Charles University in Prague
Faculty of Mathematics and Physics

**MASTER THESIS**

Martin Pilát

# Parallel Evolutionary Algorithms for Multiobjective Optimization

Department of Theoretical Computer Science and Mathematical Logic

Supervisor: Mgr. Roman Neruda, CSc.
Study Program: Computer Science

2009

I would like to thank to Mgr. Roman Neruda, CSc. for his valuable advice and for supervising this work, to Mgr. Milan Hladík for his information on various methods for multiobjective optimization and to Petra Šachová for the proof-reading of the text.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce a jejím zveřejňováním.

I hereby declare that I wrote my master thesis on my own and with the use of the cited sources only. I agree with the lending of the work.

Prague April 15, 2009 Martin Pilát

# Contents

**1 Introduction and Motivation** **1**
1.1 Applications and Case Studies . . . . . . . . . . . . . . . . . . 2
1.2 Problem Definition . . . . . . . . . . . . . . . . . . . . . . . . 3

**2 Methods Used in Multiobjective Optimization** **6**
2.1 Function Aggregation . . . . . . . . . . . . . . . . . . . . . . . 6
2.2 Normal-Boundary Intersection . . . . . . . . . . . . . . . . . . 7
2.3 Particle Swarm Optimization . . . . . . . . . . . . . . . . . . 7
2.4 Ant Colony Optimization . . . . . . . . . . . . . . . . . . . . . 8

**3 Quality Indicators** **9**
3.1 $I_{\epsilon+}$ – Additive $\epsilon$ Indicator . . . . . . . . . . . . . . . . . . . 9
3.2 $\mathcal{S}$ – Hypervolume Metric ($\mathcal{S}$ metric) . . . . . . . . . . . . . . 10
3.3 $\mathcal{C}$ – Coverage of Two Set . . . . . . . . . . . . . . . . . . . . 11
3.4 Convergence Metric . . . . . . . . . . . . . . . . . . . . . . . . 11

**4 Multiobjective Evolutionary Algorithms** **12**
4.1 Aggregation Based Algorithms . . . . . . . . . . . . . . . . . . 13
4.1.1 VEGA – Vector Evaluated Genetic Algorithm . . . . 14
4.1.2 MSOPS – Multiple Single Objective Pareto Sampling 14
4.2 Domination Based Algorithms . . . . . . . . . . . . . . . . . . 15
4.2.1 NSGA – Nondominated Sorting Genetic Algorithm . 16
4.2.2 NSGA-II . . . . . . . . . . . . . . . . . . . . . . . . . 17
4.2.3 SPEA – Strength Pareto EA . . . . . . . . . . . . . . 18
4.2.4 SPEA2 . . . . . . . . . . . . . . . . . . . . . . . . . . 19
4.2.5 SPEA2+ . . . . . . . . . . . . . . . . . . . . . . . . . 20
4.3 Indicator Based EAs . . . . . . . . . . . . . . . . . . . . . . . 21
4.3.1 IBEA – Indicator Based EA . . . . . . . . . . . . . . 22
4.3.2 SMS-EMOA – $\mathcal{S}$ Metric Selection EMOA . . . . . . . 23

4.3.3 SPAM – Set Preference Algorithm for MOO . . . . . 24
4.3.4 HypE – Hypervolume Estimation Based MOGA . . . 25
4.4 Performance Discussion . . . . . . . . . . . . . . . . . . . . . . . 26
4.4.1 Aggregation vs. Domination Based EAs . . . . . . . 26
4.4.2 Domination vs. Indicator Based EAs . . . . . . . . . 27

**5 Parallel Multiobjective Evolutionary Algorithms 28**
5.1 Models of Parallel Evolutionary Algorithms . . . . . . . . . 28
5.1.1 Master-Slave Model . . . . . . . . . . . . . . . . . . . 29
5.1.2 Island Model . . . . . . . . . . . . . . . . . . . . . . 29
5.1.3 Cellular Model . . . . . . . . . . . . . . . . . . . . . 30
5.2 DRMOGA – Divided Range MOGA . . . . . . . . . . . . . 30
5.3 DCMOGA – Distributed Cooperation Model of MOGA . . . 31
5.4 MOGADES – MOGA with Distributed Environment Scheme 31
5.5 MRMOGA – Multiple Resolutions MOGA . . . . . . . . . . 33
5.6 Parallelization Using Clustering . . . . . . . . . . . . . . . . 34
5.7 MOCell – Cellular MOGA . . . . . . . . . . . . . . . . . . . 35

**6 Using Fuzzy Constraints in Parallel MOGA 36**
6.1 Test Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . 37
6.2 Test Results . . . . . . . . . . . . . . . . . . . . . . . . . . 37

**7 Heterogeneous Island Model as a Hybridizing Technique 42**
7.1 Test Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . 42
7.2 Test Results . . . . . . . . . . . . . . . . . . . . . . . . . . 43

**8 Using SOGA as a Local Search Technique in MOGA 47**
8.1 Algorithm Description . . . . . . . . . . . . . . . . . . . . . 47
8.2 Test Setup . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
8.3 Test Results . . . . . . . . . . . . . . . . . . . . . . . . . . 50

**9 Conclusions and Future Work 60**

**A Test problems 62**

# List of Algorithms

1 Evolutionary algorithm . . . . . . . . . . . . . . . . . . . . . . . . 12
2 Vector Evaluated Genetic Algorithm . . . . . . . . . . . . . . . 14
3 Multiple Single Objective Pareto Sampling . . . . . . . . . . . 15
4 Nondominated Sorting Genetic Algorithm . . . . . . . . . . . 17
5 Nondominated Sorting Genetic Algorithm II . . . . . . . . . . 18
6 Strength Pareto Evolutionary Algorithm . . . . . . . . . . . . 19
7 Strength Pareto Evolutionary Algorithm 2+ . . . . . . . . . . 20
8 Indicator Based Evolutionary Algorithm . . . . . . . . . . . . 23
9 $\mathcal{S}$ Metric Selection EMOA . . . . . . . . . . . . . . . . . . 24
10 Set Preference Algorithm . . . . . . . . . . . . . . . . . . . . . . 25
11 Hypervolume Estimation Based MOGA . . . . . . . . . . . . . 26
12 Divided Range MOGA . . . . . . . . . . . . . . . . . . . . . . . 31
13 Distributed Cooperation Model of MOGA . . . . . . . . . . . 32
14 MOGA with Distributed Environment Scheme . . . . . . . . 32
15 Multiple Resolutions Multi-Objective Genetic Algorithm . . 34
16 Parallelization Using Clustering . . . . . . . . . . . . . . . . . 34
17 Cellular Multiobjective Genetic Algorithm . . . . . . . . . . . 35
18 MOGASOLS – MOGA island . . . . . . . . . . . . . . . . . . . 48
19 MOGASOLS – SOGA island . . . . . . . . . . . . . . . . . . . . 49

Název práce: Paralelní evoluční algoritmy pro vícekriteriální optimalizaci
Autor: Bc. Martin Pilát
Katedra (ústav): Katedra teoretické informatiky a matematické logiky
Vedoucí bakalářské práce: Mgr. Roman Neruda, CSc.
e-mail vedoucího: roman@cs.cas.cz

Abstrakt: V předložené práci studujeme možnosti paralelizace evolučních algoritmů pro vícekriteriální optimalizaci (MOGA). Uvádíme přehled existujících sekvenčních a paralelních MOGA a navrhujeme tři další metody: FCMOGA – MOGA s fuzzy omezujícími podmínkami, HIMOGA – MOGA používající heterogenní ostrovní model a MOGASOLS – MOGA používající jednokriteriální evoluční algoritmus jako prostředek lokálního prohledávání. Tyto algoritmy vyhodnocujeme na sadě testovacích problémů a porovnáváme je s existujícími MOGA.

Klíčová slova: evoluční algoritmy, paralelizace, vícekriteriální optimalizace

Title: Parallel Evolutionary Algorithms for Multiobjective Optimization
Author: Bc. Martin Pilát
Department: Department of Theoretical Computer Science and Mathematical Logic
Supervisor: Mgr. Roman Neruda, CSc.
Supervisor's e-mail address: roman@cs.cas.cz

Abstract: In the present work we study the options for parallelization of evolutionary algorithms for multiobjective optimization (MOGA). We provide the overview of existing sequential and parallel MOGAs and we propose three other methods: FCMOGA – MOGA with fuzzy constraints, HIMOGA – heterogeneous island MOGA, and MOGASOLS – MOGA with single objective local search. We test these algorithms on a set of benchmark problems and compare them with existing MOGAs.

Keywords: evolutionary algorithms, parallelization, multiobjective optimization

# Chapter 1

# Introduction and Motivation

Many real-life problems require optimizing multiple conflicting objectives: for example, if you have to travel to a distant place, you can choose from a variety of different means of transport, e.g. train, bus, and plane. If you choose the plane, you get fast where you need, however, you will have to pay much more for the trip than if you choose the train. If you choose the bus, it could take longer and be more expensive than the train. Given the two objectives, price and time, you probably will not choose the bus, as it is worse than the train with respect to both of them. However, deciding between the train and the plane is not so easy, one is cheaper, the other is faster. With no additional information, we cannot decide, which one is better, and so, both of them can be considered as an optimal solution to our problem of choosing the best mean of transport (they are so called Pareto optimal solutions). This is only a simple example, more realistic ones can be found in Section 1.1 on applications of multiobjective optimization.

Despite its simplicity, the example implies, that solving multiobjective optimization problems is an important and difficult task, as the number of optimal solutions may be very large (or even infinite). Over past decades, various approaches were presented to deal with this problem, among them the population based ones gained popularity, as they are able to find more optimal solutions in a single run. However, these approaches can be computationally expensive, and therefore various parallelization techniques are used to speed them up.

The goal of this work is to study the parallel multiobjective evolutionary algorithms (MOGA), and to provide new methods for multiobjective optimization based on this study.

To fulfill this goal, we first, at the end of this chapter provide some basic definitions, needed in the rest of the work.

In Chapter 2, we discuss methods used in multiobjective optimization with the exception of multiobjective genetic algorithms. Because comparing the solutions of multiobjective optimization is not an easy task, we describe some indicators used for it in Chapter 3.

Chapter 4 contains examples of multiobjective evolutionary algorithms. We divide these algorithms into three groups according to the method they use for assigning the fitness value.

In Chapter 5 we first describe general parallel models of MOGAs, and then mention approaches which use and modify these basic models in order to provide better performace.

Chapters 6, 7, and 8 contain our original results. In Chapter 6, we study the effect of artificial constraints, added to island based MOGAs, on their performance. We use fuzzy constraint to perform this study. In Chapter 7 we try to combine different MOGAs in a heterogeneous island model to combine their advantages. Combination of a MOGA with a single-objective evolutionary algorithm is tested in Chapter 8.

Finally, Chapter 9 concludes the results of the previous chapters and provides some suggestions for future research.

## 1.1 Applications and Case Studies

In this section, we provide some examples of applications of multiobjective optimization (MOO) in various fields of study.

(Jakobsson *et al.*, 2006) describe the use of multiobjective optimization in the field of computational electromagnetics. Here, the return loss of a patch antenna is optimized on two different frequencies. Several multiobjective optimization methods are compared.

(Yu *et al.*, 2000) use multiobjective optimization in radiotherapy. The hot spot in the target volume as well as the median dose should be minimized and an equitable fall-off of the dose in all directions should be provided. The authors show, that multiobjective optimization yields better results than human experts and provides novel treatment strategies.

Multiobjective optimization is used in hybrid electric vehicle design (Cook *et al.*, 2006). The objectives here express the fuel consumption, the electricity consumption and the responses to acceleration demands.

Multiobjective optimization can also be used for data clustering (Saha & Bandyopadhyay, 2008). The authors argue that most data clustering techniques optimize only one measure of quality of the clusters, and they show that multiobjective optimization can be successfully used to optimize more quality measures at once.

Neural networks for time series prediction are another field where multiobjective optimization can be used (Chiam *et al.*, 2006). The conflicting objectives here are the errors on the training and validation sets. Another objective function can express the size of the network.

Another use of multiobjective optimization is in the area of resource allocation problems. (Ripon *et al.*, 2007) use evolutionary multiobjective algorithm to solve the job-shop scheduling problem with two objectives (namely, makespan and tardiness) and (Datta *et al.*, 2006) use MOO for class timetabling and land-use management.

Multiobjective optimization is also used in bioinformatics: it was used in the process of phylogeny inference (Cancino & Delbem, 2006), as more criteria can be employed in order to guide the search algorithm, which can lead to different phylogenies. (Gronwald *et al.*, 2008) used multiobjective optimization in the study of stably folding peptides.

## 1.2 Problem Definition

In this section, we define the multiobjective optimization problem and basic relations used to compare the quality of possible solutions. The terms defined in this section are used throughout the rest of the work.

**Definition 1.1.** The *multiobjective optimization problem (MOP)* is a quadruple $\langle D, O, \boldsymbol{f}, C \rangle$, where

- $D$ is the *decision space*
- $O \subseteq \mathbb{R}^n$ is the *objective space*
- $C = \{g_1, \ldots, g_m\}$, where $g_i : D \to \mathbb{R}$ is the set of *constraint functions (constraints)* defining the *feasible space* $\varPhi = \{\boldsymbol{x} \in D | g_i(\boldsymbol{x}) \leq 0\}$
- $\boldsymbol{f} : \varPhi \to O$ is the vector of $n$ *objective functions (objectives)*, $\boldsymbol{f} = (f_1, \ldots, f_n), f_i : \varPhi \to \mathbb{R}$

$\boldsymbol{x} \in D$ is called the *decision vector* and $\boldsymbol{y} \in O$ is denoted as the *objective vector.*

We consider only minimization problems here. Maximization and mixed problems can be easily converted into minimization problems (e.g. by multiplying those functions which shall be maximized by $-1$).

The goal of the single-objective optimization is to minimize the objective function with respect to the constraints, i.e. find a decision vector $\boldsymbol{x} \in \varPhi$ such that the objective function is minimized. However, in multiobjective optimization there is usually not a single decision vector optimizing all the objective functions, and therefore trade-off decision vectors need to be found. The following definitions formalize this.

**Definition 1.2.** Given decision vectors $\boldsymbol{x}$, $\boldsymbol{y} \in D$ we say

- $\mathbf{x}$ *weakly dominates* $\mathbf{y}$ ($\boldsymbol{x} \preceq \boldsymbol{y}$) if $\forall i \in \{1 \dots n\} : f_i(\boldsymbol{x}) \leq f_i(\boldsymbol{y})$.
- $\mathbf{x}$ *dominates* $\mathbf{y}$ ($\boldsymbol{x} \prec \boldsymbol{y}$) if $\boldsymbol{x} \preceq \boldsymbol{y}$ and $\exists j \in \{1 \dots n\} : f_j(\boldsymbol{x}) < f_j(\boldsymbol{y})$
- $\mathbf{x}$ *and* $\mathbf{y}$ *are incomparable* if neither $\boldsymbol{x} \preceq \boldsymbol{y}$ nor $\boldsymbol{y} \preceq \boldsymbol{x}$.
- $\mathbf{x}$ *does not dominate* $\mathbf{y}$ ($\boldsymbol{x} \not\preceq \boldsymbol{y}$) if $\boldsymbol{y} \preceq \boldsymbol{x}$ or $\boldsymbol{x}$ and $\boldsymbol{y}$ are incomparable

It is easy to show that both $\prec$ and $\preceq$ define a partial order on the decision space.

**Definition 1.3.** The *solution of a MOP* is the *Pareto (optimal) set*

$$P^* = \{\boldsymbol{x} \in \varPhi | \forall \boldsymbol{y} \in \varPhi : \boldsymbol{y} \not\preceq \boldsymbol{x}\}$$

The projection of $P^*$ under $\boldsymbol{f}$ is called the *Pareto optimal front.*

**Definition 1.4.** Every $P \subseteq \varPhi$ is called *Pareto set approximation.*

Using these definitions we can finally state the goal of multiobjective optimization. It is to find the Pareto optimal set of feasible decision vectors. If this set is infinite or large (which is often the case) we seek for its approximation.

The quality of these approximations can be evaluated using the measures described in Chapter 3. Here we only extend the notion of dominance to sets of solutions and state that it is the "ultimate measure of quality", i.e. whenever a set $A$ (weakly) dominates a set $B$, the set $A$ is a better Pareto set approximation.

We first define the term of nondominated subset of a Pareto set approximation, these are those vectors, which are not dominated by any other vector in this set and are indifferent to each other. This set is used in some of the algorithms described in this work.

**Definition 1.5.** For a Pareto set approximation $P$, $\mathrm{ND}(P)$ is the *set of nondominated vectors in $P$.*

$$\mathrm{ND}(P) = \{\boldsymbol{x} \in P | \forall \boldsymbol{y} \in P : \boldsymbol{y} \not\prec \boldsymbol{x}\}$$

**Definition 1.6.** Given two Pareto set approximations $A$ and $B$ we say

- *$A$ weakly dominates $B$* ($A \preceq B$) if $\forall \boldsymbol{x} \in B \exists \boldsymbol{y} \in A : \boldsymbol{y} \preceq \boldsymbol{x}$
- *$A$ dominates $B$* ($A \prec B$) if $A \preceq B$ and $\exists \boldsymbol{y} \in A, \boldsymbol{x} \in \mathrm{ND}(B) : \boldsymbol{y} \prec \boldsymbol{x}$
- *$A$ and $B$ are incomparable* if neither $A \preceq B$ nor $B \preceq A$.

It is easy to show that the Pareto optimal set weakly dominates all Pareto set approximations.

Note that two sets are often incomparable with respect to the set dominance relation. This is one of the reasons why another quality indicators have to be used to compare the quality of Pareto set approximations. We discuss these indicators later.

# Chapter 2

# Methods Used in Multiobjective Optimization

This chapter briefly describes some methods used in multiobjective optimization. We do not mention multiobjective evolutionary algorithms here, as the rest of the work is dedicated to them. This chapter should mention the basic ideas behind the methods, some details may be left out.

## 2.1 Function Aggregation

Probably the most straightforward approach to solving multiobjective optimization problems is their reduction to single-objective optimization problems using function weighting, i.e. assigning a weight to each of the functions.

The single-objective problem is defined as minimization of

$$f = \sum_{i=1}^{n} w_i f_i$$

where

$$\sum_{i=1}^{n} w_i = 1$$

and $w_i$ are nonnegative real numbers.

The main drawback of this approach is the need to specify the weights, which have no clear connection to decision maker's preferences.

Moreover, obtaining an even spread of solutions is a challenging task, as an even spread of weights does not guarantee it.

To overcome the difficulty with assigning weights, the goal programming approach can be used. This approach uses another method to create a single-objective optimization problem from a given MOP.

The decision maker selects an *utopia vector* $\boldsymbol{g}$ in the objective space, which they would like to achieve. The single objective function is than defined as

$$f(\boldsymbol{x}) = \sum_{i=1}^{n} \mu(f_i(\boldsymbol{x}), \boldsymbol{g}_i)$$

where $\mu$ is a metric on the decision space.

This way the decision maker can easily incorporate their preferences into the multiobjective search.

## 2.2 Normal-Boundary Intersection

Normal-Boundary Intersection (Das & Dennis, 1998) converts the multiobjective problem into several single objective problems. First, the optima of individual objective functions $\boldsymbol{x}_i^*$ are found. Then, the problem is transformed into maximization of $t$ and the constraint $\Phi w + t\boldsymbol{n} = \boldsymbol{f}(\boldsymbol{x})$ is added. Here $\Phi$ is a matrix whose $i$-th column is $\boldsymbol{f}_i(\boldsymbol{x}_i^*) - \boldsymbol{f}^*$ ($\boldsymbol{f}^*$ is the vector of individual optima of the objective functions), $w$ is a weight vector and $\boldsymbol{n}$ is a vector normal to the hyperplane containing the individual optima (*CHIM – convex hull of individual minima*). This problem is solved for various values of $w$ to obtain an approximation of the Pareto optimal set.

(Messac *et al.*, 2003) proposed a similar method called Normalized Normal Constraint. In this method, the values of each objective function are normalized to the interval $[0, 1]$. Instead of maximizing $t$ the original functions are minimized in a reduced decision space (a plane normal to the CHIM divides the decision space and the solutions are searched in the part, which does not contain the minimum of the particular objective function). The authors argue that this method provides a more even spread of solutions on the Pareto front. Moreover, a Pareto filter was added to filter out dominated solutions from the resulting approximation.

## 2.3 Particle Swarm Optimization

Particle Swarm Optimization (Kennedy & Eberhart, 1995) is a population based technique, which was originally developed to deal with continuous single-objective optimization. The individuals (*particles*) in the population (*swarm*) follow the best individual (*leader*) in a parameter space in a similar way in which natural swarms moves.

In each iteration, the position of each individual is changed by adding a *speed* to its current position. The speed depends the best known location of the individual in the past. the location of currently best individual, and the location of best individual in the neighborhood. It also depends on the previous speed of the individual (inertia).

In the multiobjective context choosing the leader is a nontrivial task, because there is no best individual. Various techniques were developed to deal with this problem, for example function aggregation or dominance sorting together with density estimation.

For a more complete survey on multiobjective particle swarm optimizers see (Reyes-Sierra & Coello, 2006).

## 2.4 Ant Colony Optimization

Ant Colony Optimization (Dorigo, 1992) was originally developed for finding optimal paths in graphs. It uses a population of ants which move through the graph and leave pheromone trails. The amount of pheromone left behind usually depends on the quality of the particular path in the graph the ant was following, and is often computed after the whole path is found. When an ant has to choose from more edges, it chooses with the probability proportional to the amount of pheromone on these edges. As the other ants follow these pheromone paths, good paths has more pheromone on them and more ants are following them.

In order to adapt this approach to multiobjective optimization we must define how much pheromone to lay. The amount usually depends on an aggregation function. Another possibility is to have more colonies, each associated with different objective function and with its own pheromone. Moreover, only nondominated solutions are usually rewarded in each iteration.

An example of a recent multiobjective ant colony optimizer can be found in (Alaya *et al.*, 2007).

# Chapter 3

# Quality Indicators

In single-objective optimization it is quite easy to compare two solutions, comparing the fitness values is usually enough. On the other hand, in multiobjective optimization there is not a simple way of comparing Pareto set approximations as the sets are usually incomparable with respect to the dominance relation. Therefore, various quality indicators has been created to help comparing the Pareto set approximations (and multiobjective optimizers).

Formally, an indicator is defined as follows:

**Definition 3.1.** *(Quality) indicator $I$* is a function that maps $k$ Pareto set approximations to a real number.

Tho most common indicators are unary and binary ones and most of binary indicators can be transformed to their unary version by using a reference set.

In the following sections, some examples of the most often used indicators are presented. Another quality indicators can be found in (Zitzler *et al.*, 2008a).

## 3.1 $I_{\epsilon+}$ − Additive $\epsilon$ Indicator

**Definition 3.2.** Let $A, B$ be two Pareto set approximations,

$$I_{\epsilon+}(A, B) = \min_{\epsilon}\{\forall \boldsymbol{x} \in B \exists \boldsymbol{y} \in A : f_i(\boldsymbol{y}) - \epsilon \leq f_i(\boldsymbol{x})\}$$

This indicator shows, how much a Pareto set approximation has to be moved in the objective space in order to (weakly) dominate another approximation.

This indicator is used in some indicator based evolutionary algorithms because it can be computed very fast (in polynomial time).

There is also a multiplicative version $I_\epsilon$ of this indicator defined in a similar way. Both these versions were proposed in (Zitzler *et al.*, 2003).

## 3.2 $\mathcal{S}$ – Hypervolume Metric ($\mathcal{S}$ metric)

**Definition 3.3.** Let $R \subset O$ be a reference set. The *hypervolume metric* $\mathcal{S}$ is defined as

$$\mathcal{S}(A) = \lambda(H(A, R))$$

where

- $H(A, R) = \{x \in O | \exists \boldsymbol{a} \in A \exists \boldsymbol{r} \in R : \forall i \in \{1, \dots, n\} : f_i(\boldsymbol{a}) \preceq \boldsymbol{x}_i \preceq \boldsymbol{r}_i\}$ where $f_i$ is the $i$-th objective function
- $\lambda$ is Lebesgue measure with $\lambda(H(A, R)) = \int_O \boldsymbol{1}_{H(A,R)}(z)dz$ and $\boldsymbol{1}_{H(A,R)}$ is the characteristic function of the set $H(A, R)$

The reference set often contains only a single reference point. This indicator calculates the hypervolume of the space dominated by the Pareto front approximation. Sometimes the value of this metric is normalized in such a way, that the Pareto optimal front $P^*$ has the value $\mathcal{S}(P^*) = 1$. Note that the optimal value can not be achieved with finite Pareto set approximation.

$\mathcal{S}$ metric proposed by (Zitzler & Thiele, 1998) is considered to be the most important quality indicator in the field of MOGA. Modern multiobjective algorithms use it as a part of the fitness assignment. Therefore, it has been intensively studied, and important features has been proved about the hypervolume indicator.

It is the only indicator which is strictly monotonic with respect to Pareto dominance relation (Fleischer, 2003). This means that maximizing this indicator is equivalent to finding the Pareto optimal set.

(Auger *et al.*, 2009) showed, that this indicator provides good spread of solutions over the Pareto front and moreover, it is biased towards the parts of the Pareto optimal front, where the slope is around 45 degrees. These are the most interesting solutions for the decision maker, as they provide fair trade off between the objectives.

On the other hand, the calculation of the hypervolume indicator is very time-consuming and it was proved by (Bringmann & Friedrich, 2008), that the problem of computing the hypervolume is $\#P$-complete, i.e. no polynomial algorithm exists unless $P = NP$. In fact, the fastest algorithms for computing the $\mathcal{S}$ metric work in time $\mathcal{O}(N^{n/2})$ (Beume & Rudolph, 2006) or $\mathcal{O}((n/2)^N)$ (While *et al.*, 2006). Where $N$ is the size of the set and $n$ is the number of objective functions.

## 3.3 $\mathcal{C}$ – Coverage of Two Set

$\mathcal{C}$ metric, or coverage of two set, is a generalization of the dominance relation. It maps an ordered pair of Pareto set approximations to the interval $[0, 1]$.

**Definition 3.4.** Let $X, Y$ be two Pareto set approximations. The function $\mathcal{C}(X, Y)$ is defined as follows.

$$\mathcal{C}(X,Y) = \frac{|\{\boldsymbol{y} \in Y | \exists \boldsymbol{x} \in X : x \preceq y\}|}{|Y|}$$

This metric expresses how many solutions in one set is dominated by the solutions in the other set. Note that both $\mathcal{C}(X, Y)$ and $\mathcal{C}(Y, X)$ have to be considered, since they are not necessarily complements.

## 3.4 Convergence Metric

**Definition 3.5.** Let $P = \{\boldsymbol{p}_1, \ldots, \boldsymbol{p}_H\}$ is the set of $H$ uniformly distributed vectors from the Pareto front and $A = \{\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N\}$ is a Pareto set approximation. The *convergence metric* is defined as

$$\frac{1}{|A|} \sum_{i=1}^{|A|} min_{j=1,\ldots,H} \delta(\boldsymbol{f}(\boldsymbol{a}_i), \boldsymbol{p}_j)$$

where $\boldsymbol{f}$ is the vector objective function and $\delta$ is the Euclidean distance.

This metric was introduced in (Deb *et al.*, 2003) and expresses the average distance of the individuals in the Pareto set approximation to the Pareto optimal front. If it is possible, the set $P$ can be replaced with the actual Pareto optimal front and the average distance is computed exactly.

The main disadvantage of this metric is the need to know the Pareto optimal front in advance.

# Chapter 4

# Multiobjective Evolutionary Algorithms

Evolutionary algorithms are a population based search technique inspired by Darwinian evolution. The individuals in the population create offspring by mating. Sometimes, mutation occurs. The probability, that an individual will be selected for the mating depends on his *fitness* (i.e. a function, which expresses the quality of the individual).

**Algorithm 1** Evolutionary algorithm

```
Create random initial population
while termination criterion not met do
   Evaluate the individuals in the population
   Select individuals for mating according to their fitness
   Perform crossover
   Perform mutation
   Perform environmental selection
end while
```

An evolutionary algorithm usually runs for a prespecified number of generations, however a different stopping criterion can be chosen.

The *mating selection* selects the individuals for crossover and mutation. Most of the evolutionary algorithms nowadays use so called *tournament selection*. It randomly chooses two individuals from the population and the one with better fitness is selected.

The selection and mutation operators are problem specific and also depend on the chosen encoding of the individuals. The *crossover operator* combines two individuals to create a new one (e.g. by swapping some bits between them), the *mutation operator* changes randomly a singe individual (e.g. flips randomly some of its bits). Both of these operators have a probability with which they are used. Usually, the probability of crossover is quite large (0.8), whereas the mutation probability is low (0.01).

The *environmental selection* chooses which individuals should remain in the population to the next generation. The simplest evolutionary algorithms simply use the offspring population. However, more advanced algorithms use an environmental selection which includes some kind of *elitism* (i.e. the best individuals always remain in the population).

The main difference between single-objective (SOGA) and multiobjective (MOGA) evolutionary algorithms is in the fitness assignment technique. In multiobjective evolutionary algorithms, the fitness should drive the algorithm towards the Pareto optimal front, while keeping good spread of solutions.

MOGAs can be divided into three groups according to the type of information they use in the fitness assignment:

**Aggregation based EAs** transform the MOP into single-objective optimization problem e.g. by function weighting, goal programming etc.

**Domination based EAs** use the information provided in the dominance relation together with a niching method to keep diversity in the population

**Indicator based EAs** use quality indicators (e.g. hypervolume indicator) to compare individuals or populations

In the following sections, we discuss the most important algorithms from each of these groups. The last section of this chapter briefly compares the algorithms.

## 4.1 Aggregation Based Algorithms

Aggregation based algorithms use some kind of aggregating function to assign the fitness value, or to perform environmental selection.

### 4.1.1 VEGA – Vector Evaluated Genetic Algorithm

VEGA (Schaffer, 1985) is one of the first multiobjective evolutionary algorithms and was often used as a reference during 1990s.

The main idea is to select different parts of the population according to different objective functions.

**Environmental Selection:** Population is sorted according to each of $n$ objective functions and $N/n$ best solutions with respect to each objective are selected to the next generation.

**Algorithm 2** Vector Evaluated Genetic Algorithm

```
while termination criterion not met do
  for i = 1 ... n do
    select $N/n$ best individuals according to $f_i$ to the mating pool
  end for
  shuffle the mating pool
  perform crossover and mutation
  perform selection
end while
```

**Drawbacks and criticism:** This algorithm has difficulties with maintaining the diversity in the population and the Pareto set approximation is often biased towards some parts of the decision space. After a large number of generations the individuals tend to converge to the optima of individual objective functions. Schaffer was aware of this problem and proposed some methods to avoid it, but none of them really worked.

### 4.1.2 MSOPS – Multiple Single Objective Pareto Sampling

MSOPS (Hughes, 2005) uses the aggregation (any type of aggregation can be used) during fitness assignment. It performs multiple single-objective optimizations in one run.

This algorithm is the only one we know about, that uses the genetic operators from differential evolution (Storn & Price, 1995) to create new population. This operators self adapt to the fitness landscape, reducing the size of mutations as the search converges.

**Fitness assignment:** The individuals are evaluated over all *target vectors* (i.e. the parameters of the aggregation technique). Then, the individual which was the best in most cases is assigned the best fitness and so on.

**Crossover:** First, a new chromosome is generated by adding weighted difference to the parent chromosome.

$$P_t = F(P_a - P_b) + P_c$$

where $P_a$, $P_b$, and $P_c$ are chosen from the population randomly without replacement. The new chromosome is then crossed with the parent. The position to cross $L$ is chosen randomly with the probability

$$P(L \geq v) = C^{v-1}$$

The parameters $F$ and $C$ should lie in the interval $[0.5, 1]$. Value of 0.7 was used for both parameters in the original paper.

**Algorithm 3** Multiple Single Objective Pareto Sampling

```
Create random initial population
while termination criterion not met do
  Evaluate the individuals for all target vectors and save the results in
  score matrix $S$ (columns corespond to target vectors, rows to individu-
  als)
  Replace the values in each column with their rank (best value is replaced
  by 1 and so on), thus creating the rank matrix $R$
  Sort each row of the matrix $R$ in ascending order
  Assign the fitness – the individual which is most often ranked 1 has the
  best fitness and so on
  Generate new generation using differential evolution technique
end while
```

## 4.2 Domination Based Algorithms

This group of algorithms uses the dominance relation in the fitness assignment process, thus following suggestions presented in (Goldberg, 1989). As the dominance relation itself does not preserve the diversity in the population, another techniques, such as niching, are needed to obtain a good spread of solutions.

These algorithms have been very popular since mid 1990s and some of the state-of-the-art algorithms belong to this group.

In the following parts, the most important algorithms of this group are discussed. Another domination based algorithms were proposed for example by (Fonseca & Fleming, 1993) and (Horn *et al.*, 1994).

### 4.2.1 NSGA – Nondominated Sorting Genetic Algorithm

NSGA (Srinivas & Deb, 1994) uses the dominance relation to divide the population into *nondominated fronts*, which are sets of incomparable individuals.

**Fitness assignment:** The population $P$ is divided into nondominated fronts $F_1, F_2, \ldots$ where

$$F_i = \begin{cases} \mathrm{ND}(P) & \text{if } i = 1 \\ \mathrm{ND}(P \setminus \bigcup_{k=1}^{i-1} F_i) & \text{otherwise} \end{cases}$$

For each individual $i$ the *niche count* is calculated as the sum of

$$Sh(d_{ij}) = \begin{cases} 1 - (\frac{d_{ij}}{\delta_{share}})^2 & \text{if } d_{ij} < \delta_{share} \\ 0 & \text{otherwise} \end{cases}$$

over all individuals $j$ in the same nondominated front. Here, $d_{ij}$ is the distance of the individuals $i$ and $j$ in the objective space and $\delta_{share}$ is a niching parameter which has to be set in advance.

The individuals in the first nondominated front are assigned dummy fitness which is afterwards divided by the niche count. Than individuals in the second nondominated front are assigned dummy fitness smaller than the fitness of the worst individual in the first front, their dummy fitness is again divided by the niche count and so on for all the nondominated fronts.

**Criticism:** NSGA has been mainly criticized for

- The need to specify $\delta_{share}$
- The lack of elitism

**Algorithm 4** Nondominated Sorting Genetic Algorithm

```
while termination criterion not met do
  front = 1
  while population not classified do
    identify nondominated individuals
    assign dummy fitness
    sharing in current front
    front = front + 1
  end while
  perform reproduction according to dummy fitness
end while
```

### 4.2.2 NSGA-II

NSGA-II (Deb *et al.*, 2000) is the successor of NSGA and solves both of its main problems. Moreover, it implements faster nondominated sorting procedure.

**Niching:** The $\delta_{share}$ parameter and the niche count were replaced by the *crowding distance*, which is the sum of distances to the nearest neighbors. Best solutions in each objective function has the crowding distance set to infinity.

**Elitism:** The elitism was implemented using different selection scheme: preceding the selection, the parent and children populations are merged, fitness is calculated for all the individuals in the merged population and the best individuals are selected to the next generation.

**Fitness assignment:** Each individual is assigned the number of nondominated front it belongs to and the crowding distance. When comparing two individuals $i$ and $j$ the one with lower front number is better. If both $i$ and $j$ belong to the same front the one with larger crowding distance is better. Note that no fitness is computed in NSGA-II as binary tournaments are used and this condition is checked directly.

**Algorithm 5** Nondominated Sorting Genetic Algorithm II

```
while termination criterion not met do
  $R_t = Q_t \cup P_t$ {combined populations}
  $F = (F_1, F_2, \dots)$ all nondominated fronts from $R_t$
  $i = 1$
  while $|P_{t+1}| + |F_i| \leq N$ do
    compute the crowding distance for individuals in $F$
    $P_{t+1} = P_{t+1} \cup F_i$
    $i = i + 1$
  end while
  sort $F_i$ according to crowding distance
  $P_{t+1} = P_{t+1} \cup F_i[1 : (N - |P_{t+1}|)]$
  $Q_{t+1}$ = perform selection, crossover and mutation on $P_{t+1}$
  $t = t + 1$
end while
```

### 4.2.3 SPEA – Strength Pareto EA

SPEA (Zitzler & Thiele, 1999) was one of the first multiobjective algorithms which implemented elitism. In the time of its creation it outperformed almost all existing multiobjective evolutionary algorithms.

**Elitism:** SPEA uses external archive of nondominated solutions. The number of solutions in the archive is limited and if there are more nondominated solutions than the limit, the archive is truncated. The truncation is based on clustering technique which removes individuals from the most crowded regions.

**Archive truncation:** First create a cluster for each individual. Until the number of clusters is larger than the archive size, merge two nearest clusters (Euclidean distance in objective space is used) into one larger cluster. Finally choose centroids of the clusters as the truncated archive.

**Fitness assignment:** To compute the fitness, all individuals in the archive are assigned strength, which is the number of individuals in the population, they dominate. The fitness of the individuals in the archive is their strength.

Other individuals are assigned fitness which is the sum of strengths of the individuals in the archive they dominate plus one.

Note that the fitness is minimized in SPEA.

**Algorithm 6** Strength Pareto Evolutionary Algorithm

```
Randomly initialize population $P_0$ and create empty archive $\bar{P}_0$
while termination condition not met do
    Set temporal external set $\bar{P}' = \bar{P}_t$
    Copy nondominated individuals from $P_t$ to $\bar{P}'$
    Remove dominated individuals from $\bar{P}'$
    Reduce $\bar{P}'$ using clustering
    $\bar{P}_{t+1}$ is the reduced set from the previous step.
    Calculate fitness as described in the previous paragraph
    Create $P'$ using binary tournament on $P_t \cup \bar{P}_t$
    Create $P_{t+1}$ by crossover and mutation on $P'$
end while
```

**Drawbacks:** Among the problems of SPEA belong:

- The truncation method sometimes removes best solutions with respect to some objective function.
- Sometimes (e.g. if there is only one individual in the archive) all the individuals in the population have the same fitness, thus degrading the algorithm to random search.

### 4.2.4 SPEA2

SPEA2 (Zitzler *et al.*, 2001) is an improved version of SPEA which solves its problems.

**Fitness assignment:** When computing the fitness function of individual $i$, first, the raw fitness $R(i)$ is calculated. It is the number of other individuals it dominates. Next the distance to $k$-th nearest neighbor $\delta_i$ is calculated and density is computed as

$$D(i) = \frac{1}{\delta_i + 2}$$

The sum $R(i) + D(i)$ is the fitness of individual $i$.

**Archive truncation:** The truncation procedure in SPEA2 removes those individuals which has the minimum distance to another individual, ties are broken considering the second smallest distance and so on. The optimal individuals in each of the objective function are always kept in the archive.

Moreover, if the number of individuals in the archive is smaller than archive size more individuals are added according to their fitness. This means that unlike in SPEA the number of individuals in the archive does not change.

The transcript of the algorithm is not provided here, as it is basically the same as SPEA.

### 4.2.5 SPEA2+

**Algorithm 7** Strength Pareto Evolutionary Algorithm 2+

```
Randomly initialize population $P_0$ and create empty archives $A_0^O$ and $A_0^V$, set $t = 0$
while termination condition not met do
  Fitness values of individuals in $P_t$, $A_t^O$ and $A_t^V$ are calculated using SPEA2 fitness assignment.
  All nondominated individuals in $P_t$, $A_t^O$ and $A_t^V$ are copied to $A_{t+1}^O$ and $A_{t+1}^V$
  if archive size is exceeded then
    Perform truncation in the objective space on $A_{t+1}^O$ and truncation in the decision space on $A_{t+1}^V$
  end if
  if archive size is smaller than limit then
    Fill archives with dominated individuals with best fitness
  end if
  Create $P_{t+1}$ by copying $A_{t+1}^O$
  Perform neighborhood crossover and mutation on $P_{t+1}$
  $t = t + 1$
end while
```

(Kim *et al.*, 2004) proposed an improved version of SPEA2. They argued that most MOGAs use sharing in the objective space to obtain a good spread of solutions, but only few of them are concerned with the diversity in the decision space. However, if there are (for example) two different decision vectors with the same objective values, both of them could be interesting for the decision maker.

To maintain the diversity in the decision space, the authors added another archive in which the truncation is based on the decision values instead of the objective values (otherwise the method is the same).

Moreover, the crossover operator used in SPEA2 was replaced by *neighborhood crossover* as proposed in (Watanabe *et al.*, 2002). (In fact, neighborhood crossover is a kind of combination of selection and crossover operators.)

**Neighborhood crossover:** An objective function is chosen in each generation and the individuals are sorted according to its objective values. Next, *neighborhood shuffle* is performed on the sorted population and $i$-th and $(i+1)$th individuals are selected and the original crossover is performed on them.

The neighborhood shuffle shuffles the individuals in such a way, that they stay close (less than 10% of the population size) to their original position in the sorted population.

## 4.3 Indicator Based EAs

Indicator based evolutionary algorithms present new approach in evolutionary multiobjective optimization.

These algorithms use indicators (i.e. the hypervolume) instead of (or together with) dominance ranking during fitness assignment. The value of the indicator is optimized directly and niching is not needed as indicators itself help to provide diversity. Moreover, the indicators can express the decision maker's preferences.

**Definition 4.1.** A binary indicator $I$ is denoted as *dominance preserving* if for all decision vectors $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$

- $\boldsymbol{x} \prec \boldsymbol{y} \Rightarrow I(\{\boldsymbol{x}\}, \{\boldsymbol{y}\}) < I(\{\boldsymbol{y}\}, \{\boldsymbol{x}\})$ and
- $\boldsymbol{x} \prec \boldsymbol{y} \Rightarrow I(\{\boldsymbol{z}\}, \{\boldsymbol{x}\}) \geq I(\{\boldsymbol{z}\}, \{\boldsymbol{y}\})$

In (Zitzler & Künzli, 2004) it is shown that dominance preserving indicators guide the search towards the Pareto optimal set, i.e. that minimizing these indicators is equivalent to finding the Pareto set.

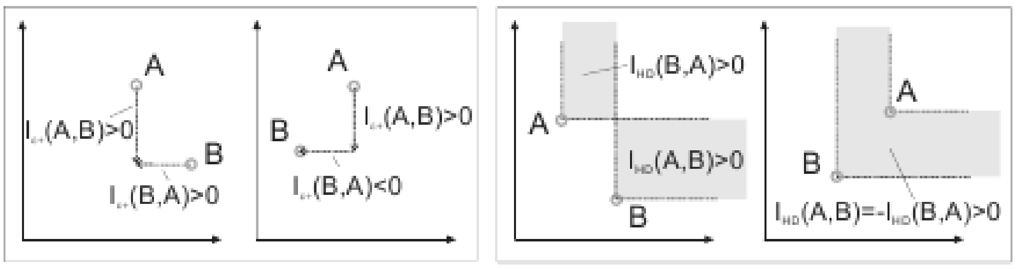

Figure 4.1: $I_\epsilon$-indicator (left) and $I_{HD}$-indicator (right)

Most commonly used indicators include the previously defined $I_{\epsilon+}$ indicator and a hypervolume based indicator $I_{HD}$, that can be defined as follows. Both of these indicators are dominance preserving.

**Definition 4.2.** For two Pareto set approximations $A, B$ the *hypervolume based indicator*

$$I_{HD}(A,B) = \begin{cases} \mathcal{S}(B) - \mathcal{S}(A) & \text{if } A \preceq B \\ \mathcal{S}(A+B) - \mathcal{S}(A) & \text{otherwise} \end{cases}$$

Here + is the multiset union.

The most successful algorithms in this group use some kind of hypervolume indicator. However, computing the value of this indicator is very inefficient and therefore, various ways of speeding up these algorithms are used. For example, (Brockhoff & Zitzler, 2007) proposed a method for reducing the number of objectives and (Bader & Zitzler, 2008) used Monte Carlo method for estimating the hypervolume indicator.

### 4.3.1 IBEA – Indicator Based EA

(Zitzler & Künzli, 2004) proposed a general indicator based evolutionary algorithm.

**Fitness assignment:** The indicator value of each pair of individuals is computed, and an individual $i$ is assigned fitness

$$F(i) = \sum_{j \in P \setminus \{i\}} -e^{-I(\{j\},\{i\})/\kappa}$$

**Algorithm 8** Indicator Based Evolutionary Algorithm

```
Create random initial population $P$
while termination criterion not met do
    Scale objective values to the interval $[0, 1]$ and the indicator values to $[-1, 1]$
    Calculate fitness using the scaled values
    while population size exceeds limit do
        Select the individual $\boldsymbol{x}^*$ with worst fitness and remove it
        Update fitness of other individuals
    end while
    Perform binary tournaments to fill the mating pool $P'$
    Perform crossover and mutation on $P'$ and add $P'$ to $P$
end while
```

Here, $\kappa$ is a scaling factor which has to be set in advance. The purpose of the exponential is to amplify the differences between dominated and non-dominated individuals.

**Environmental selection:** Iteratively the worst individuals are removed from the population and the fitness of the other individuals is updated (it is, for individual $i$

$$F(i) = F(i) + e^{-I(\{i^*\},\{i\})/\kappa}$$

where $i^*$ is the individual which was removed in the iteration).

**Scaling of the values:** The authors experimentally observed that the algorithm is sometimes sensitive to the value of $\kappa$ and to the chosen reference point for the hypervolume based indicator $I_{HD}$. Therefore, they suggested scaling the objective values to the interval $[0, 1]$ and choosing 2 for all the objective functions as the reference point for $I_{HD}$. They suggest using the values of indicators scaled to $[-1, 1]$ and the value of $\kappa = 0.1$.

### 4.3.2 SMS-EMOA – $\mathcal{S}$ Metric Selection EMOA

SMS-EMOA (Beume *et al.*, 2007) is one of the evolutionary algorithms based on the hypervolume metric. It is a steady-state algorithm (i.e. only one individual is changed in each generation) which combines nondominated sorting with the $\mathcal{S}$ metric in its selection procedure.

**Algorithm 9** $\mathcal{S}$ Metric Selection EMOA

```
Create random population $P$
while termination criterion not met do
    Create new individual $\boldsymbol{x}$ using individuals in $P$
    Add $\boldsymbol{x}$ into $P$
    Divide the population into nondominated fronts $F_1, \ldots, F_k$
    if $k > 1$ then
        Remove $\boldsymbol{y} = \text{argmax}_{\boldsymbol{y}} |\{\boldsymbol{x} \in P | x \prec y\}|$
    else
        Remove $\boldsymbol{y} = \text{argmin}_{y} (\mathcal{S}(P) - \mathcal{S}(P \setminus \{\boldsymbol{y}\}))$ from population
    end if
end while
```

**Environmental selection:** First the individuals are divided into non-dominated fronts. If there is more than one front, the individual in the last front, which is dominated by most individuals is removed. Otherwise the one which adds the least to the overall hypervolume is removed.

In this way, the $\mathcal{S}$ metric is calculated only $N$ times per generation. If more individuals ($m$) were created (and removed) in each iteration the number of evaluations would grow rapidly as every subset of size $m$ would have to be considered. This accounts for $\binom{N+m}{m}$ evaluations per generation.

### 4.3.3 SPAM – Set Preference Algorithm for MOO

SPAM (Zitzler *et al.*, 2008b) uses indicators (but not necessarily) to construct the set preference relation $\preccurlyeq$, which is than used to compare Pareto set approximations. For example a set preference relation based on an unary indicator $I$ is defined as

$$A \preccurlyeq_I B \Leftrightarrow I(A) < I(B)$$

This algorithm uses two operators. One performs random mutation of the set. The other performs heuristic mutation and works as mating selection, crossover and mutation. If the heuristically generated set is better than the original one, it is replaced, otherwise the randomly generated one is compared to the original one.

The authors also show other set preference relations (even some, which are constructed from more basic relations) and their effects on the resulting Pareto set approximation. These relations can express decision maker's

**Algorithm 10** Set Preference Algorithm

```
Create random initial set $P$
while termination criterion not met do
  Create set $P'$ from $P$ by random mutation
  Create set $P''$ from $P$ by heuristic mutation
  if $P'' \preccurlyeq P$ then
    $P \leftarrow P''$
  else
    if $P' \preccurlyeq P$ then
      $P \leftarrow P'$
    end if
  end if
end while
```

preferences and effectively guide the search towards interesting regions of the decision space.

### 4.3.4 HypE – Hypervolume Estimation Based MOGA

In order to overcome the problem with the complexity of computing the hypervolume, (Bader & Zitzler, 2008) proposed Monte Carlo sampling method to estimate it. The hypervolume is computed exactly only for MOPs with up to three objective functions, otherwise it is estimated using Monte Carlo sampling.

**Fitness assignment:** The fitness of each individual is the hypervolume it dominates, but the parts of it which are dominated by more individuals are divided uniformly among them. Thus the sum of fitnesses of all individuals equals the $\mathcal{S}$ metric of the population.

Although this may seem difficult to compute, the authors proposed modified algorithm for computing hypervolume, which also computes this fitness. Also a Monte Carlo version of it exists.

**Environmental selection:** The parent and the children populations are merged and divided into nondominated fronts and the individuals from the best fronts are added to the new population. From the front, that does not fit completely to the population, the individuals are iteratively removed according to their fitness. After each iteration, the fitness is recalculated.

**Algorithm 11** Hypervolume Estimation Based MOGA

```
Create random initial population
while termination criterion not met do
  Assign the fitness
  $P'$ = Perform mating selection, crossover and mutation on $P$
  Divide $P \cup P'$ into nondominated fronts $F_1, \ldots, F_k$
  Add fronts $F_1, \ldots, F_j$, such that $| \cup_{i=1}^{j} F_i| < N$ to the new population $P''$
  while $|P''| + |F_{j+1}| > N$ do
    Compute fitness of individuals in $F_{j+1}$
    Remove the worst individual from $F_{j+1}$
  end while
  $P = P'' \cup F_{j+1}$
end while
```

The Monte Carlo sampling uses a fixed number of samples (authors suggest 10,000). They also tested adaptive Monte Carlo simulation, but the overhead for maintaining the necessary information and the additional computation outweights the advantages.

## 4.4 Performance Discussion

### 4.4.1 Aggregation vs. Domination Based EAs

VEGA is the worst of the presented algorithms, as it can not maintain the diversity in the population. Its main importance is, that it started the attempts to create better MOGAs.

The situation is quite different for MSOSP. In (Hughes, 2005) the author experimentally showed, that NSGA-II works better than MSOPS on biobjective problems. On the other hand, when the number of objectives is larger, MSOPS is able to outperform NSGA-II.

The reason is, that with the growing dimension of the MOP, the probability, that two individual are mutually incomparable, rises. When the dimension exceeds 10, almost all randomly generated objective vectors can not be compared using the dominance relation (Ishibuchi *et al.*, 2008). This implies, that the selection pressure of domination based evolutionary algorithms is very low, and its only source is the niching procedure. There is almost none selection pressure towards the Pareto optimal front.

### 4.4.2 Domination vs. Indicator Based EAs

The indicator based MOGAs perform usually better than the domination based ones. It is not a surprise, as the domination based algorithms do not optimize the $\mathcal{S}$-metric. Moreover these algorithms do not have the problems with low selection pressure, that was discussed in the previous section.

On the other hand, indicator based algorithms that use the hypervolume metric are usually very slow (Bader & Zitzler, 2008). IBEA (based on the $I_{\epsilon+}$ indicator) seems to be a good choice, if time matters. Although it does not provide as good approximations as hypervolume based algorithms, it is much faster. In fact it even runs faster than NSGA-II.

# Chapter 5

# Parallel Multiobjective Evolutionary Algorithms

As the evolutionary algorithms can be computationally expensive, parallel methods for their computation haves been studied and are often used in real-life problems solving.

In this chapter, we present the most often used methods for parallel evolutionary computing, and then describe some MOGAs, which use and modify these models.

Note that the presented models can be combined together to create hierarchical models (i.e. the island in the island model can use master-slave model to evaluate the fitness faster).

## 5.1 Models of Parallel Evolutionary Algorithms

Parallel evolutionary algorithms, can be divided into three groups, based on the way they use to parallelize the computation: the master-slave model uses multiple computers to compute the fitness, the island model divides the population into several subpopulations, and the cellular model uses massively parallel computing (i.e. each individual has its own processor).

Although these models have been created for parallel computation, sometimes they are used in single-processor environment. This particularly holds for the cellular model.

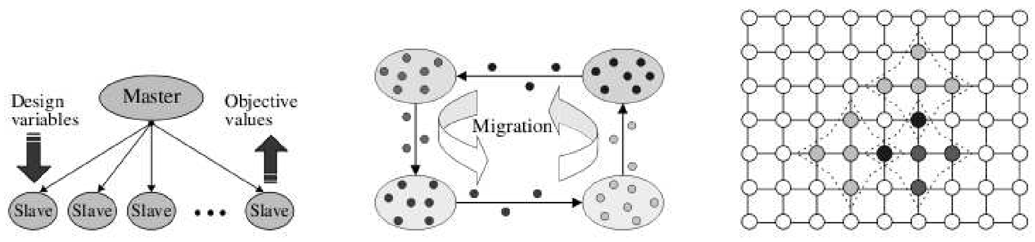

Figure 5.1: The three parallel models – master-slave model (left), island model, and cellular model (right)

### 5.1.1 Master-Slave Model

The master-slave model (also called *fine grained parallelization*) uses parallel computing of the fitness function. The evolution runs on a single processor, but the fitness evaluation is distributed.

This model is particularly useful when the fitness calculation is time-consuming. This is often the case in real-life problems, where the fitness requires solving of differential equations (e.g. when designing the shape of a wing), or where the fitness is assigned on the basis of a simulation (e.g. in evolutionary robotics).

The quality of the solutions found by this algorithms is the same as the one of those found by their single processor version. However, the runtime is considerably reduced.

### 5.1.2 Island Model

Island model, sometimes also called the *coarse-grain parallelization*, divides the population into smaller subpopulations, and each of them is optimized on a different processor, these subpopulations are called islands. Each island run a serial evolutionary algorithm. After a number of generations (called the *epoch*) some of the individuals are exchanged between the islands.

There are several variants of this model:

**Multi-start model** No communication is performed among the islands.

**Homogeneous island model** Individual island execute the same evolutionary algorithm with the same parameters.

**Heterogeneous island model** Each of the islands execute the algorithm with different parameters. Sometimes different encodings of the individuals or even different evolutionary algorithms are used on each island. Individual islands can also explore different parts of the search space.

The way how the islands communicate is defined by the topology of the connections between them. Most often choices include circle, toroid, square grid or complete topology.

Island based algorithms often provide better results than the original single processor algorithm with the same population size. They are also faster, as the subpopulations have less individuals. Because of these features, this is probably the most often used parallelization model.

### 5.1.3 Cellular Model

The cellular model was originally created for massively parallel systems. Each individual is assigned a cell (a processor). Mating is performed only between the individuals in the neighboring cells (neighborhood is defined by a topology).

The limited mating helps to keep the diversity in the population as any changes are propagated quite slowly.

This method can also be used instead of the master-slave model, when the fitness evaluations are costly.

## 5.2 DRMOGA – Divided Range MOGA

DRMOGA (Hiroyasu *et al.*, 1999) is one of the first multiobjective algorithms, that were designed to use parallel computing.

In DRMOGA the individuals are periodically gathered and divided among the islands according to the value of an objective function. Each time the dividing occurs the objective function used to divide the individuals is changed. The individual islands than run a serial multiobjective algorithm to solve the problem.

**Algorithm 12** Divided Range MOGA

```
Create random initial population
while termination criterion not met do
  Run MOGA in each group
  if end of epoch then
    Gather all individuals from the islands
    Choose an objective function $f_i$
    Sort all the individuals according to $f_i$
    Divide the individuals into groups of the same size
  end if
end while
```

## 5.3 DCMOGA – Distributed Cooperation Model of MOGA

(Okuda *et al.*, 2002) proposed parallel algorithm called DCMOGA. It uses $n+1$ ($n$ is the number of objective functions) groups to solve the multi-objective problem. $n$ of these groups are single-objective and they optimize individual objective functions. One of these groups is multiobjective and it performs multiobjective optimization. After all single-objective groups converge they are changed into multiobjective groups.

The idea here is, that the SOGA groups will find optima for the individual objective functions, and the MOGA group will approximate the rest of the front.

## 5.4 MOGADES – MOGA with Distributed Environment Scheme

MOGADES (Kamiura *et al.*, 2002) uses the weighted aggregation to assign the fitness. Each of the islands uses different weights which are (deterministically) changed during the evolution.

Every island has a Pareto archive to store the nondominated individuals and an elite archive to store the best individuals in its population.

**Pareto archive truncation:** If the maximum size of the Pareto archive is exceeded, the archive is truncated. The fitness of the individuals (the

**Algorithm 13** Distributed Cooperation Model of MOGA

```
Initialize all individuals
Divide the individuals into $n + 1$ groups
while termination condition not met do
    Run optimization in each group
    if epoch end then
        $M_i$ = the best solution in objective $f_i$ from the MOGA group
        $S_i$ = the best solution in SOGA group for the objective $f_i$
        if $M_i < S_i$ then
            Send some solutions from SOGA group of $f_i$ to MOGA group
        end if
        if $S_i < M_i$ then
            Send some solutions from MOGA group to SOGA group of $f_i$
        end if
        if $S_i^O == S_i \forall i \in \{1, \ldots, n\}$ then
            Convert all SOGA groups to MOGA groups
        end if
        $S_i^O = S_i$
    end if
end while
```

**Algorithm 14** MOGA with Distributed Environment Scheme

```
Generate random individuals
Divide the individuals into $M$ groups, $M$ is the number of islands
Set the weights $w_i = (i - 1)/(M - 1)$
while termination criterion not met do
    Perform crossover and mutation
    Normalize the value of the objective functions by the maximum of each
    function
    Calculate the fitness of the individuals
    Perform selection
    Copy nondominated individuals to Pareto archive and truncate the
    archive if its maximum size is exceeded
    Copy best individuals into elite archive
    Perform migration and weight adjustment
end while
```

weighted sum) is calculated and the archive is sorted according to it. The the difference in the fitness of neighboring individuals is computed and the individual with smallest distance is removed.

**Migration:** MOGADES uses a ring topology. During migration the weights on the islands are adjusted. The weight $w_i$ of island $i$ is computed as

$$w_i = w_{i+1}\frac{d_{i,i+1}}{d_{i-1,i} + d_{i,i+1}} + w_{i-1}\frac{d_{i-1,i}}{d_{i-1,i} + d_{i,i+1}}$$

where $d_{i,i+1}$ is the distance of the individuals with best values on islands $i$ and $i + 1$

The authors used MOGADES as the MOGA group in DCMOGA and called this algorithm DCMOGADES.

## 5.5 MRMOGA – Multiple Resolutions MOGA

MRMOGA (Jaimes & Coello, 2007) uses heterogeneous island model to create overlapping regions of the decision space. The islands use different resolutions (i.e. numbers of bits) to encode the solutions. The idea is, that the islands with lower resolution will converge faster to the Pareto optimal front, whereas the higher resolution islands will perform fine-tuning of the solutions found by the lower resolution ones.

To avoid wasting of computational resources after the lower resolution islands converged, the algorithm contains a technique to monitor the convergence. When an island converges, its resolution is increased.

Each island runs a serial evolutionary algorithm and contains an archive of best solutions.

**Migration:** At the end of each epoch, individuals from each island archive are selected and sent to the neighbors. The received individuals are placed into the population of the island. During the migration some of the individuals need to be recoded, as each of the islands may use different resolution.

**Archive truncation:** Individuals are divided into nondominated fronts and the worst individuals from the last front are removed. The authors do not explain the way in which these individuals are chosen.

**Algorithm 15** Multiple Resolutions Multi-Objective Genetic Algorithm

```
Create random population $P_i$ on island $i \in \{1, \ldots, I\}$
while termination criterion not met do
    perform one generation of serial MOGA
    if end of epoch then
        send some individual from archive to neighbors
        receive individuals from neighbors to population
    end if
end while
Combine all populations from all islands to the final population
```

## 5.6 Parallelization Using Clustering

(Streichert *et al.*, 2005) proposed using clustering for dividing the population into subpopulations in parallel evolutionary algorithms. They used k-means clustering to parallelize NSGA-II.

They applied constraints to the islands to keep the solutions in the ranges specified by the clusters and support the focus of each island to that particular part of the search space. Unexpectedly, this solution was not able to outperform the island version of NSGA-II without clustering.

However, when the constraints were removed, the algorithm performed much better and was able to outperform the version without clustering on some of the test problems.

**Algorithm 16** Parallelization Using Clustering

```
Initialize all islands
while termination criterion not met do
    Evolve all islands one generation
    if epoch ended then
        Gather the individuals from all islands
        Perform k-means clustering
        Send clusters back to islands
    end if
end while
```

The authors concluded that (Streichert *et al.*, 2005):

> “[...]the standard island MOEA with migration proves to be quite robust and hard to beat. We found that simple and con-

tiguous structure of the standard test functions allows lateral exploration once a good solution is found. This has been verified by removing the zone constraints[...]"

## 5.7 MOCell – Cellular MOGA

MOCell (Nebro *et al.*, 2008) is one of the few MOGAs using the cellular model, in this case with toroidal grid topology. It also uses a Pareto archive to store the best individuals found during the search. If the archive is larger than the limit, it is truncated in the same way as in NSGA-II (i.e. using crowding distance).

**Algorithm 17** Cellular Multiobjective Genetic Algorithm

```
while termination criterion not met do
  for each individual $i$ do
    Select parents from the neighborhood of $i$
    Create offspring by crossover and mutation
    Evaluate the offspring
    if Offspring not dominated by $i$ then
      Insert the offspring into auxiliary population and into Pareto archive
    end if
  end for
  Copy auxiliary population to the regular population
  Copy random individuals from Pareto archive to population
end while
```

# Chapter 6

# Using Fuzzy Constraints in Parallel MOGA

Some of the multiobjective genetic algorithms described in the previous section use constraints to divide the search space into several subspaces. We have already mentioned that (Streichert *et al.*, 2005) concluded that using no constraints can yield better results than using them. In this chapter we study the effect of the constraints on the performance of parallel MOGA.

In order to keep the things simple, we use only two islands and divide the search space into two parts by a constraint of the form $f_1 < c$ and $f_1 > c$, where $c = (\max f_1 + \min f_1)/2$.

(Streichert *et al.*, 2005) tested using constraints and not using constraints. We would like to try something between these two possibilities, therefore we add a fuzzy part to the expression. We change the constraint to

$$f_1(\boldsymbol{x}) - c < ldR(0,1)$$

(the other island uses the complementary constraint). Here, $l$ is what we call *constraint looseness*, $d$ is the difference between maximum a minimum value of the first objective function, and $R(0,1)$ is a random number between 0 and 1 (from uniform distribution). This way, the individuals which violate the constraint only a little have higher probability to stay in the population. If $l = 0$ the constraint is strict, equivalent to the one described in the previous paragraph. If $l = \infty$ the constraint is never violated and has no effect at all.

The individuals which violate the constraint are assigned fitness lower than all other individuals. Individuals with smaller violation get better fitness, than those with larger violation.

The constraints also influence the communication between the islands: if the constraints are to hard (looseness is close to 0) most of the individuals sent to the other island would not survive in the population, because they will violate its constraints. On the other hand, if the constraint is loose, most of the individuals will survive, thus making the communication more effective.

We call the multiobjective genetic algorithm with fuzzy constraints FC-MOGA.

## 6.1 Test Setup

We use the benchmark problems ZDT1 to ZDT4 and ZDT6 (two objectives) from the ZDT set (Zitzler *et al.*, 2000), and DTLZ1, DTLZ3 and DTLZ4 (three objective) from the DTLZ set (Deb *et al.*, 2005) to test this algorithm. We use 150 decision variables for ZDT1-3, 10 decision variables for ZDT4, and 30 decision variables for ZDT6. The DTLZ problems use 300 decision variables. See Appendix A for details.

For the constraint looseness we try the values 0, 0.1, 0.2, 0.3, 0.5, 1, 2, 5 and infinity. We use NSGA-II, $\epsilon$-IBEA and HD-IBEA with population size 100 run for 500 generations on both islands. The constraints are updated every 100 generations.

## 6.2 Test Results

We provide the boxplots created by 20 runs of FCMOGA with the described settings in Figures 6.2 and 6.2. Tables 6.1 to 6.3 provide p-values of one-sided T-test run on the results for some configurations. Note that some outliers may be removed to improve the readability. Larger boxplots and more statistical test results can be found on the attached CD.

The results show, that there is no constraint looseness setting, that would guarantee good performance for all test problems. We observe, that simpler problems with less decision variables, which can be solved more easily (ZDT1, ZDT4, and ZDT6), require very loose constraints. On the other hand, harder problems (all DTLZ problems, ZDT2 and ZDT3) are solved better with harder constraints.

We believe this is affected by the difficulty of the problems. For easy problems a good local optimum is found quickly and then the islands need

| - | 0.0 | 0.1 | 0.2 | 0.3 | 0.5 | 1.0 | 2.0 | 5.0 | Inf |
|---|---|---|---|---|---|---|---|---|---|
| 0.0 | - | 0.741 | 0.837 | 0.0566 | **0.000670** | **7.42e-06** | **0.000902** | 0.00251 | 0.00806 |
| 0.1 | 0.259 | - | 0.569 | 0.0277 | **0.000102** | **3.50e-06** | **0.000481** | **0.000506** | 0.00141 |
| 0.2 | 0.163 | 0.431 | - | 0.0188 | **2.89e-06** | **3.09e-06** | **0.000375** | **6.57e-05** | **5.56e-05** |
| 0.3 | 0.943 | 0.972 | 0.981 | - | 0.404 | **0.000889** | 0.0236 | 0.430 | 0.624 |
| 0.5 | 0.999 | 1.00 | 1.00 | 0.596 | - | **0.000352** | 0.0183 | 0.540 | 0.862 |
| 1.0 | 1.00 | 1.00 | 1.00 | 0.999 | 1.00 | - | 0.808 | 1.00 | 1.00 |
| 2.0 | 0.999 | 1.00 | 1.00 | 0.976 | 0.982 | 0.192 | - | 0.982 | 0.992 |
| 5.0 | 0.997 | 0.999 | 1.00 | 0.570 | 0.460 | **0.000354** | 0.0180 | - | 0.803 |
| Inf | 0.992 | 0.999 | 1.00 | 0.376 | 0.138 | **0.000109** | 0.00768 | 0.197 | - |

Table 6.1: Statistical comparison (p-values of one sided T-test) of FCMOGA with different constraint looseness for HD-IBEA on ZDT2. Statistically significant results (significance level 0.001) are in bold. Low values mean, that the constraint looseness in the row is significantly better than the one in the column

to cooperate in order to improve this solution. Therefore, effective communication between the islands is needed. On the other hand, when solving the hard problems, the reduced search space seems to be more important than the communication. In the reduced search space, the individuals are closer to each other and therefore, the genetic operators become more effective. The progress of the algorithm in this case is faster thanks to the closeness of the individuals, and the importance of the communication decreases.

We believe that after a larger number of generations, when the algorithm converges towards the Pareto optimal front, the benefit of communication will be larger, and the algorithm would perform better with looser constraints.

The results achieved by HD-IBEA on DTLZ3 and DTLZ4 problems support this hypotheses. HD-IBEA converges on these problems much closer to the Pareto optimal front than the other algorithms. Its results are better with looser constraints (constraint looseness about 0.3). On contrary NSGA-II, which cannot converge to the optima on these problems, performs better with hard constraints. Similar observation can be done on the other test problems.

We also observe, that very loose constraints (but not with infinite looseness), often yield the worst results. In this case a lot of individuals, which violate the constraint stay in the population for a few generations and than die before they are sent to the other island. This probably makes the "effective size" of the population smaller than its actual size.

| - | 0.0 | 0.1 | 0.2 | 0.3 | 0.5 | 1.0 | 2.0 | 5.0 | Inf |
|---|---|---|---|---|---|---|---|---|---|
| 0.0 | - | 0.827 | 0.0168 | 0.00616 | **1.95e-05** | **1.24e-08** | **7.09e-10** | **1.52e-12** | **8.80e-11** |
| 0.1 | 0.173 | - | 0.00168 | **0.000527** | **1.89e-06** | **2.12e-09** | **1.05e-10** | **4.75e-13** | **1.80e-11** |
| 0.2 | 0.983 | 0.998 | - | 0.366 | 0.00400 | **9.95e-07** | **5.73e-08** | **3.23e-11** | **4.53e-09** |
| 0.3 | 0.994 | 0.999 | 0.634 | - | 0.00494 | **6.44e-07** | **7.47e-08** | **2.29e-11** | **3.99e-09** |
| 0.5 | 1.00 | 1.00 | 0.996 | 0.995 | - | 0.00157 | **2.10e-05** | **1.99e-08** | **3.55e-06** |
| 1.0 | 1.00 | 1.00 | 1.00 | 1.00 | 0.998 | - | 0.0160 | **0.000193** | 0.0112 |
| 2.0 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.984 | - | 0.147 | 0.546 |
| 5.0 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.853 | - | 0.902 |
| Inf | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 | 0.989 | 0.454 | 0.0979 | - |

Table 6.2: Statistical comparison (p-values of one sided T-test) of FCMOGA with different constraint looseness for NSGA-II on DTLZ1. Statistically significant results (significance level 0.001) are in bold. Low values mean, that the constraint looseness in the row is significantly better than the one in the column

| - | 0.0 | 0.1 | 0.2 | 0.3 | 0.5 | 1.0 | 2.0 | 5.0 | Inf |
|---|---|---|---|---|---|---|---|---|---|
| 0.0 | - | 0.0432 | **0.000323** | 0.0134 | **0.000300** | 0.0233 | 0.906 | 1.00 | 1.00 |
| 0.1 | 0.957 | - | 0.0282 | 0.237 | 0.0193 | 0.278 | 0.998 | 1.00 | 1.00 |
| 0.2 | 1.00 | 0.972 | - | 0.857 | 0.382 | 0.860 | 1.00 | 1.00 | 1.00 |
| 0.3 | 0.987 | 0.763 | 0.143 | - | 0.0991 | 0.529 | 0.999 | 1.00 | 1.00 |
| 0.5 | 1.00 | 0.981 | 0.618 | 0.901 | - | 0.901 | 1.00 | 1.00 | 1.00 |
| 1.0 | 0.977 | 0.722 | 0.140 | 0.471 | 0.0987 | - | 0.998 | 1.00 | 1.00 |
| 2.0 | 0.0939 | 0.00159 | **5.17e-06** | **0.000624** | **8.42e-06** | 0.00172 | - | 0.993 | 1.00 |
| 5.0 | **0.000408** | **3.12e-06** | **1.72e-08** | **3.85e-06** | **6.11e-08** | **1.98e-05** | 0.00656 | - | 0.869 |
| Inf | **2.96e-05** | **2.76e-07** | **2.22e-09** | **5.48e-07** | **1.01e-08** | **3.42e-06** | **0.000393** | 0.131 | - |

Table 6.3: Statistical comparison (p-values of one sided T-test) of FCMOGA with different constraint looseness for $\epsilon$-IBEA on ZDT4. Statistically significant results (significance level 0.001) are in bold. Low values mean, that the constraint looseness in the row is significantly better than the one in the column

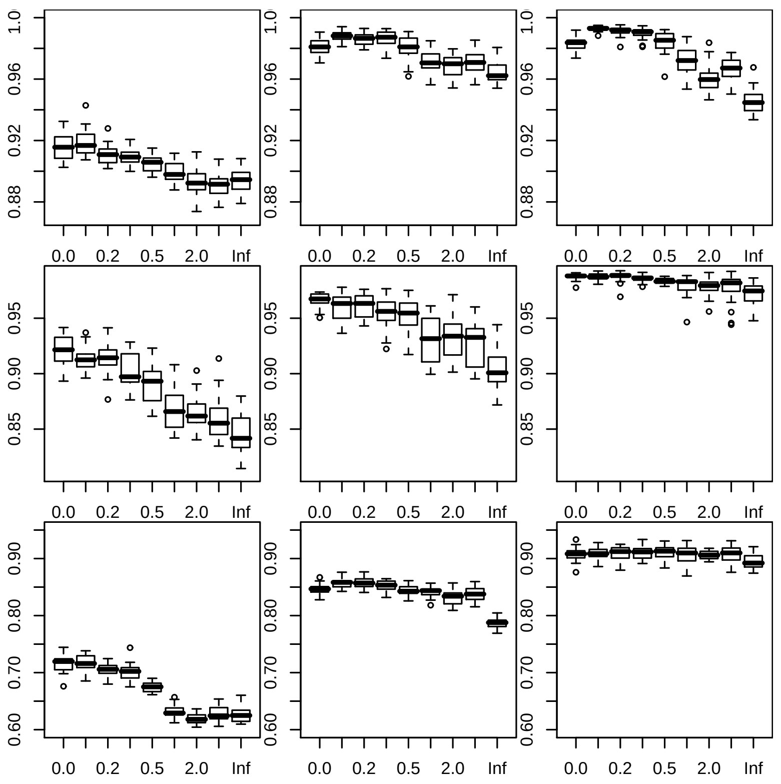

Figure 6.1: FCMOGA results on DTLZ1, DTLZ3, DTLZ4 (top to down). Each column presents results of one algorithm: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). Constraint looseness is set to 0, 0.1, 0.2, 0.3, 0.5, 1.0, 2.0, 5.0, and infinity. The boxplots are created after 20 runs for each setting. The horizontal axis is the constraint looseness, the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

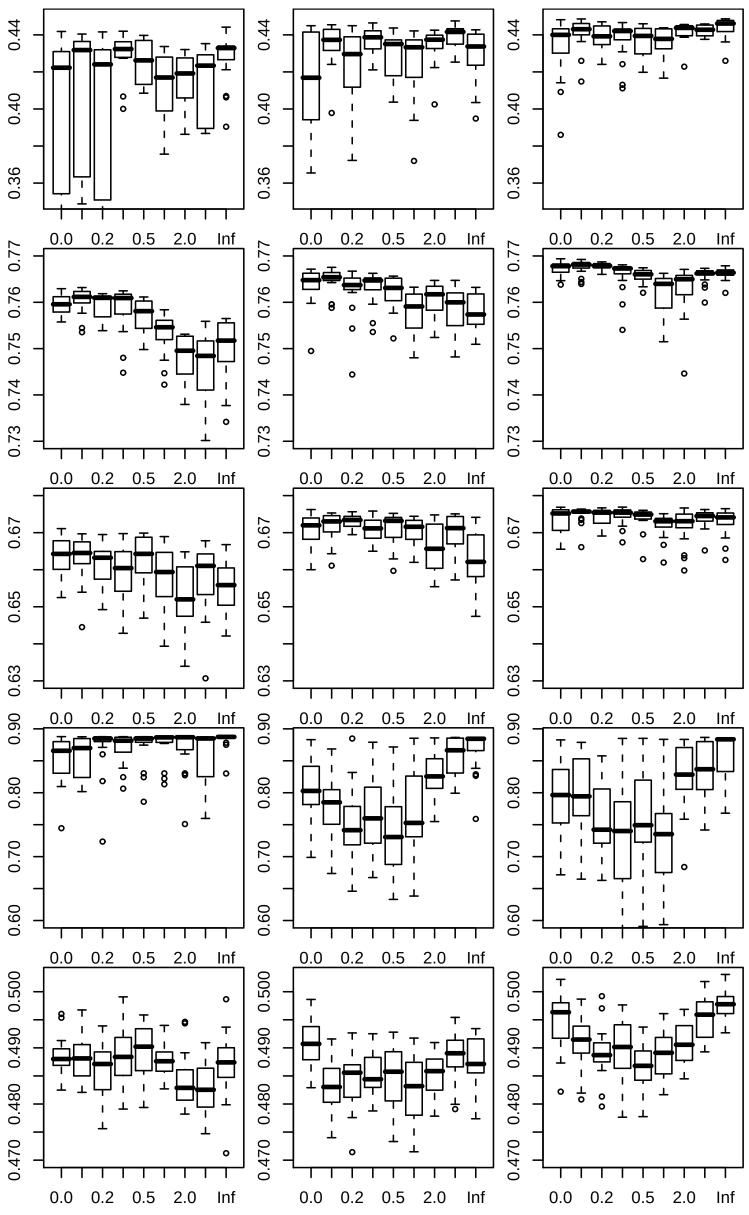

Figure 6.2: FCMOGA results on ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 (top to down). Each column presents results of one algorithm: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). Constraint looseness is set to 0, 0.1, 0.2, 0.3, 0.5, 1.0, 2.0, 5.0, and infinity. The boxplots were created using 20 runs for each setting. The horizontal axis is the constraint looseness, the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

# Chapter 7

# Heterogeneous Island Model as a Hybridizing Technique

Different MOGAs have different advantages. Domination based MOGAs are fast, but hypervolume based MOGAs can obtain better solutions. In this chapter we try to combine different MOGAs in a heterogeneous island model to combine the advantages.

To test the performance of the combined heterogeneous island MOGA (HIMOGA), we use two islands, each of them running different algorithm.

## 7.1 Test Setup

We used NSGA-II, $\epsilon$-IBEA, and HD-IBEA again. We combined all pairs of them and compared them to homogeneous islands with the same MOGA on each of them.

Several benchmark problems were used: ZDT1 to ZDT4 and ZDT6 (two objectives) from the ZDT set (Zitzler *et al.*, 2000), and DTLZ1, DTLZ3 and DTLZ4 from the DTLZ set (Deb *et al.*, 2005) (three objectives). We use 150 decision variables for ZDT1 to ZDT3, 10 decision variables for ZDT4, and 30 decision variables for ZDT6. The DTLZ problems use 300 decision variables. See Appendix A for details.

The evolution run for 500 generations with population size 100 on both islands. We also tested HIMOGA with the same population size and time-limit of 2 minutes instead of the limit on the number of generations.

## 7.2 Test Results

We run each of the tests 20 times and the boxplots in Figures 7.1 and 7.2 summarizes the results. Note, that some outliers may be removed from the boxplots to improve readability. Larger boxplots and statistical tests results can be found on the attached CD. Here, we present the T-test results only for two of the test cases, ZDT1 after 500 generations and DTLZ1 after 2 minutes.

Looking at the results after 500 generations, we can see, that the hybridization sometimes helps to find better solution. The faster of the two algorithms converges faster in the beginning and provides good individuals for the other MOGA. When the faster algorithm finishes, the other MOGA continues on its own. On many of the test functions, this leads to better solutions. Sometimes (ZDT1, ZDT3, ZDT6) the hybrid pair of islands finds better solution than any homogeneous pair of islands. We believe, that this results are affected by the fact, that the slower algorithm works with the resulting population of the faster one (or at least some individuals from this population), thus having “effectively” more generations.

In order to remove this effect, we tried running the evolution with a time limit of 2 minutes instead of a fixed number of generations. In this case, the results of the HIMOGA are not as good as in the previous group of tests. However, promising results are achieved on the ZDT1, ZDT3, DTLZ1 and DTLZ3 problems. Here HIMOGA ($\epsilon$-IBEA with HD-IBEA) achieved better results than the homogeneous islands MOGA.

HIMOGA shows, that heterogeneous island models can be used to improve the performance of MOGAs. We also believe, that if we used some even slower MOGA (i.e. a hypervolume based one), the differences would be larger.

| - | NN | EE | HH | NE | NH | EH |
|---|---|---|---|---|---|---|
| NN | - | 0.239 | 0.651 | 1.00 | 1.00 | 1.00 |
| EE | 0.761 | - | 0.789 | 0.996 | 0.997 | 0.997 |
| HH | 0.349 | 0.211 | - | 0.955 | 0.966 | 0.965 |
| NE | **1.05e-08** | 0.00445 | 0.0447 | - | 1.00 | 0.999 |
| NH | **4.39e-09** | 0.00329 | 0.0345 | **0.000476** | - | 0.438 |
| EH | **4.51e-09** | 0.00333 | 0.0348 | **0.000730** | 0.562 | - |

Table 7.1: Statistical comparison (p-values of one sided T-test) of HIMOGA with different MOGA combinations on ZDT1 after 500 generations. Statistically significant results (significance level 0.001) are in bold. Low values mean, that the combination in the row is significantly better than the one in the column. N, E, and H stands for NSGA-II, $\epsilon$-IBEA, and HD-IBEA respectively

| - | NN | EE | HH | NE | NH | EH |
|---|---|---|---|---|---|---|
| NN | - | 1.00 | 1.00 | 1.00 | 1.00 | 1.00 |
| EE | **0.00** | - | **0.00** | 0.835 | **0.00** | 0.999 |
| HH | **4.00e-15** | 1.00 | - | 1.00 | **5.65e-06** | 1.00 |
| NE | **0.00** | 0.165 | **0.00** | - | **0.00** | 0.991 |
| NH | **1.56e-10** | 1.00 | 1.00 | 1.00 | - | 1.00 |
| EH | **0.00** | **0.000521** | **0.00** | 0.00913 | **0.00** | - |

Table 7.2: Statistical comparison (p-values of one sided T-test) of HIMOGA with different MOGA combinations on DTLZ1 after 2 minutes. Statistically significant results (significance level 0.001) are in bold. Low values mean, that the combination in the row is significantly better than the one in the column. N, E, and H stands for NSGA-II, $\epsilon$-IBEA, and HD-IBEA respectively

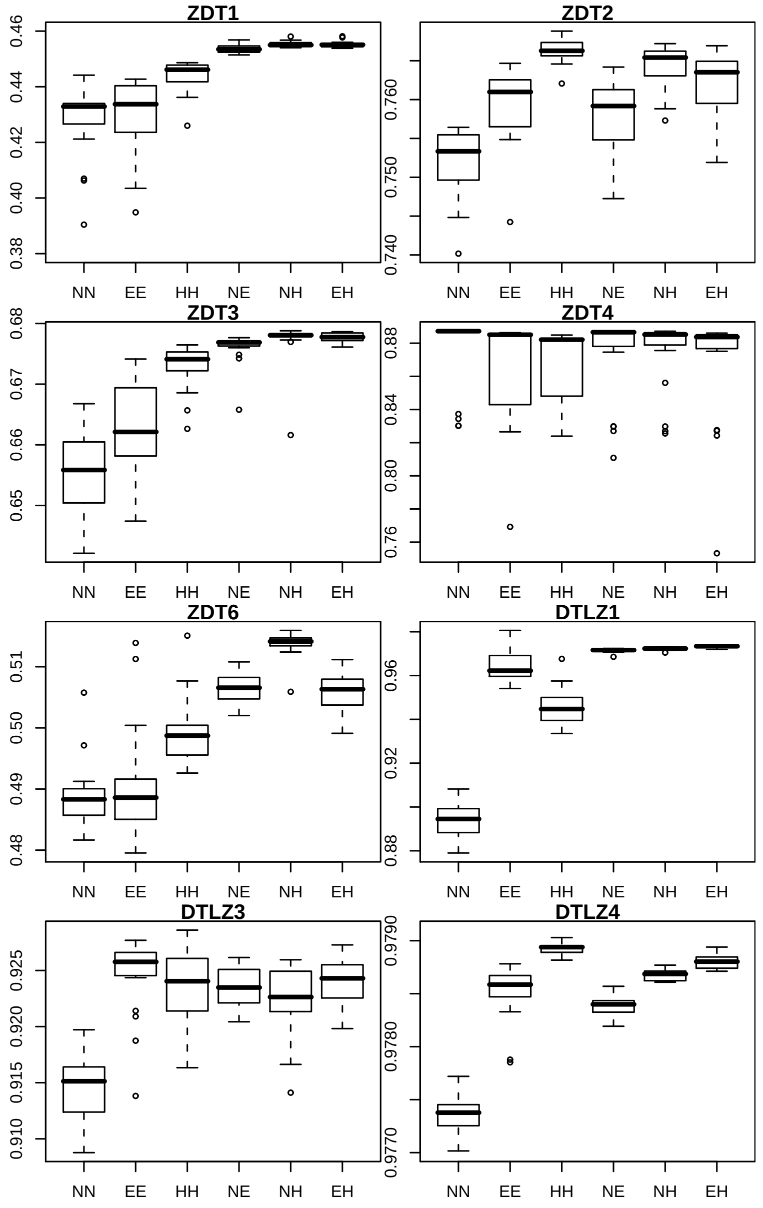

Figure 7.1: HIMOGA results on ZDT1, ZDT2, ZDT3, ZDT4, ZDT6, DTLZ1, DTLZ3 and DTLZ5 after 500 generations. The boxplots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms (N, E, and H stands for NSGA-II, $\epsilon$-IBEA, and HD-IBEA respectively), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

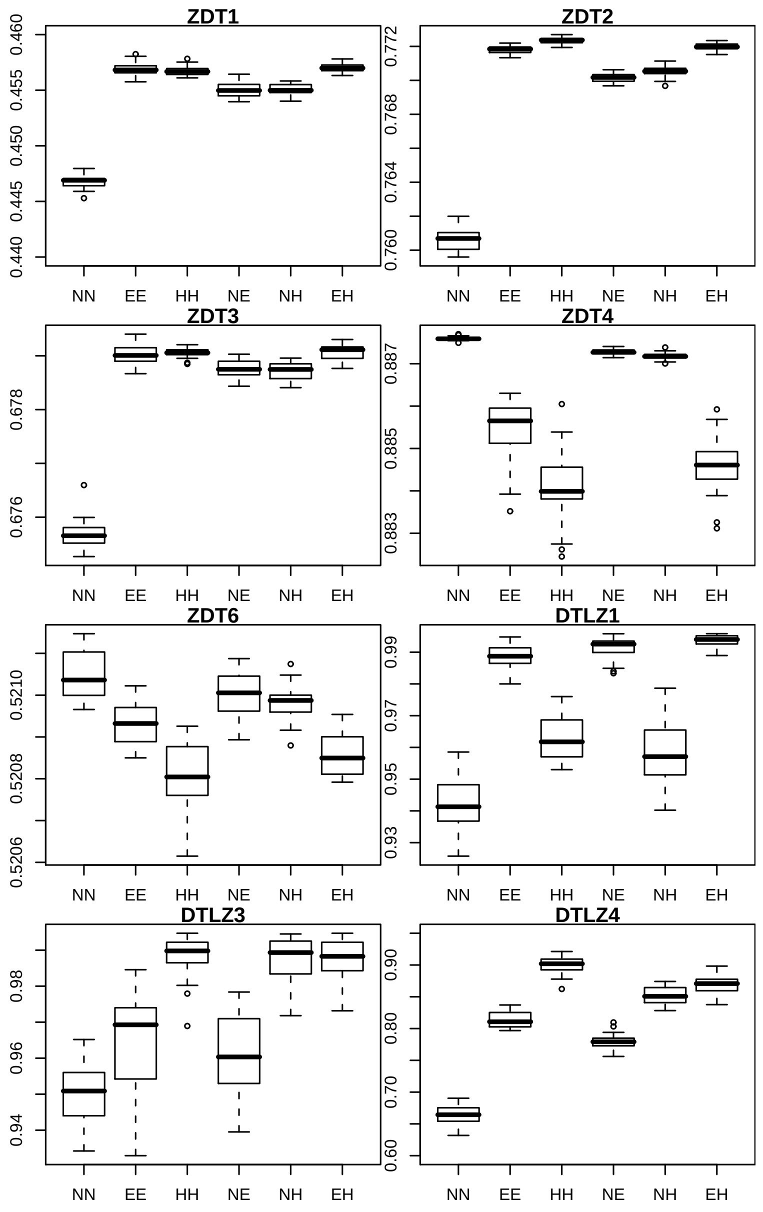

Figure 7.2: HIMOGA results on ZDT1, ZDT2, ZDT3, ZDT4, ZDT6, DTLZ1, DTLZ3 and DTLZ5 after 2 minutes. The boxplots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms (N, E, and H stands for NSGA-II, $\epsilon$-IBEA, and HD-IBEA respectively), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

# Chapter 8

# Using SOGA as a Local Search Technique in MOGA

MOGAs are good in finding good spread of solutions on the Pareto optimal front, but they are computationally expensive. On the other hand, SOGAs can only find a single Pareto optimal solution in each run, but they are much faster.

In this chapter we propose and test the use of SOGAs as a local search technique in MOGA. To combine these two algorithms, we use the heterogeneous island model where some islands run the MOGA while other islands run the SOGA. The SOGA runs only for a small number of generations, with a small population size, and therefore the whole evolution takes only a short time, MOGA can make only a few (usually less than 5) generations in the same amount of time.

We call this algorithm MOGASOLS (Multiobjective Genetic Algorithm with Single Objective Local Search).

## 8.1 Algorithm Description

We use the island model, where the islands are divided into two groups. One group contains the MOGA islands, and the other contains the SOGA islands.

The MOGA islands run a MOGA and after a number of generations they send some of the individuals to the SOGA island.

The SOGA island run a goal programming based single-objective genetic algorithm. They wait for individuals from the MOGA islands and as soon as they receive them they compute the goal vector and start the evolution. When the evolution finishes, they send the resulting population to the MOGA island.

The goal vector $\boldsymbol{g}$ is computed from the population $P$ as

$$g_i = \max_{\boldsymbol{x} \in P} f_i(\boldsymbol{x}) - 2(\max_{\boldsymbol{x} \in P} f_i(\boldsymbol{x}) - \min_{\boldsymbol{x} \in P} f_i(\boldsymbol{x}))$$

where $f_i$ is the $i$-th objective function.

**Algorithm 18** MOGASOLS – MOGA island

```
$t = 1$
Create random initial population $P$
while termination criterion not met do
    receive individuals from another islands and add them to $P$
    run one generation of MOGA
    if $t \mod c_{MOGA} = 0$ then
        Select $i_{MOGA}$ individuals and send them to another MOGA island
    end if
    if $t \mod c_{SOGA} = 0$ then
        Select $i_{SOGA}$ individuals and send them to a SOGA island
    end if
    $t = t + 1$
end while
```

## 8.2 Test Setup

To test the algorithm, we used the benchmark problems ZDT1 to ZDT4 and ZDT6 from the ZDT set (Zitzler *et al.*, 2000), and DTLZ1, DTLZ3 and DTLZ4 (three objectives) from the DTLZ set (Deb *et al.*, 2005). We use 150 decision variables for ZDT1 to ZDT3, 10 decision variables for ZDT4, and 30 decision variables for ZDT6. The DTLZ problems use 300 decision variables. See Appendix A for details.

We use simulated binary crossover (Deb & Agrawal, 1995) and real polynomial mutation on all islands.

On the MOGA islands we used NSGA-II, $\epsilon$-IBEA and HD-IBEA. The communication rate $c_{MOGA} = 10$, the number of individuals sent to another MOGA island is $i_{MOGA} = 20$. Every $c_{SOGA} = 10$ generations $i_{SOGA} = 20$

**Algorithm 19** MOGASOLS – SOGA island

```
while termination criterion not met do
    Wait for individuals from a MOGA island
    Receive individuals from MOGA island $M$ to $P$
    $t = 1$
    Compute goal vector $g = \max_{\boldsymbol{x}\in P} \boldsymbol{f}(\boldsymbol{x}) - 2(\max_{\boldsymbol{x}\in P} \boldsymbol{f}(\boldsymbol{x}) - \min_{\boldsymbol{x}\in P} \boldsymbol{f}(\boldsymbol{x}))$
    while $t < m_{SOGA}$ do
        Compute fitness $F(\boldsymbol{x}) = \sum_i^n (f_i(\boldsymbol{x}) - g_i)^2$
        Create population $P'$ using selection, crossover and mutation
        Merge $P$ and $P'$ and select $i_{SOGA}$ best individuals to the next generation
        $t = t + 1$
    end while
    Send $P$ back to MOGA island $M$
end while
```

| Configuration | Population size |
|---|---|
| 1+0/1+1/2+0 | 400 |
| 2+0/2+1/3+0 | 200 |
| 4+0/4+1/5+0 | 100 |
| 8+0/8+1/9+0 | 50 |

Table 8.1: MOGA island population size for different configurations of MOGASOLS

individuals are sent to a SOGA island. The SOGA islands evolves these individuals for $m_{SOGA} = 50$ generations. The MOGA islands terminates after 200 generations.

We tested different configurations and use the notion $a + b$ to distinguish them. Here, $a$ is the number of MOGA islands and $b$ is the number of SOGA islands. The basic configurations, are $1 + 0, 2 + 0, 4 + 0$, and $8 + 0$. We added one SOGA island to each of these configuration to test the effect of it ($1 + 1, 2 + 1, 4 + 1$, and $8 + 1$). We compared these to adding another MOGA island to the basic configurations ($2 + 0, 3 + 0, 5 + 0$, and $9 + 0$). For the population sizes used on the MOGA islands see Table 8.1. (Note, there are two 2+0 configurations with different population sizes, they can be distinguished by the other configurations mentioned in the same context).

## 8.3 Test Results

We run 20 tests for each of the configurations. Figures 8.2 to 8.9 show the boxplots of the $\mathcal{S}$-values achieved by these algorithm. Note that some outliers may be removed to improve the readability, for ZDT6 some of the boxplots are missing, as the results of this configuration are significantly worse, than the results of the other configurations. Larger boxplots together with statistic test results can be found on the attached CD.

The results show that using a SOGA island together with MOGA islands significantly improves the performace of the multiobjective optimizer on most test problems used in this work. The difference gets larger as the number of islands increases, which means that MOGASOLS works particularly good with smaller populations (see Figure 8.1 for comparison on ZDT2). Not only does MOGASOLS improve the $\mathcal{S}$-value of the resulting Pareto set approximations, it also uses less function evaluations. With our parameters, the SOGA island needs 1000 evaluations for each invocation, the same number the MOGA island (with 400 individuals in population) needs for 2.5 generations, i.e. the SOGA islands uses four times lower number of function evaluation than the MOGA island (they are invoked every 10 generations). This means that using MOGASOLS can also reduce the execution time, when solving problems with hard-to-compute objective functions.

However, there are also some cases, where MOGASOLS does not work better than the other algorithms with similar population sizes.

On ZDT1, when the SOGA island is used with NSGA-II based MOGA islands, the solutions from SOGA island causes premature convergence on the MOGA islands, thus degrading the results significantly. This could be solved using less frequent communication between the MOGA and SOGA islands or not using all the solutions found by the SOGA island.

On those problems, where the basic configuration with no SOGA island can find the optimal solution, no improvement is achieved with MOGASOLS. This is also supported by the results of T-test on DTLZ4 for NSGA-II and HD-IBEA, which are presented in Tables 8.2 and 8.3. The configuration with one more MOGA islands provides the best results, as it has more individuals in the combined population, thus providing better spread of the solutions on the Pareto front. However, the results on these problems are practically identical for all configurations.

| - | 2+0 | 1+0 | 1+1 |
|---|---|---|---|
| 2+0 | - | 0.00277 | 1.00 |
| 1+0 | 0.997 | - | 1.00 |
| 1+1 | **8.03e-09** | **4.65e-11** | - |

Table 8.2: Statistical comparison (p-values of one sided T-test) of MOGASOLS with different configurations and NSGA-II on DTLZ4. Statistically significant results (significance level 0.001) are in bold. Low values mean, that the configuration in the row is significantly better than the one in the column

| - | 2+0 | 1+0 | 1+1 |
|---|---|---|---|
| 2+0 | - | **4.59e-07** | **1.72e-08** |
| 1+0 | 1.00 | - | **0.000842** |
| 1+1 | 1.00 | 0.999 | - |

Table 8.3: Statistical comparison (p-values of one sided T-test) of MOGASOLS with different configurations and HD-IBEA on DTLZ4. Statistically significant results (significance level 0.001) are in bold. Low values mean, that the configuration in the row is significantly better than the one in the column

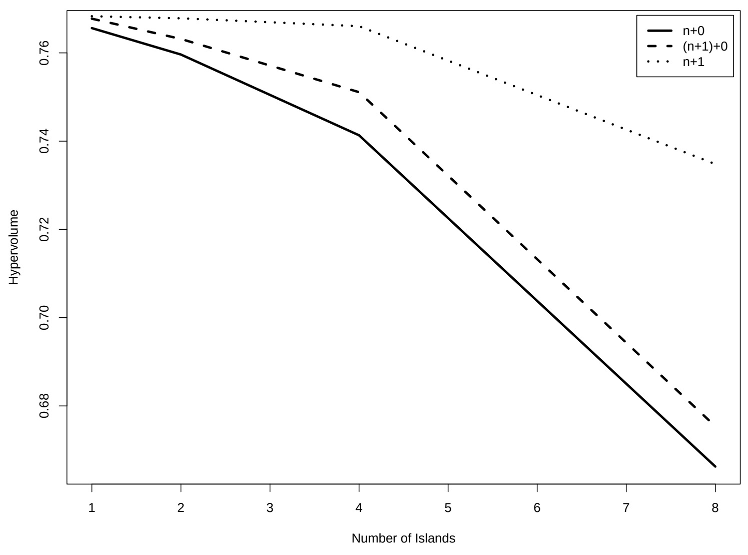

Figure 8.1: Dependence of average $\mathcal{S}$-value achieved by MOGASOLS with HD-IBEA on the number of islands used for different configurations (ZDT2 test problem).

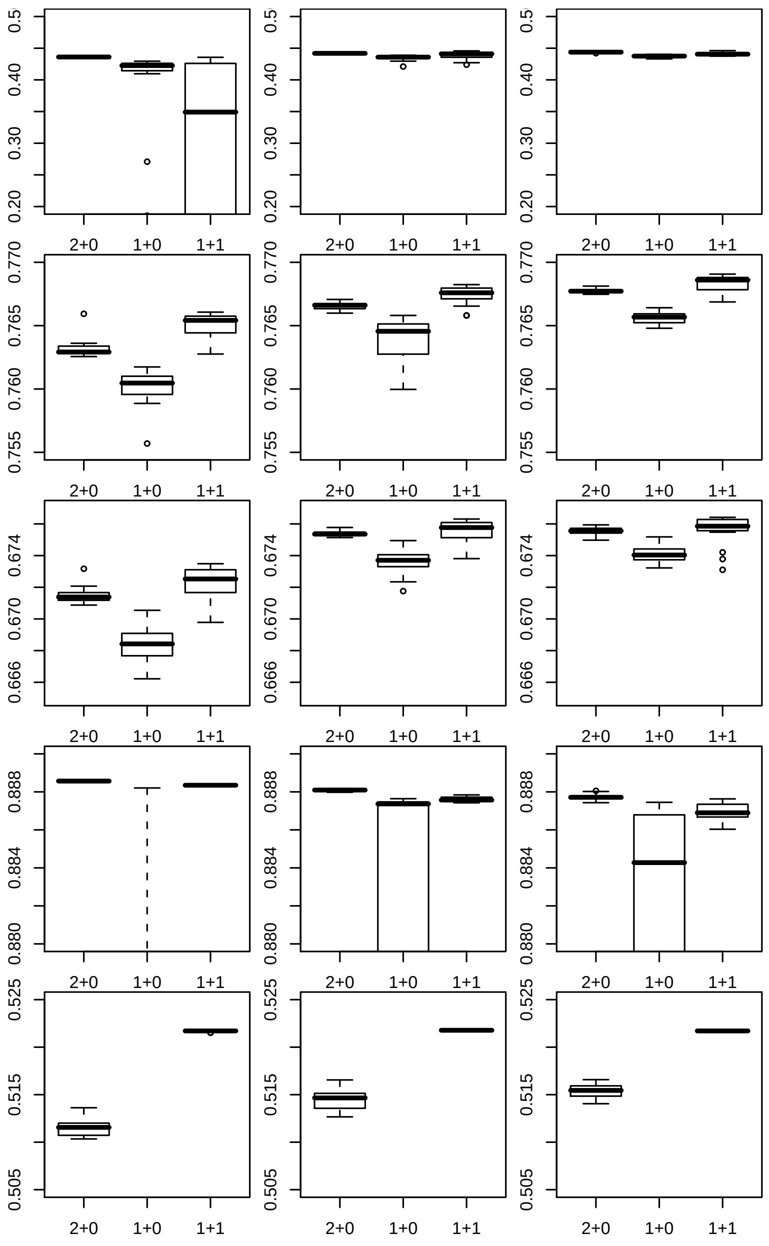

Figure 8.2: MOGASOLS results on ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The box-plots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a + b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

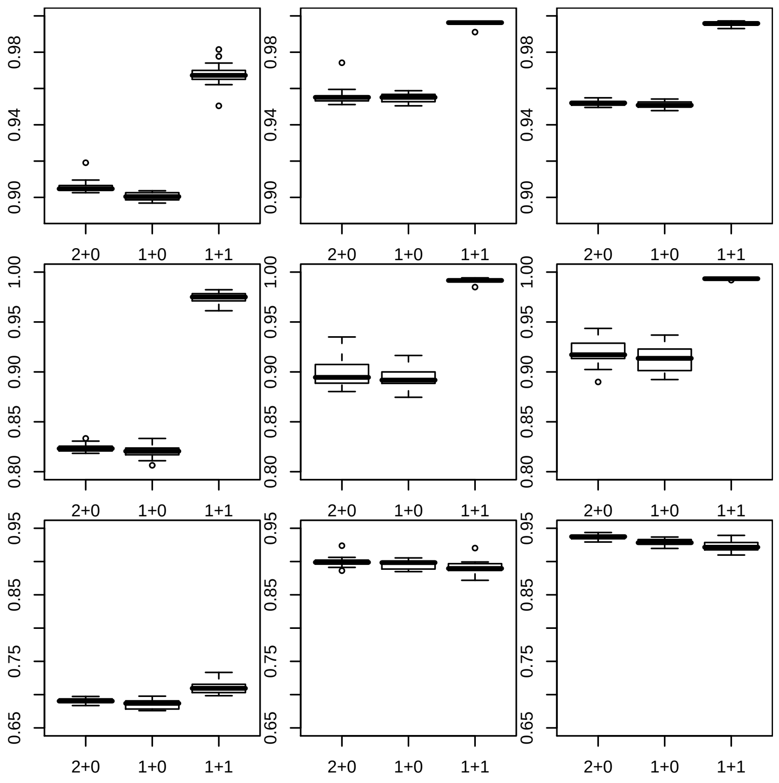

Figure 8.3: MOGASOLS results on DTLZ1, DTLZ3, and DTLZ4 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The boxplots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a + b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

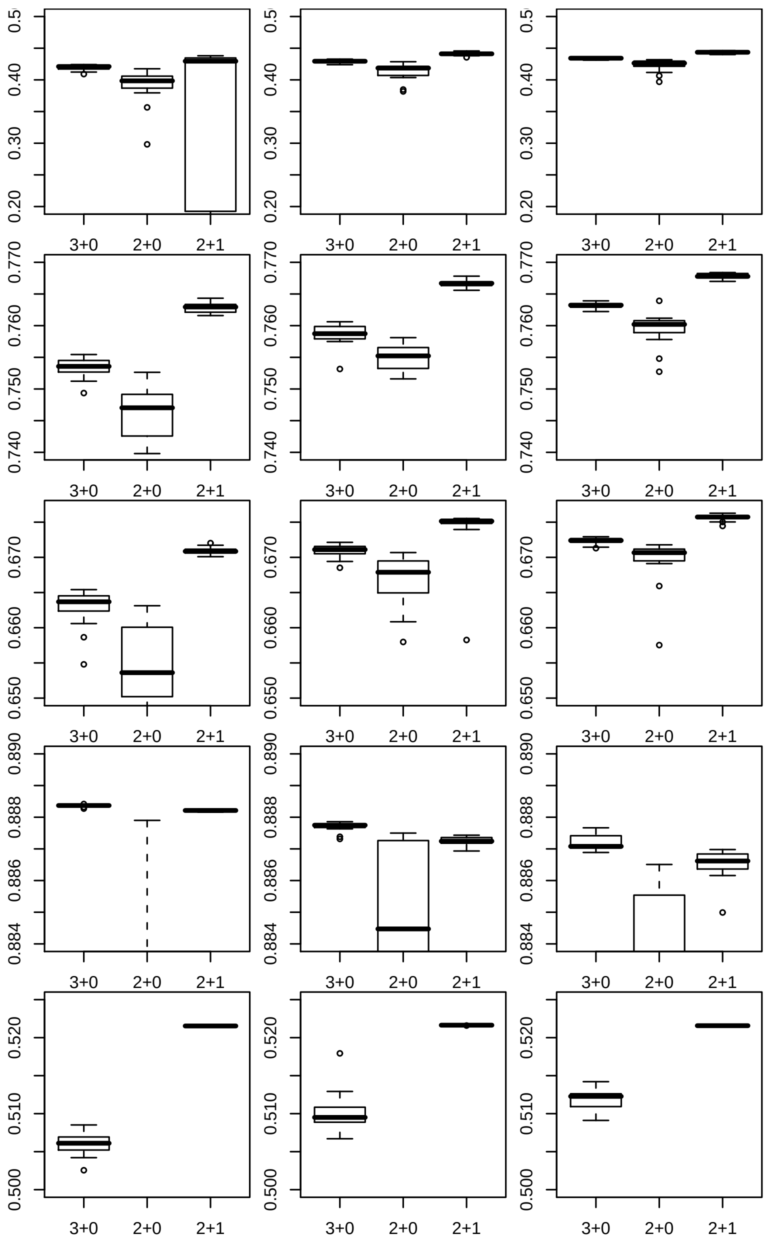

Figure 8.4: MOGASOLS results on ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The box-plots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a + b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

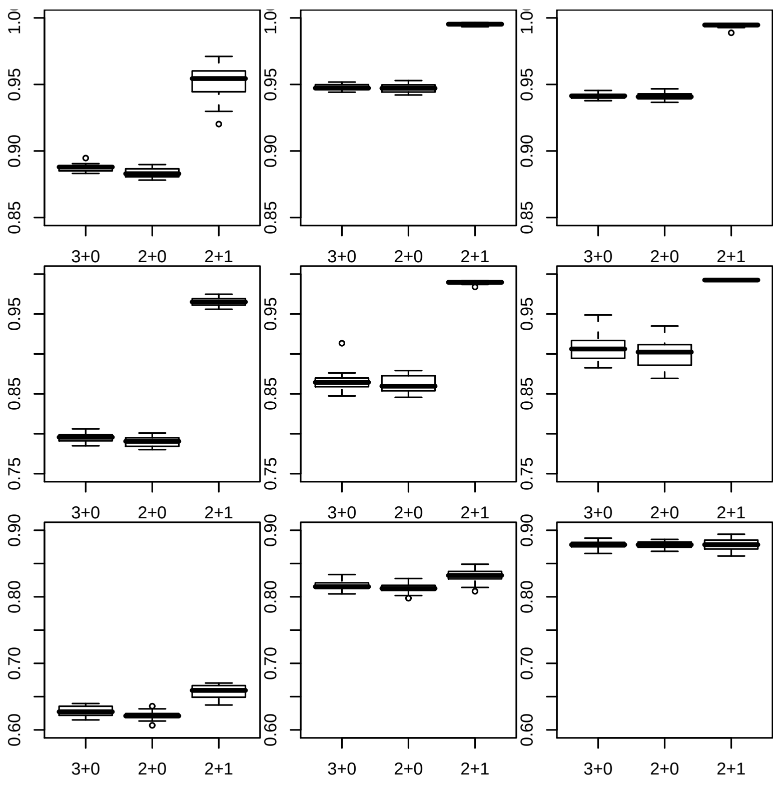

Figure 8.5: MOGASOLS results on DTLZ1, DTLZ3, and DTLZ4 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The boxplots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a$ + $b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

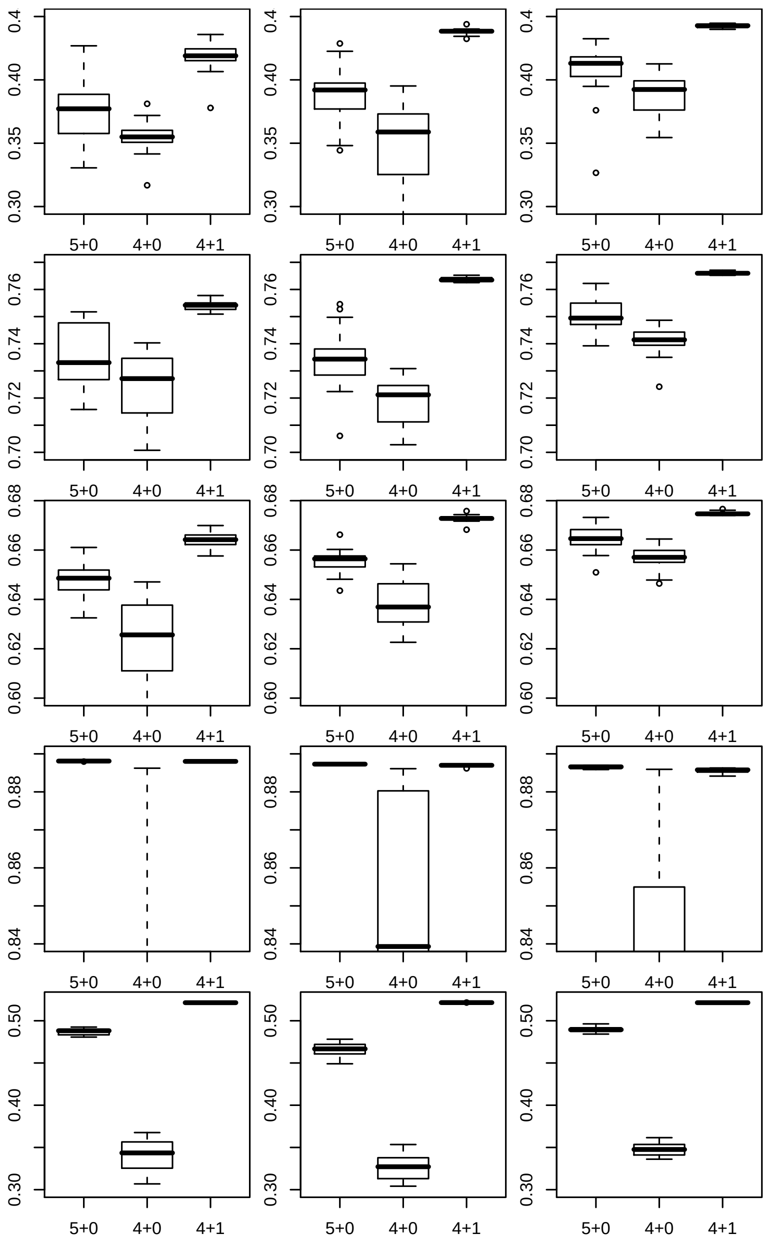
Figure 8.6: MOGASOLS results on ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The box-plots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a + b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

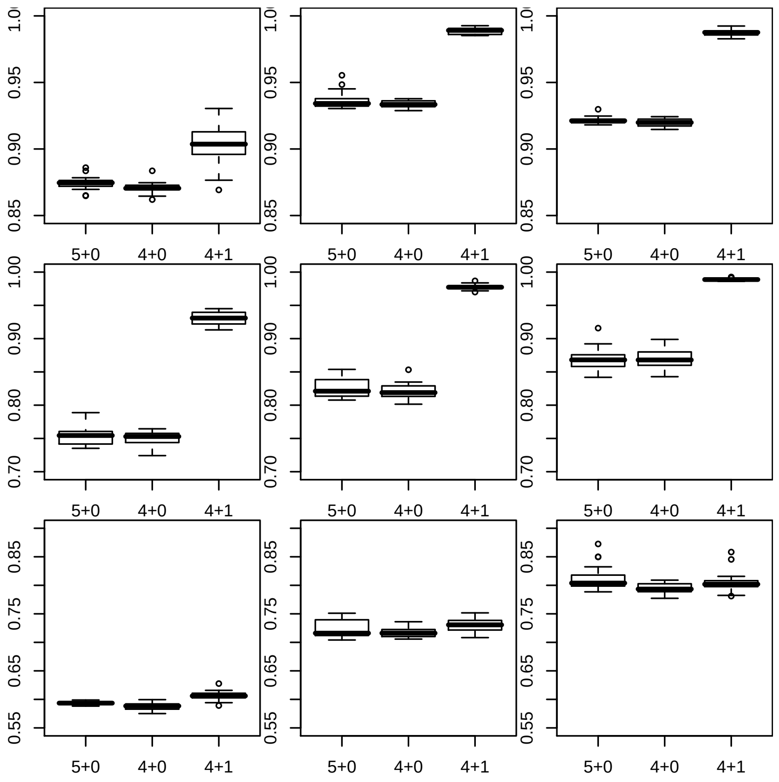

Figure 8.7: MOGASOLS results on DTLZ1, DTLZ3, and DTLZ4 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The boxplots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a$ + $b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

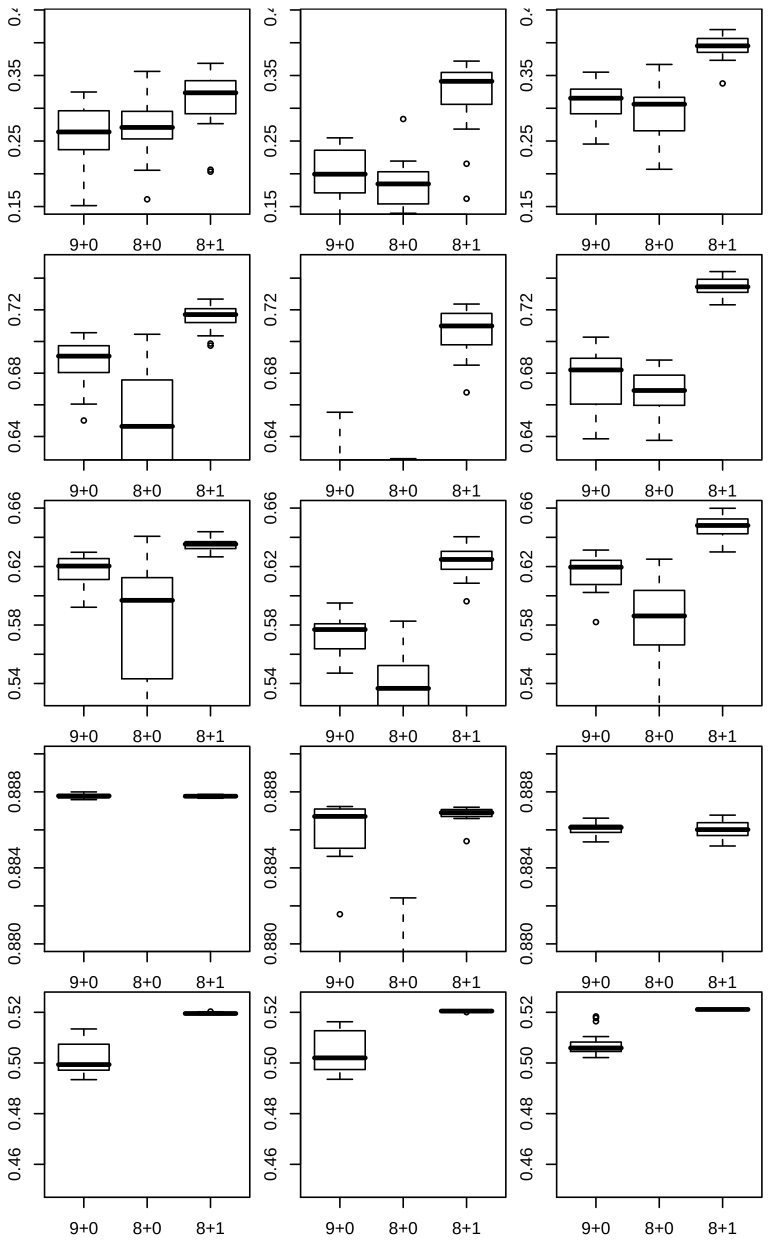

Figure 8.8: MOGASOLS results on ZDT1, ZDT2, ZDT3, ZDT4, and ZDT6 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The box-plots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a + b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

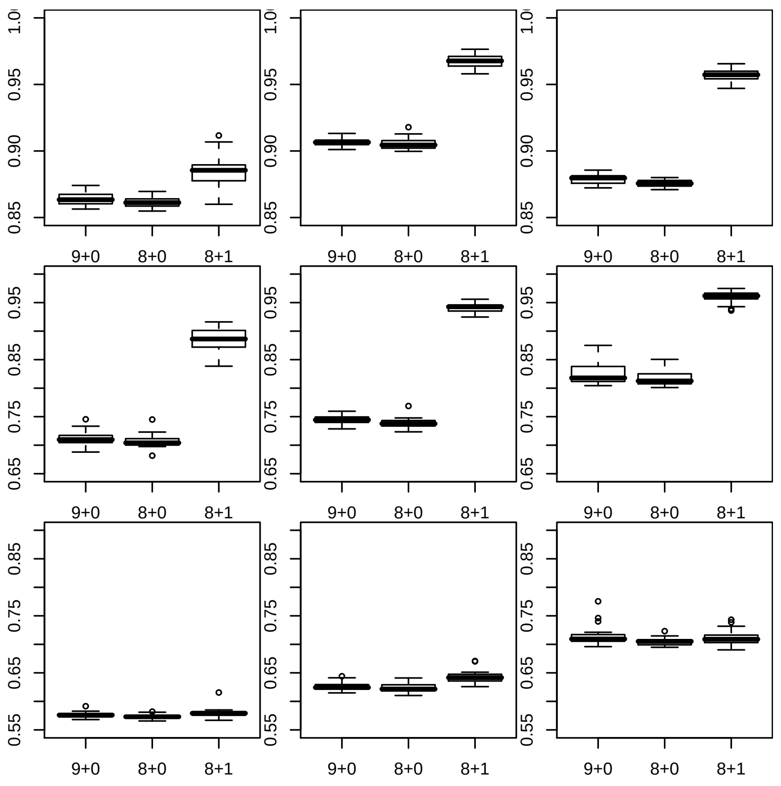

Figure 8.9: MOGASOLS results on DTLZ1, DTLZ3, and DTLZ4 (top to bottom). Each column presents results of one algorithm on the MOGA islands: NSGA-II, $\epsilon$-IBEA and HD-IBEA (left to right). The boxplots were created using 20 runs for each setting. The horizontal axis presents different combinations of the algorithms ($a + b$ means, that $a$ MOGA islands and $b$ SOGA islands were used), the vertical axis is the $\mathcal{S}$-value achieved by the algorithm.

# Chapter 9

# Conclusions and Future Work

In this work, we studied the options for parallelization of multiobjective evolutionary algorithms. The goal was to propose a new parallel MOGA, which would perform well on a set of selected benchmark problems. In order to achieve this goal, we tried three different approaches. Among them, using single-objective local search proved to be the most promising one.

We discussed that using fuzzy constraints can improve the performance of island based MOGA. We experimentally showed that harder constraints should be used in the beginning of the evolution and softer constraints work better when fine-tuning the Pareto front approximation. Based on this observation, we suggest using adaptive constraint looseness which would increase during the evolution. We believe this would further improve the performance of parallel MOGAs.

The results of HIMOGA showed that hybridizing different MOGAs in the island model can under certain circumstances yield better solutions than using more island with the same MOGA. However, the differences are not very large, and the use of this approach in practical situations is questionable. More viable approach could be using the final population of one MOGA as an initial population for another (possibly slower, but better) MOGA.

Finally, we showed that combining MOGAs and SOGAs in a heterogeneous island model yields better results, than using more MOGA islands. Moreover, it can also decrease the number of function evaluations, thus reducing the runtime. On some test problems, this was the only approach that was able to provide the optimal solutions.

In the future, we plan to test the algorithms proposed in this work on a wider set of benchmark problems from various fields.

We will also continue studying the algorithms proposed in this work. We would like to create a version of FCMOGA with adaptive constraint looseness to combine its advantages on different problems and in different stages of the evolution. We also plan to test HIMOGA with hard-to-compute objective functions where the speed differences will be smaller, thus limiting the advantage of faster MOGAs. We will also test MOGASOLS with different SOGAs and study its performance on problems with more objectives. We will try to use this approach to construct an algorithm which would use lower number of function evaluations, thus improving the runtime.

# Appendix A

# Test problems

**ZDT1**

$$\begin{aligned}\text{Minimize } \boldsymbol{f}_1 &= x_1 \\ \text{Minimize } \boldsymbol{f}_2 &= g(\boldsymbol{x})h(\boldsymbol{f}_1, g)\end{aligned}$$

where $x_i \in [0, 1]$,

$$g(\boldsymbol{x}) = 1 + 9\sum_{i=2}^{m} x_i/(m-1)$$

and

$$h(\boldsymbol{f}_1, g) = 1 - \sqrt{\boldsymbol{f}_1/g}$$

We use $m = 150$ in this work.

**ZDT2**

$$\begin{aligned}\text{Minimize } \boldsymbol{f}_1 &= x_1 \\ \text{Minimize } \boldsymbol{f}_2 &= g(\boldsymbol{x})h(\boldsymbol{f}_1, g)\end{aligned}$$

where $x_i \in [0, 1]$,

$$g(\boldsymbol{x}) = 1 + 9\sum_{i=2}^{m} x_i/(m-1)$$

and

$$h(\boldsymbol{f}_1, g) = 1 - (\boldsymbol{f}_1/g)^2$$

We use $m = 150$ in this work.

**ZDT3**

$$\begin{aligned}\text{Minimize } \boldsymbol{f}_1 &= x_1\\ \text{Minimize } \boldsymbol{f}_2 &= g(\boldsymbol{x})h(\boldsymbol{f}_1, g)\end{aligned}$$

where $x_i \in [0,1]$,

$$g(\boldsymbol{x}) = 1 + 9\sum_{i=2}^{m} x_i/(m-1)$$

and

$$h(\boldsymbol{f}_1, g) = 1 - \sqrt{\boldsymbol{f}_1/g} - (\boldsymbol{f}_1/g)\sin(10\pi \boldsymbol{f}_1)$$

We use $m = 150$ in this work.

**ZDT4**

$$\begin{aligned}\text{Minimize } \boldsymbol{f}_1 &= x_1\\ \text{Minimize } \boldsymbol{f}_2 &= g(\boldsymbol{x})h(\boldsymbol{f}_1, g)\end{aligned}$$

where $x_i \in [0,1]$,

$$g(\boldsymbol{x}) = 1 + 10(m-1) + \sum_{i=2}^{m}(x_i^2 - 10\cos(4\pi x_i))$$

and

$$h(\boldsymbol{f}_1, g) = 1 - (\boldsymbol{f}_1/g)^2$$

We use $m = 10$ in this work.

**ZDT6**

$$\begin{aligned}\text{Minimize } \boldsymbol{f}_1 &= x_1 - e^{-4x_1}\sin^6(6\pi x_1)\\ \text{Minimize } \boldsymbol{f}_2 &= g(\boldsymbol{x})h(\boldsymbol{f}_1, g)\end{aligned}$$

where $x_i \in [0,1]$,

$$g(\boldsymbol{x}) = 1 + 9[\sum_{i=2}^{m} x_i/(m-1)]^{0.25}$$

and

$$h(\boldsymbol{f}_1, g) = 1 - (\boldsymbol{f}_1/g)^2$$

We use $m = 30$ in this work.

**DTLZ1**

$$\begin{aligned}
\text{Minimize } \boldsymbol{f}_1 &= \frac{1}{2}x_1x_2(1+g(\boldsymbol{x}_3)) \\
\text{Minimize } \boldsymbol{f}_2 &= \frac{1}{2}x_1(1-x_2)(1+g(\boldsymbol{x}_3)) \\
\text{Minimize } \boldsymbol{f}_2 &= \frac{1}{2}(1-x_1)(1-x_2)(1+g(\boldsymbol{x}_3))
\end{aligned}$$

where $0 \leq x_i \leq 1, \forall i$ and functional $g(\boldsymbol{x}_3)$ requires $|\boldsymbol{x}_3| = k$ variables

$$g(\boldsymbol{x}_3) = 100(|\boldsymbol{x}_3| + \sum_{x_i \in \boldsymbol{x}_3} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)))$$

In this work, we use $k = 300$

**DTLZ3**

$$\begin{aligned}
\text{Minimize } \boldsymbol{f}_1 &= (1+g(\boldsymbol{x}_3))\cos x_1\pi/2\cos x_2\pi/2 \\
\text{Minimize } \boldsymbol{f}_2 &= (1+g(\boldsymbol{x}_3))\cos x_1\pi/2\sin x_2\pi/2 \\
\text{Minimize } \boldsymbol{f}_2 &= (1+g(\boldsymbol{x}_3))\sin x_2\pi/2
\end{aligned}$$

where $0 \leq x_i \leq 1, \forall i$ and functional $g(\boldsymbol{x}_3)$ requires $|\boldsymbol{x}_3| = k$ variables

$$g(\boldsymbol{x}_3) = 100(|\boldsymbol{x}_3| + \sum_{x_i \in \boldsymbol{x}_3} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)))$$

In this work, we use $k = 300$

**DTLZ4**

$$\begin{aligned}
\text{Minimize } \boldsymbol{f}_1 &= (1+g(\boldsymbol{x}_3))\cos x_1^\alpha\pi/2\cos x_2^\alpha\pi/2 \\
\text{Minimize } \boldsymbol{f}_2 &= (1+g(\boldsymbol{x}_3))\cos x_1^\alpha\pi/2\sin x_2^\alpha\pi/2 \\
\text{Minimize } \boldsymbol{f}_2 &= (1+g(\boldsymbol{x}_3))\sin x_2^\alpha\pi/2
\end{aligned}$$

where $0 \leq x_i \leq 1, \forall i$ and functional $g(\boldsymbol{x}_3)$ requires $|\boldsymbol{x}_3| = k$ variables

$$g(\boldsymbol{x}_3) = \sum_{x_i \in \boldsymbol{x}_3} (x_i^\alpha - 0.5)^2$$

In this work, we use $k = 300$ and $\alpha = 100$

# Bibliography

ALAYA, I., SOLNON, C. & GHEDIRA, K. (2007). Ant colony optimization for multi-objective optimization problems. In: *ICTAI '07: Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence*. Washington, DC, USA: IEEE Computer Society.

AUGER, A., BADER, J., BROCKHOFF, D. & ZITZLER, E. (2009). Theory of the Hypervolume Indicator: Optimal $\mu$-Distributions and the Choice of the Reference Point. In: *Foundations of Genetic Algorithms (FOGA 2009)*. Workshop version.

BADER, J. & ZITZLER, E. (2008). HypE: An Algorithm for Fast Hypervolume-Based Many-Objective Optimization. TIK Report 286, Computer Engineering and Networks Laboratory (TIK), ETH Zurich.

BEUME, N., NAUJOKS, B. & EMMERICH, M. (2007). SMS-EMOA: Multiobjective selection based on dominated hypervolume. *European Journal of Operational Research* **181**(3), 1653–1669.

BEUME, N. & RUDOLPH, G. (2006). Faster s-metric calculation by considering dominated hypervolume as klee's measure problem. In: *Computational Intelligence* (KOVALERCHUK, B., ed.). IASTED/ACTA Press.

BRINGMANN, K. & FRIEDRICH, T. (2008). Approximating the volume of unions and intersections of high-dimensional geometric objects. *CoRR* **abs/0809.0835**.

BROCKHOFF, D. & ZITZLER, E. (2007). Improving Hypervolume-based Multiobjective Evolutionary Algorithms by Using Objective Reduction Methods. In: *Congress on Evolutionary Computation (CEC 2007)*. IEEE Press.

Cancino, W. & Delbem, A. C. B. (2006). A multi-objective evolutionary approach for phylogenetic inference. In: Obayashi *et al.* (2007), pp. 428–442.

Chiam, S. C., Tan, K. C. & Mamun, A. A. (2006). Multiobjective evolutionary neural networks for time series forecasting. In: Obayashi *et al.* (2007), pp. 346–360.

Cook, R., Molina-Cristobal, A., Parks, G. T., Correa, C. O. & Clarkson, P. J. (2006). Multi-objective optimisation of a hybrid electric vehicle: Drive train and driving strategy. In: Obayashi *et al.* (2007), pp. 330–345.

Das, I. & Dennis, J. E. (1998). Normal-boundary intersection: a new method for generating Pareto optimal points in multicriteria optimization problems. *SIAM Journal on Optimization* **8(3)**, 631–657. URL `citeseer.ist.psu.edu/das96normalboundary.html`.

Datta, D., Deb, K. & Fonseca, C. M. (2006). Multi-objective evolutionary algorithms for resource allocation problems. In: Obayashi *et al.* (2007), pp. 401–416.

Deb, K. & Agrawal, R. B. (1995). Simulated binary crossover for continuous search space. In: *Complex Systems*, vol. 9. pp. 115–148.

Deb, K., Agrawal, S., Pratap, A. & Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimisation: NSGA-II. In: *PPSN* (Schoenauer, M., Deb, K., Rudolph, G., Yao, X., Lutton, E., Guervós, J. J. M. & Schwefel, H.-P., eds.), vol. 1917 of *Lecture Notes in Computer Science*. Springer.

Deb, K., Mohan, M. & Mishra, S. (2003). A Fast Multi-objective Evolutionary Algorithm for Finding Well-Spread Pareto-Optimal Solutions. KanGAL report 2003002, Indian Institute of Technology, Kanpur, India.

Deb, K., Thiele, L., Laumanns, M. & Zitzler, E. (2005). Scalable Test Problems for Evolutionary Multi-Objective Optimization. In: *Evolutionary Multiobjective Optimization: Theoretical Advances and Applications* (Abraham, A., Jain, R. & Goldberg, R., eds.), chap. 6. Springer, pp. 105–145.

DORIGO, M. (1992). *Optimization, Learning, Natural Algorithms.* Ph.D. thesis, Politecnico di Milano, Italy.

FLEISCHER, M. (2003). The measure of pareto optima. In: *EMO* (FONSECA, C. M., FLEMING, P. J., ZITZLER, E., DEB, K. & THIELE, L., eds.), vol. 2632 of *Lecture Notes in Computer Science.* Springer.

FONSECA, C. M. & FLEMING, P. J. (1993). Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: *Genetic Algorithms: Proceedings of the Fifth International Conference.* Morgan Kaufmann. URL `http://citeseer.ist.psu.edu/175195.html`.

GOLDBERG, D. E. (1989). *Genetic Algorithms in Search Optimization and Machine Learning.* Addison-Wesley.

GRONWALD, W., HOHM, T. & HOFFMANN, D. (2008). Evolutionary pareto-optimization of stably folding peptides. vol. 9. URL `http://www.biomedcentral.com/1471-2105/9/109`.

HIROYASU, T., MIKI, M. & WATANABE, S. (1999). Divided range genetic algorithms in multiobjective optimization problems. In: *In Proceedings of International Workshop on Emergent Synthesis (IWES'99.*

HORN, J., NAFPLIOTIS, N. & GOLDBERG, D. E. (1994). A niched pareto genetic algorithm for multiobjective optimization. In: *Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence.*

HUGHES, E. J. (2005). Evolutionary many-objective optimisation: Many once or one many? In: *2005 IEEE Congress on Evolutionary Computation (CEC'2005)*, vol. 1. Edinburgh, Scotland: IEEE Service Center.

ISHIBUCHI, H., TSUKAMOTO, N. & NOJIMA, Y. (2008). Evolutionary many-objective optimization: A short review. In: *Proceedings of 2008 IEEE Congress on Evolutionary Computation.*

JAIMES, A. L. & COELLO, C. A. C. (2007). Mrmoga: a new parallel multi-objective evolutionary algorithm based on the use of multiple resolutions. *Concurrency and Computation: Practice and Experience* **19**(4), 397–441.

JAKOBSSON, S., EDELVIK, F. & ANDERSSON, B. (2006). Multiobjective optimization in computational electromagnetics.

Kamiura, J., Hiroyasu, T., Miki, M. & Watanabe, S. (2002). Mogades: multi-objective genetic algorithm with distributed environment scheme. Atlanta, GA, USA: Dynamic Publishers, Inc.

Kennedy, J. & Eberhart, R. (1995). Particle swarm optimization. In: *IEEE Int'l. Conf. on Neural Networks, IV*.

Kim, M., Hiroyasu, T., Miki, M. & Watanabe, S. (2004). SPEA2+: Improving the performance of the strength pareto evolutionary algorithm 2. In: *PPSN* (Yao, X., Burke, E. K., Lozano, J. A., Smith, J., Guervós, J. J. M., Bullinaria, J. A., Rowe, J. E., Tiño, P., Kabán, A. & Schwefel, H.-P., eds.), vol. 3242 of *Lecture Notes in Computer Science*. Springer.

Messac, A., Ismail-Yahaya, A. & Mattson, C. A. (2003). The normalized normal constraint method for genereating the pareto frontier. In: *Structural and Multidisciplinary Optimization*, vol. 25.

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B. & Alba, E. (2008). A cellular genetic algorithm for multiobjective algorithm. In: *International Journal of Intelligent Systems*.

Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T. & Murata, T. (eds.) (2007). *Evolutionary Multi-Criterion Optimization, 4th International Conference, EMO 2007, Matsushima, Japan, March 5-8, 2007, Proceedings*, vol. 4403 of *Lecture Notes in Computer Science*. Springer.

Okuda, T., Hiroyasu, T., Miki, M. & Watanabe, S. (2002). DC-MOGA: Distributed cooperation model of multi-objective genetic algorithm. In: *Science and Engineering Review of Doshisha University*, vol. 42.

Reyes-Sierra, M. & Coello, C. A. C. (2006). Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computational Intelligence Research* **2**(3).

Ripon, K. S. N., Tsang, C.-H. & Kwong, S. (2007). An evolutionary approach for solving the multi-objective job-shop scheduling problem. In: *Evolutionary Scheduling* (Dahal, K. P., Tan, K. C. & Cowling, P. I., eds.), vol. 49 of *Studies in Computational Intelligence*. Springer, pp. 165–195.

Saha, S. & Bandyopadhyay, S. (2008). Application of multiobjective optimization for data clustering.

Schaffer, J. D. (1985). Multiple objective optimization with vector evaluated genetic algorithms. In: *ICGA* (Grefenstette, J. J., ed.). Lawrence Erlbaum Associates.

Srinivas, N. & Deb, K. (1994). Multiobjective optimization using non-dominated sorting in genetic algorithms. *Evolutionary Computation* **2**(3), 221–248.

Storn, R. & Price, K. (1995). Differential evolution- a simple and efficient adaptive scheme for global optimization over continuous spaces. Tech. rep.

Streichert, F., Ulmer, H. & Zell, A. (2005). Parallelization of multi-objective evolutionary algorithms using clustering algorithms. In: *EMO* (Coello, C. A. C., Aguirre, A. H. & Zitzler, E., eds.), vol. 3410 of *Lecture Notes in Computer Science*. Springer.

Watanabe, S., Hiroyasu, T. & Miki, M. (2002). NCGA: Neighborhood cultivation genetic algorithm for multi-objective optimization psroblems. In: *2002 Genetic and Evolutionary Computation Conference. Late-Breaking Papers* (Cantú-Paz, E., ed.). New York.

While, R. L., Hingston, P., Barone, L. & Huband, S. (2006). A faster algorithm for calculating hypervolume. *IEEE Trans. Evolutionary Computation* **10**(1), 29–38.

Yu, Y., Zhang, J. B., Cheng, G., Schell, M. C. & Okunieff, P. (2000). Multi-objective optimization in radiotherapy: applications to stereotactic radiosurgery and prostate brachytherapy. *Artificial Intelligence in Medicine* **19**(1), 39–51.

Zitzler, E., Deb, K. & Thiele, L. (2000). Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation* **8**(2), 173–195.

Zitzler, E., Knowles, J. & Thiele, L. (2008a). Quality Assessment of Pareto Set Approximations. In: *Multiobjective Optimization: Interactive and Evolutionary Approaches* (Branke, J., Deb, K., Miettinen, K. & Slowinski, R., eds.). Springer, pp. 373–404.

ZITZLER, E. & KÜNZLI, S. (2004). Indicator-Based Selection in Multiobjective Search. In: *Conference on Parallel Problem Solving from Nature (PPSN VIII)* (YAO, X. *et al.*, eds.), vol. 3242 of *LNCS*. Springer.

ZITZLER, E., LAUMANNS, M. & THIELE, L. (2001). SPEA2: Improving the strength pareto evolutionary algorithm. TIK Report 103, Computer Engineering and Networks Laboratory (TIK), ETH Zurich, Zurich, Switzerland.

ZITZLER, E. & THIELE, L. (1998). Multiobjective Optimization Using Evolutionary Algorithms - A Comparative Case Study. In: *Conference on Parallel Problem Solving from Nature (PPSN V)*. Amsterdam.

ZITZLER, E. & THIELE, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE Trans. Evolutionary Computation* **3**(4), 257–271.

ZITZLER, E., THIELE, L. & BADER, J. (2008b). SPAM: Set Preference Algorithm for Multiobjective Optimization. In: *Conference on Parallel Problem Solving From Nature (PPSN X)* (RUDOLPH, G. *et al.*, eds.), vol. 5199 of *LNCS*. Springer.

ZITZLER, E., THIELE, L., LAUMANNS, M., FONSECA, C. M. & GRUNERT DA FONSECA, V. (2003). Performance Assessment of Multiobjective Optimizers: An Analysis and Review. *IEEE Transactions on Evolutionary Computation* **7**(2), 117–132.