preferably well distributed solutions as close to the front as possible.[8]

The variator and the WDFS selector are reset as a final step of the initialization phase (see figure (5.2)).

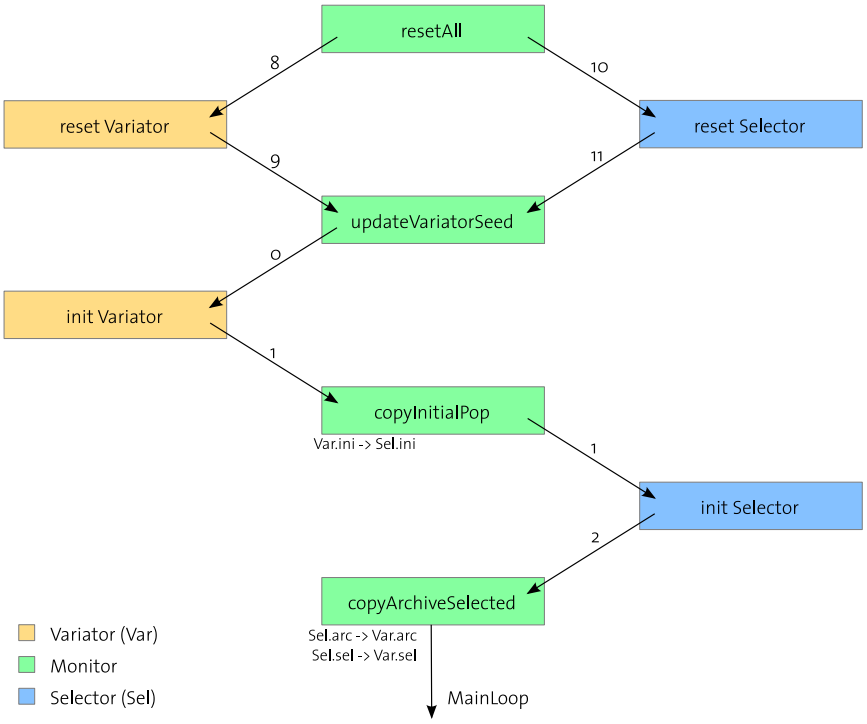

Figure 5.2: Reset procedure (the arrows are labeled with state numbers).

[8]This step is actually only done for the standard input method. As this is the default method, only this case is considered.

5.2.2 Main Loop

Within the main loop the evolutionary multiobjective optimization is processed. As in the PISA framework, this is basically done by moving the output file of the variator (`var`[9]) to the selector side and moving the output files of the current selector (`sel` and `arc`) to the variator side (this process is visualized in figure (5.3)). During these iterations, the state variable *currentGeneration* is incremented after each finished iteration. At this stage, only one selector is interaction with the monitor while the other is sleeping. During a user interaction, the selectors can be exchanged according to the new preferences.

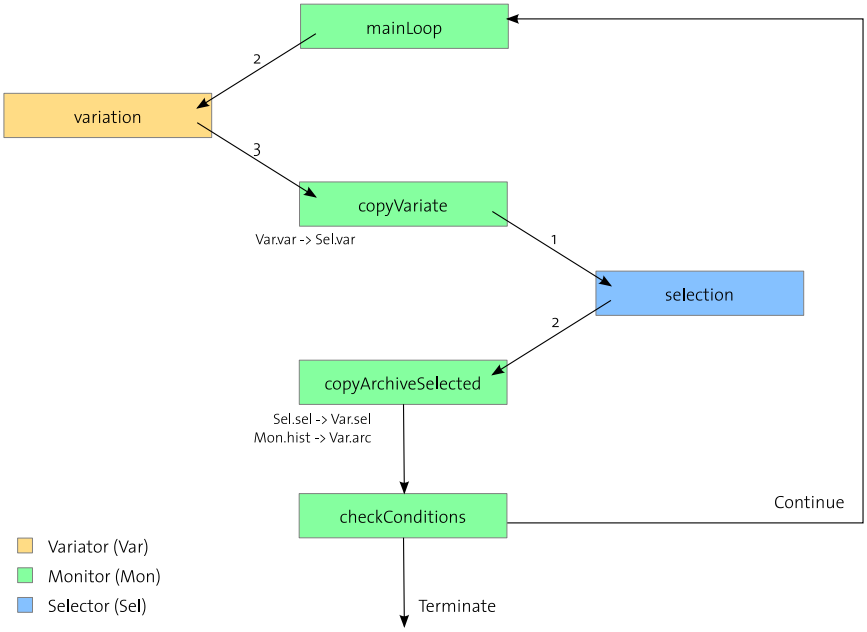

Figure 5.3: Main Loop: A round without user interaction (the arrows are labeled with state numbers).

Whenever this counter *currentGeneration* is a multiple of *userInteraction*[10], i.e.

$$currentGeneration \equiv k \cdot userInteraction,\ k \in \mathbb{N},$$

the interaction with the user begins (as shown in figure (5.4)). The exact procedure is covered in section 5.3. After a successful user interaction, the newly chosen selector is reset (`resetSel`).

It is important that the selector parses the parameter file and the other PISA files (`cfg`, `ini`, ...) in PISA state 1, otherwise the update of the selector doesn't work.[11]. If

[9] The used notation is dsecribed in appendix A.1.
[10] See section 5.1, parameter *userInteraction*.
[11] Selector WERA had to be corrected for this purpose. See appendix B.1.

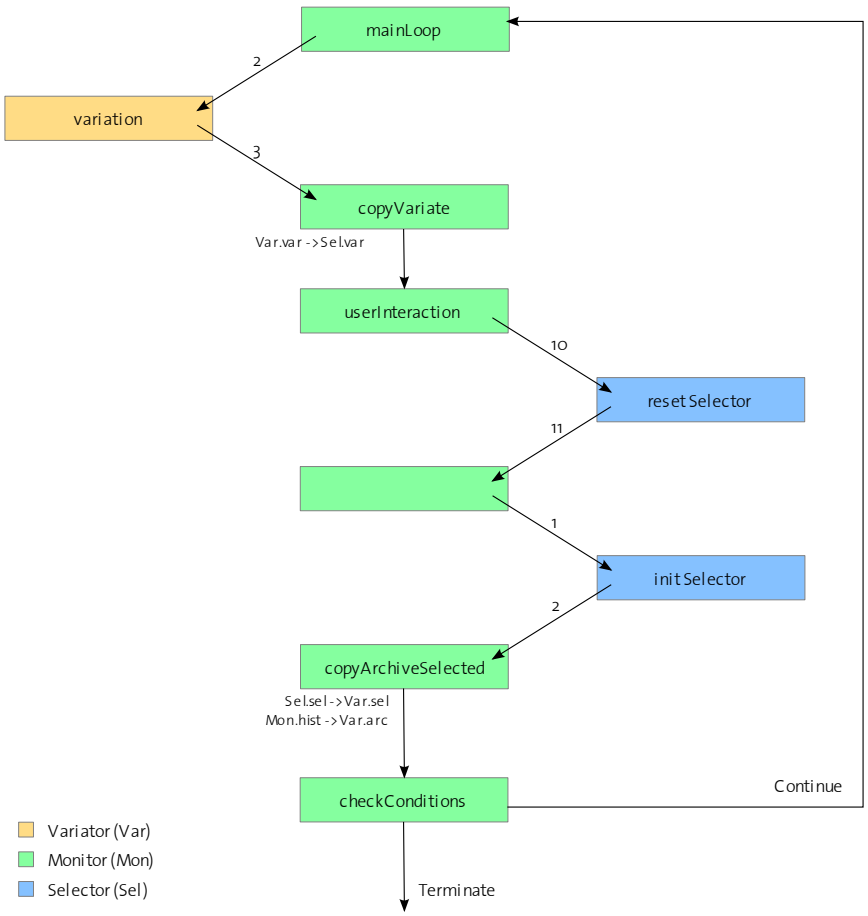

Figure 5.4: Main Loop: A round with user interaction (the arrows are labeled with state numbers).

the selector reads the files at the startup only, and not within its state machine loop, a possible change of those files will never affect the selector. A workaround could be to terminate the selector and restart it, but this would take more time.

5.2.3 Termination

The termination of the program is determined explicitly if the user wishes to exit or implicitly if the maximum number of generations (*numberOfGenerations*) is reached.

The backups of the parameter files are restored and if `alpha` had to be changed in the configuration files, the original value will also be recovered.

Finally, the monitor initiates the termination of the variator and the selectors (figure (5.5)).

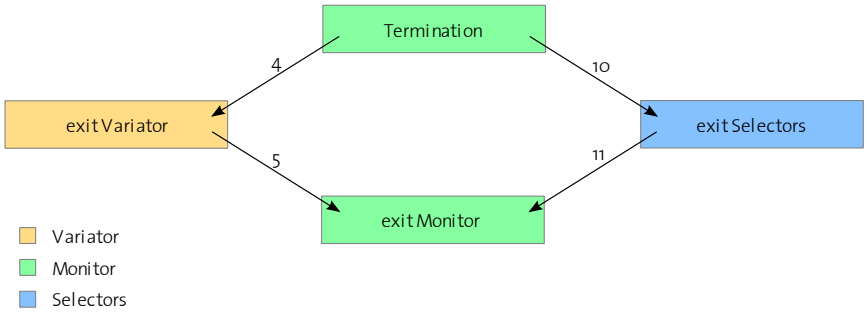

Figure 5.5: Termination (the arrows are labeled with state numbers).

5.3 Interaction

The first step of the user interaction is the presentation of the current solutions, i.e. the offspring population of the last variation. The control of the program is passed to the external executable specified by *plotResultsExec*. This program is called with the *string* arguments described in table (5.1). It needs so many information only because it plots the pareto front, which is problem-dependent. The parameter file of the current selector is used to draw the chosen distributions.

Argument	**Meaning**	**Possible values**
`ia_method`	Interaction Method	`wera` or `wdfs`
`number`	Current Generation	
`var_file`	Variation file `var`	
`param_file`	Parameter file of the selector	
`save_plots`	Save plots to `monOutputDir`	`1`=yes or `0`=no
`ia_type`	Interaction Type	`stdin` or `file`
`problem`	Variator Problem	ZDT$\{1,\ldots,6\}$, DTLZ$\{1,2,3,4,7\}$

Table 5.1: Input arguments for *plotResultsExec*.

After the presentation, the user is able to choose between the two selectors WDFS and WERA. In the case of the WERA selector, the new parameters are calculated from the preferences as explained in chapter 4. The parameters of the WDFS selector are entered directly, which needs knowledge about the several possible distributions. These distributions and their parameters are described in section 2.3.

Before resetting the newly chosen selector, the monitor creates a new `ini` file for the selector containing the population of the *history* archive.[12] If the new population size `alpha` is different from the old one, the configuration file `cfg` will be updated with this new value.

[12] See section 5.4.

The newly obtained parameters for the selector are written to the output log file `monOutputDir/log.txt` after each successful interaction with the user. This enables the possibility to comprehend the decision-making later.

5.3.1 Standard Input Interaction

The default interaction type `stdin`[13] is used whenever direct interaction is required. In most scenarios, the user does't know the exact outcome of an optimization procedure. This method enables the user to set his / her preferences after each interaction cycle. In the case of two objectives, the current set of solutions is presented in a mathematical plot.

The parameters for the WDFS selector are read from the standard input. The user can determine the number of distributions first and then enter the specifications. Table (5.2) is an example of such an input. If the number of distributions is equal to zero, the old parameters are reused. After the verification of the data, the parameters are replaced in the parameter file of the WDFS selector.

Uniform Distributions	**Gaussian Distributions**

`User Interaction #0`
`**********`
`Number of uniform distributions: 2`
`Parameter Set 1:`
`  alpha (between 0 and 1): 0.75`
`  lower bound:`
`    lower(1): 0`
`    lower(2): 0`
`  upper bound:`
`    upper(1): 0.5`
`    upper(2): 0.5`
`  Parameter Set 2:`
`  alpha (between 0 and 1): 0.25`
`  lower bound:`
`    lower(1): 0`
`    lower(2): 0.5`
`  upper bound:`
`    upper(1): 1`
`    upper(2): 1`
`  update selector's parameter file`
	`User Interaction #1`
	`**********`
	`Interaction Method (wdfs | wera | unif | exit): wdfs`
	`  Number of preference points: 2`
	`  Parameter Set 1:`
	`    alpha (between 0 and 1): 1`
	`    vector mu:`
	`      mu(1): 0.81`
	`      mu(2): 0.1`
	`    vector dir:`
	`      dir(1): 1`
	`      dir(2): 0`
	`    sigma_eps: 0.01`
	`    sigma_t: 0.05`
	`  Parameter Set 2:`
	`    alpha (between 0 and 1): 0.5`
	`    vector mu:`
	`      mu(1): 0.11`
	`      mu(2): 0.67`
	`    vector dir:`
	`      dir(1): 0`
	`      dir(2): 1`
	`    sigma_eps: 0.01`
	`    sigma_t: 0.05`

Table 5.2: A typical input scenario for the WDFS selector.

In the case the user decided the WERA selector, the external program *selWeraIAExec* is called (input arguments are specified in table (5.3). The program gets the new preferences from the user directly. This process is covered in chapter 3. Afterwards,

[13] See section 5.1, parameter *interactionType*.

the new parameters for the WERA selector are calculated as mentioned in chapter 4 and the parameter file is updated accordingly.

Argument	Meaning
`var_file`	Variation file `var`
`param_file`	Parameter file of the WERA selector

Table 5.3: Input arguments for *selWeraIAExec*.

Fading of Previous Parameters

During the input of Gaussian parameter sets, it is possible to fade out the old distributions instead of disappearing directly. This regression is controlled with these operands:

- Fade-out factor $f \in (0, 1)$
- Threshold $T \in (0, 1]$

For each old parameter set, its weight α is multiplied with f. If $\alpha \cdot f > T$, the new value is updated, otherwise the parameter set is removed.

5.3.2 File Input Interaction

The interaction type `file` simplifies automatic test cases, but requires more knowledge about possible outcomes. This is because the choice of preferences is actually done before the interaction.

The input file contains in each line a parameter set for each interaction and looks like[14]:

```
'round' WS PosInt WS 'method' WS MSTR WS PSTR
```

The positive integer indicates the interaction cycle beginning from 0. More than one distributions per round are allowed, but they have to be for the same selector. `MSTR` can either be `'wdfs'` or `'wera'` indicating the preferred selector. If `MSTR` is equal to `'exit'`, the monitor will exit immediately. The last string `PSTR` contains the distribution for the specified selector. Contrary to the standard input method, the parameter values are not checked for their correctness.

5.4 Saving the History

The *history* archive is a feature to improve the performance of the optimization. The basic idea is to add all non-dominated elements of the current offspring population to this archive before the interaction takes place. While the newly chosen selector is reset,

[14] See appendix A.3 for further details.

this *history* archive becomes the initial population. As a consequence the selector must be able to read the new configuration after the reset.

Even though the selector might not have been used during the last optimization procedure, it has a good initial population thanks to the *history* archive. A further advantage is that the initial population is not too concentrated on a specific region in the case the user entered these preferences. This is important if the user is unsteady about his / her statements and would like to focus the search on a different region.[15] The process of refocusing should be done in possibly few generations to increase performance.

The normal procedure of the PISA protocol is to move the archive file `arc` from the selector to the variator side.[16] This step has to be modified. The archive file of the selector contains the indices of those individuals the variator should keep. As the variator must not remove the individuals of the *history* archive as well, both populations are merged together and written to the archive file of the variator.

For the next optimization round, the initial population size `alpha` is updated according to the new size of the *history* archive. The monitor has to make sure that this value does not drop below the parent population size `mu`. As long as the current *history* size is lower than `mu`, dominated solutions are added to this archive as well. But this is only the case in early interaction rounds as the archive size is increasing relatively fast.

If the size of the *history* list gets larger than *maxHistorySize*, the archive has to be reduced. This is done with the WDFS selector:

- The *history* archive is written to the `ini` file of the selector.
- The user has to enter a uniform distribution.
- The WDFS selector is reset and started for one cycle.
- All elements in the archive that aren't in the resulting population in file `sel` will be removed.

5.5 Portability

The new monitor application was developed and tested on a Linux distribution although the PISA aims to be platform-independent. As mentioned in section 5.2.1, the monitor starts its children in order to increase comfort. But the creation of a new process is very system-dependent. Although this functionality has been implemented for Windows® as well, it has never been tested. It would be possible to remove this feature and start the several processes manually or alternatively with the help of a script file.

[15] Chapter 6 analyses the dynamic behavior in detail.
[16] Figure (5.3), function `copyArchiveSelected`.

Chapter 6

Dynamic Behavior

6.1 The Test Setup

For our test case, the ZDT1 [14] test problem is used with these configurations:

cfg	**dtlz_param.txt**	**javasel_param.txt**
alpha 40 mu 40 lambda 40 dim 2	problem ZDT1 seed * number_decision_variables 2 maxgen 1000 outputfile ../variator/dtlz/dtlz_output.txt individual_mutation_probability 1 individual_recombination_probability 1 variable_mutation_probability 1 variable_swap_probability 0.5 variable_recombination_probability 1 eta_mutation 20 eta_recombination 15 use_symmetric_recombination 1	seed 1000 nrOfSamples 30000 dist *

Table 6.1: The setup for the test case. Entries with an asterisk are replaced during runtime.

Only the WDFS selector is used in this test scenario. A user interaction is done every 50^{th} generation with the following distributions:

1. round: uniform distribution: $w(\mathbf{z}) = \begin{cases} 1, & \mathbf{z} \in [0,1] \times [0,1] \\ 0, & \text{else} \end{cases}$
2. round: Gaussian distribution: $\boldsymbol{\mu_1} = (0.81, 0.10)^T, \mathbf{t_1} = (1,0)^T, \sigma_\epsilon = 0.01, \sigma_t = 0.05$
3. round: Gaussian distribution: $\boldsymbol{\mu_2} = (0.11, 0.67)^T, \mathbf{t_2} = (0,1)^T, \sigma_\epsilon = 0.01, \sigma_t = 0.05$

The preference points $\boldsymbol{\mu_1}, \boldsymbol{\mu_2}$ are close to the pareto front. An example of the distributions and resulting solutions is shown in figure (6.1). The monitor is started 12 times with different seeds. The same procedure is repeated without the *history* archive

implemented. In this case, the offspring population file `var` of the variator is taken as new initial population file `ini` for the selector. This is why `alpha` has to be equal to `lambda` for this test case, which seems to be a condition of the DTLZ variator anyway.

To automate the test procedure, the interaction type is set to `file`. The input file contains the distribution parameters for each round. This is the resulting input file based on the distributions above:

```
round 0 method wdfs dist uni 1.0 lower 0.0 0.0 upper 1.0 1.0
round 1 method wdfs dist gau 1.0 mu 0.81 0.10 dir 1.0 0.0 sigma_eps 0.01 sigma_t 0.05
round 2 method wdfs dist gau 1.0 mu 0.11 0.67 dir 0.0 1.0 sigma_eps 0.01 sigma_t 0.05
round 3 method exit
```

6.2 Test Results

The parameter *outputSet* of the monitor is set to 1 which means that after every generation the offspring population is written to the output directory. The files are read in and analyzed. For each seed the number after how many generations 90% of the solutions are close to the preference point is calculated. The closeness is defined as a circle around the middle point $\boldsymbol{\mu_2}$:

$$(z_1 - \mu_{2,1})^2 + (z_2 - \mu_{2,2})^2 \leq 0.9 \cdot 3\sqrt{\sigma_\epsilon^2 + \sigma_t^2}$$

with history	11	14	11	13	9	13	12	19	11	11	11	12
without history	40	27	43	41	32	24	47	24	31	45	28	34

Table 6.2: Number of generations needed till the solutions are close to the new preference points.

Figure (6.2) is a visualization of the numerical results from table (6.2). The median is plotted in red. For the monitor with *history* archive, it takes 11.4167 generations on average to come close to the preference point, whereas without *history* archive, 34.6667 generations are needed.

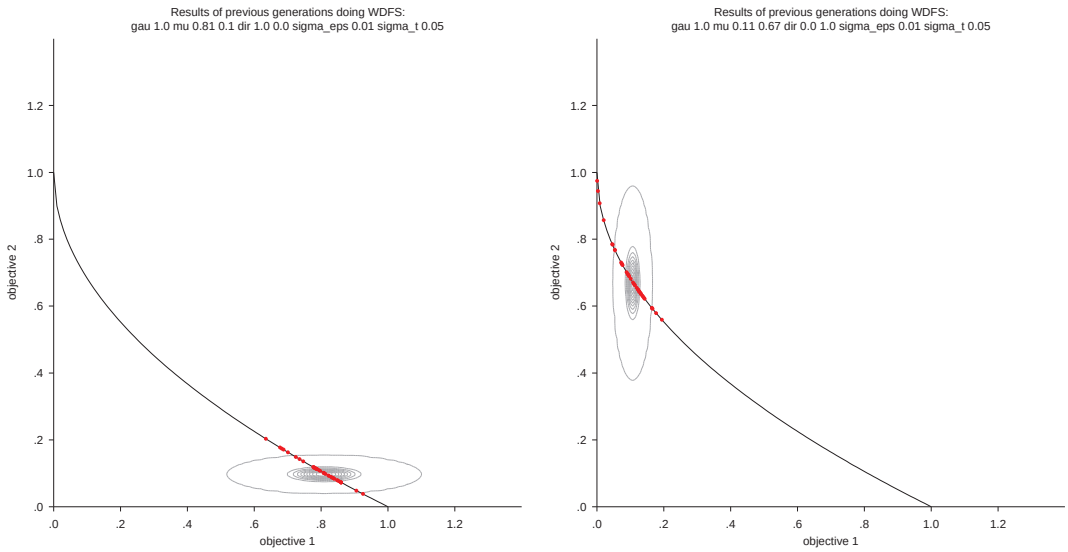

Figure 6.1: Example for two interaction cycles using the *history* archive. The Gaussian distributions are plotted in gray, the resulting populations in red. Left: 6 generations after interaction. Right: 14 generations after interaction.

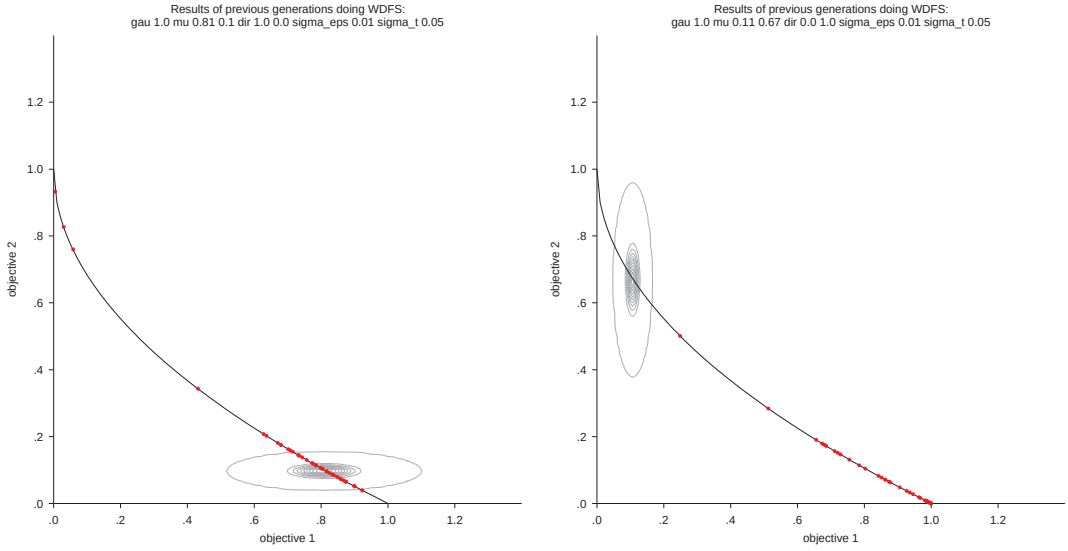

Figure 6.2: Example for two interaction cycles without *history* archive. The same seed as in figure (6.1) is used. The progress after the same number of generations is shown to compare the outcome.

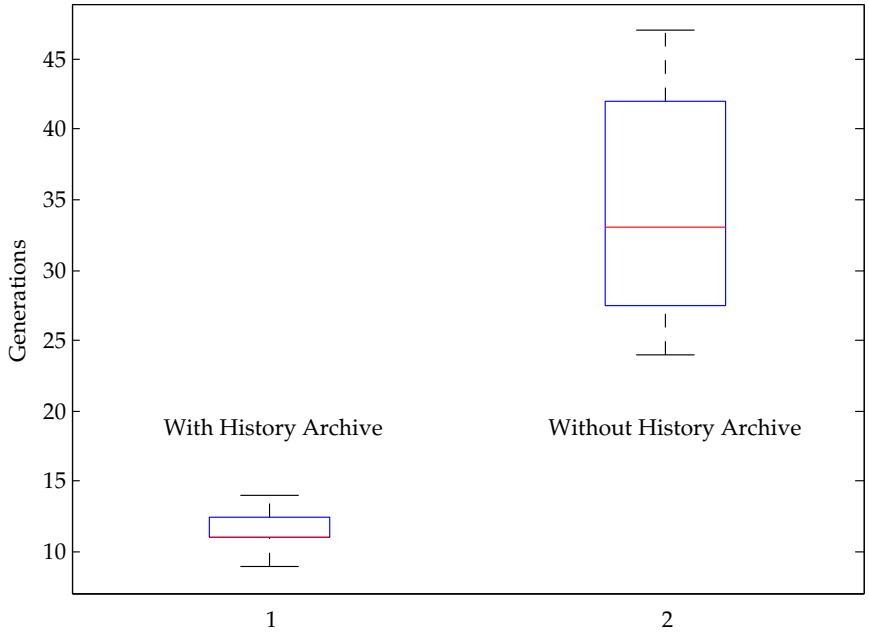

Figure 6.3: The resulting distribution of generations needed to reach the region of the preference point.

Chapter 7

Conclusion & Outlook

Originally, the EMO and MCDM steps were separated into two parts. In this thesis, we designed an interaction cycle embedding the user interaction into the EMO procedure. This enables the expert to perform a focused optimization after each specified amount of generations within the EMO. We developed a suitable interaction method and different proposals how to process the captured preference model, such that the used selector can interpret them. Additionally, our framework provides a history archive, which offers a speed up during focus changes.

Emphasizing the population in a specific spot of the pareto front delivers more potential solutions to choose from. So far, this focusing had to be done before the optimization, which required a profound knowledge of the problem. The new approach allows a complete new proceeding. Our framework enables the user to educate him- / herself about the problem during the search procedure. This makes the decision making more efficient, especially for high-complexity problems, as the user can change his region of interest during runtime.

It seems to be difficult to find test cases, which compare our interaction cycle with existing methods. Due to the fact that the user can change his / her preferences within the optimization procedure, similar conditions for comparison are required.

The RABI (Rays by Angle Bisector), WELI (Weights Linearly Interpolated) and SIFA (Scalarizing Function Approach) approaches are proposed to process the captured preferences. Up to a few limitations, the first two proposals work nicely to transform the user ratings into a weight distribution for the pareto front. The latter one couldn't age, as no suitable scalarizing function was available at that time.

The analysis of the dynamic behavior disclosed the essential need of a history solution. Changing the focus region with the implemented history needs approximately three times less generations to change to a new area of interest on the pareto-front, using the ZDT1 problem.

For further work in this area, we propose that this framework should be tested with more complex problems, which would evaluate the usefulness for real world problems. It would be interesting, if one could expand this procedure to more than two objectives, involving the adaption of the WERA selector and user interaction method.

To realize the SIFA approach, profound research is needed to find suitable scalarizing functions. For our requirements they need more degrees of freedom to enable a representation of all possible user preferences.

Appendix A

PISA

A.1 PISA Files

In the following, we use this notation taken from PISA [2].

`cfg`	configuration file
`sta`	state file
`ini`	initial population file
`var`	offspring population file
`sel`	selected individuals file
`arc`	archived individuals file

A.2 PISA Sizes

The configuration file for each selector and variator contains the following values: The

`alpha`	initial population size
`mu`	parent population size
`lambda`	offspring population size
`dim`	number of objectives

user has to make sure, that the corresponding values are the same for each program.

A.3 Parameter File Syntax

The notation is same as in the PISA specification [2].

```
WS     := (Space | Newline | Tab)+
Digit  := '0-9'
Exp    := ('E' | 'e') ('+'? | ('-'?) Digit+
```

```
PosInt := '1-9' Digit*
Float  := (Digit+ '.' Digit*) | ('.' Digit+ Exp?) | (Digit+ Exp)
MSTR   := ('wdfs' | 'wera' | 'exit')
PSTR   := (PWDFS | PWERA)
PWDFS  := (PGAU | PUNI)
```

WDFS Selector

The parameters $(\mu_1, \dots \mu_k)^T, (t_1, \dots t_k)^T, \sigma_\epsilon, \sigma_t$ and $(b_1^l, \dots b_k^l)^T, (b_1^u, \dots b_k^u)^T$ have the format:

```
PGAU   := 'dist gau' WS Float WS 'mu' WS Float+ WS 'dir' WS
          Float+ WS 'sigma_eps' Float 'sigma_t' Float
PUNI   := 'dist uni' WS Float WS 'lower' WS Float+ WS 'upper'
          WS Float+
```

WERA Selector

```
PWERA  := 'rays' WS (PosInt WS)+ Float (WS Float WS Float)+
```

Appendix B

Framework

B.1 Changes to the WERA Selector

The original WERA selector doesn't read the configuration file `cfg` in state 1 but before. So we had to move this step to the state machine loop:

`StateMachine.java:`

```
public void run() {
...
  if (state == 1) /* inital selection */
  {
    io.initialize(ParameterSet.getParameterFileName(), ParameterSet.getCommFilePath());
    popNew.empty();
    if (io.readIni(popNew, ParameterSet.getDim(), ParameterSet.getAlpha()) == false) {
      iteration();
      state = 2;
      io.writeFlag(stateFile, state);
    }
  }
...
}
```

As the function `readCfgFile()` called from function `io.initialize()` only updates the PISA sizes if they are not set[1], the values are reset to zero before the configuration file is read.

[1] The values `alpha`, `mu`, `lambda` and `dim` are not set if they are all equal to zero.

IO.java:

```
private void readCfgFile() throws Exception {
  ...
  // reset sizes alpha, mu, lambda and dim to zero
  ParameterSet.resetSizes();
  ...
}
```

B.2 System Requirements

As mentioned in section 5.3, the new monitor program uses external functionalities of MATLAB®, which are compiled as standalone applications. For this purpose, the MATLAB® Compiler™ is required [15]. This means, that the Compiler Runtime must be installed in order to run the resulting executables.[2]. The "main" functions `plotResult.m` and `wera.m` are compiled with this command:

```
mcc -mv plotResults/plotResults.m  -d plotResults/
mcc -mv wera/wera.m -d wera/
```

The resulting binaries are refered to *plotResultsExec* and *selWeraIAExec*. They require the MATLAB® runtime libraries for a successful execution. These libraries are installed together with the MATLAB® Compiler™. But the programs need exact knowledge about the location of the libraries. This is done by the environmental variable `LD_LIBRARY_PATH`. For a Linux distribution, this would typically look like:

```
MATLAB_CR_PATH=/usr/share/matlab/mcr/v79;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_CR_PATH/runtime/glnxa64
                :$MATLAB_CR_PATH/bin/glnxa64
                :$MATLAB_CR_PATH/sys/os/glnxa64
                :$MATLAB_CR_PATH/sys/java/jre/glnxa64/jre/lib/amd64/
                :$MATLAB_CR_PATH/sys/java/jre/glnxa64/jre/lib/amd64/native_threads
                :$MATLAB_CR_PATH/sys/java/jre/glnxa64/jre/lib/amd64/server;
export LD_LIBRARY_PATH
```

Both selectors are implemented in Java and work with the MATLAB® Builder™ JA [16]. The environmental variable `LD_LIBRARY_PATH` has to be extended with the directories of the runtime libraries of the MATLAB® Builder™ JA:

[2]The MATLAB® command `mcrinstaller` displays the path to the installation file.

```
MATLAB_ROOT_PATH=/usr/share/matlab;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_ROOT_PATH/sys/os/glnxa64
                :$MATLAB_ROOT_PATH/bin/glnxa64
                :$MATLAB_ROOT_PATH/extern/lib/glnxa64
                :$MATLAB_ROOT_PATH/sys/java/jre/glnxa64/jre/lib/amd64/native_threads
                :$MATLAB_ROOT_PATH/sys/java/jre/glnxa64/jre/lib/amd64/server
                :$MATLAB_ROOT_PATH/sys/java/jre/glnxa64/jre/lib/amd64;
export LD_LIBRARY_PATH
```

If one path is missing, an error-free execution is not guaranteed. The procedure of setting the right variables seems to be a bit complicated. For this reason, the monitor is started with a shell script, which sets all required environmental variables. Then, the monitor creates its child processes, which inherit these variables. This script is shown in the next section.

B.3 Start-Up Script

```
#!/bin/bash

# killing old processes ###################################################
type pkill > /dev/null
if [ $? != 0 ]; then
exit
fi

pkill -f hype_wera
pkill -f hype_wdfs
pkill -f dtlz

# getting processor information ###########################################
uname -m | grep 64 > /dev/null
if [ $? = 0 ]; then
glnx="glnxa64";
arch="amd64";
else
glnx="glnx86";
arch="i386";
fi

# setting up environmental variables ######################################

# MATLAB
MATLAB_ROOT_PATH=/usr/share/matlab;
MATLAB_CR_PATH=/usr/share/matlab/mcr/v79;

# PATH
PATH=$PATH:$MATLAB_ROOT_PATH/bin;
PATH=$PATH:$MATLAB_ROOT_PATH/bin/$glnx;
PATH=$PATH:$MATLAB_ROOT_PATH/toolbox/compiler/mcr/compiler;
export PATH
```

```
#echo "PATH: $PATH"

# Libraries for matlab runtime
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_ROOT_PATH/sys/os/$glnx;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_ROOT_PATH/bin/$glnx;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_ROOT_PATH/extern/lib/$glnx;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_ROOT_PATH/sys/java/jre/$glnx/jre/lib/
                $arch/native_threads;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_ROOT_PATH/sys/java/jre/$glnx/jre/lib/
                $arch/server;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_ROOT_PATH/sys/java/jre/$glnx/jre/lib/
                $arch;
# Libraries for matlab compiler runtime
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_CR_PATH/runtime/$glnx;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_CR_PATH/bin/$glnx;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_CR_PATH/sys/os/$glnx;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_CR_PATH/sys/java/jre/$glnx/jre/lib/
                $arch/
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_CR_PATH/sys/java/jre/$glnx/jre/lib/
                $arch/native_threads;
LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$MATLAB_CR_PATH/sys/java/jre/$glnx/jre/lib/
                $arch/server;
export LD_LIBRARY_PATH
#echo "LD_LIBRARY_PATH: $LD_LIBRARY_PATH"

# X11
XAPPLRESDIR=$MATLAB_ROOT_PATH/X11/app-defaults
export XAPPLRESDIR
#echo "XAPPLRESDIR: $XAPPLRESDIR"


# setting up parameters ##############################################################
monParam="monitor_param.txt"
monOutputDir="pisa_files/"

if [ $# = 0 ]; then
poll=0.5
else
poll=$1
fi

echo "  monParam:      $monParam"
echo "  monOutputDir:  $monOutputDir"
echo "  polling:       $poll"
echo "  monitor started..."

# starting process ###################################################################
bin/monitor $monParam $monOutputDir $poll
```

Bibliography

[1] C. A. Coello, "A short tutorial on evolutionary multiobjective optimization," 2001.

[2] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA—A Platform and Programming Language Independent Interface for Search Algorithms," in *Conference on Evolutionary Multi-Criterion Optimization (EMO 2003)*, ser. LNCS, C. M. Fonseca *et al.*, Eds., vol. 2632. Berlin: Springer, 2003, pp. 494–508.

[3] A. Eiben and J. Smith, *Introduction to Evolutionary Computation - Chapter 2*, 1st ed., ser. Natural Computing Series. Springer, 2003.

[4] L. Thiele, "Using the monitor in pisa," *Computer Engineering and Networks Lab (TIK), ETH Zurich.*

[5] J. Bader and E. Zitzler, "A Hypervolume-Based Multiobjective Optimizer for High-Dimensional Objective Spaces," in *Conference on Multiple Objective and Goal Programming (MOPGP 2008)*, ser. Lecture Notes in Economics and Mathematical Systems. Springer, 2009, to appear.

[6] E. Zitzler, D. Brockhoff, and L. Thiele, "The Hypervolume Indicator Revisited: On the Design of Pareto-compliant Indicators Via Weighted Integration," in *Conference on Evolutionary Multi-Criterion Optimization (EMO 2007)*, ser. LNCS, S. Obayashi *et al.*, Eds., vol. 4403. Berlin: Springer, 2007, pp. 862–876.

[7] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Articulating User Preferences in Many-Objective Problems by Sampling the Weighted Hypervolume," in *Genetic and Evolutionary Computation Conference (GECCO 2009)*, G. Raidl *et al.*, Eds. ACM, 2009, to appear.

[8] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Investigating and Exploiting the Bias of the Weighted Hypervolume to Articulate User Preferences," in *Genetic and Evolutionary Computation Conference (GECCO 2009)*, G. Raidl *et al.*, Eds. ACM, 2009, to appear.

[9] P. Korhonen, *Multiple Criteria Decision Analysis: State of the Art Surveys - Chapter 16.* Springer New York, 2005, vol. Volume 78.

[10] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Articulating User Preferences in Many-Objective Problems by Sampling the Weighted Hypervolume," in *Genetic and Evolutionary Computation Conference (GECCO 2009)*, G. Raidl *et al.*, Eds. ACM, 2009, to appear.

[11] [Online]. Available: http://en.wiktionary.org/wiki/utility_function

[12] A. M. Geoffrion and A. F. J. S. Dyer, "An interactive approach for multi-criterion optimization, with an application to the operation of an academic department," *Management science 1972 19: 357-368*, 1972.

[13] K. Miettinen and M. M. Mäkelä, "On scalarizing functions in multiobjective optimization," *OR Spectrum*, pp. 193 – 213, 2002.

[14] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable test problems for evolutionary multi-objective optimization," in *Evolutionary Computation Based Multi-Criteria Optimization: Theoretical Advances and Applications*, Springer-Verlag, 2005.

[15] [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/compiler/

[16] [Online]. Available: http://www.mathworks.com/access/helpdesk/help/toolbox/javabuilder/