NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF NAVAL ARCHITECTURE AND MARINE ENGINEERING
DIVISION OF MARINE ENGINEERING

Development of Optimization Software Using Evolutionary Algorithms. Applications in Optimal Machine Design.

Diploma Thesis

Nanopoulos Sotiris-Alexandros

Thesis Committee:
Supervisor: Christos I. Papadopoulos, Assistant Professor NTUA
Members: Lambros Kaiktsis, Associate Professor NTUA
Alexandros Gkinis, Associate Professor NTUA

OCTOBER 2016

Acknowledgments

During my five years of studies in the National Technical University of Athens, I have never encountered a project that was of such challenge and of such interest at the same time as this thesis. For that, I can only express my sincere and humble appreciation to Assistant Professor Christos I. Papadopoulos for giving me the opportunity to face such an interesting research topic. His technical insights, innovative ideas and support not only made this thesis possible but also shaped my mind as an engineer.

Secondly, I want to thank the entire research community of this university for their help. During this thesis, I asked for the time and advice of many colleagues of mine from many different departments that gave me their professional opinion.

I would also want to express my gratitude to my family for supporting throughout my studies. Last but not least, I would like to thank my friends who motivate me every day to improve and become a better version of myself.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΝΑΥΠΗΓΩΝ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΝΑΥΤΙΚΗΣ ΜΗΧΑΝΟΛΟΓΙΑΣ

Ανάπτυξη Κώδικα Βελτιστοποίησης με Χρήση Εξελικτικών Αλγορίθμων. Εφαρμογή στην Βελτιστοποίηση Σχεδίασης Στοιχείων Μηχανών.

Διπλωματική εργασία του
Νανόπουλου Σωτήρη-Αλέξανδρου
Επιβλέπων: Χρήστος Ι. Παπαδόπουλος
Επίκουρος Καθηγητής ΕΜΠ

Αθήνα, Οκτώβριος 2016

Περίληψη

Η περιγραφή του κόσμου με χρήση περίπλοκων μοντέλων και προσομοιώσεων γεννά την ανάγκη για ανάπτυξη προγραμμάτων βελτιστοποίησης τα οποία θα μπορούν να βελτιστοποιήσουν τις παραμέτρους των μοντέλων αυτών. Για τη βελτιστοποίηση τέτοιων πολύπλοκων μοντέλων, η μέθοδος με την οποία προκύπτουν τα καλύτερα δυνατά αποτελέσματα είναι αυτή της χρήσης των εξελικτικών αλγορίθμων βελτιστοποίησης. Οι εξελικτικοί αλγόριθμοι, παρότι προσφέρουν, εν γένει, ιδιαίτερα ικανοποιητική εξερεύνηση του πεδίου ορισμού των αντικειμενικών συναρτήσεων, έχουν το μειονέκτημα ότι απαιτούν μεγάλο αριθμό υπολογισμών για να καταλήξουν στη βέλτιστη λύση του προβλήματος.Το μειονέκτημα αυτό είναι τόσο σημαντικό που εμποδίζει την ευρεία χρήση των εξελικτικών αλγορίθμων σε βιομηχανικές εφαρμογές. Ένας άλλος επιστημονικός κλάδος, αυτός της τεχνητής νοημοσύνης, παρέχει εργαλεία στους ερευνητές της βελτιστοποίησης ώστε να μειώσουν το μεγάλο αριθμό των υπολογισμών που απαιτούνται από τους εξελικτικούς αλγορίθμους. Το σημαντικότερο εργαλείο είναι αυτό των νευρωνικών δικτύων, τα οποία χρησιμοποιούνται με διάφορους τρόπους για να επιταχυνθεί η σύγκλιση των εξελικτικών αλγορίθμων.

Στην παρούσα εργασία πραγματοποιήθηκε ανάπτυξη ενός προγράμματος βελτιστοποίησης το οποίο βασίζεται στους εξελικτικούς αλγορίθμους. Για να γίνει το πρόγραμμα αυτό ανταγωνιστικό ως προς τα υπόλοιπα ακαδημαϊκά/εμπορικά προγράμματα βελτιστοποίησης, αναπτύχθηκε στο πλαίσιο της εργασίας αυτής ένας αλγόριθμος για την προεκβολή ενός μετώπου Pareto, ο οποίος επιταχύνει την σύγκλιση των εξελικτικών αλγορίθμων. Ο αλγόριθμος αυτός εκμεταλλεύεται τα νευρωνικά δίκτυα και τους προηγούμενους υπολογισμούς, με σκοπό να μπορέσει να βρει καινούργια άτομα τα οποία υπερισχύουν των προηγουμένων ατόμων του πληθυσμού, με έναν ψευδο-ντετερμινιστικό τρόπο. Οι κλασικές μέθοδοι εξελικτικής βελτιστοποίησης και ο αλγόριθμος προεκβολής αναπτύχθηκαν σε περιβάλλον προγραμματισμού C++. Ταυτόχρονα, με στόχο την αύξηση της χρηστικότητας

του προγράμματος αναπτύχθηκε ένα γραφικό περιβάλλον το οποίο παρέχει τη δυνατότητα στον χρήστη να παρακολουθεί διαγραμματικά σε πραγματικό χρόνο την πορεία της βελτιστοποίησης.

Η αποτελεσματικότητα του προγράμματος βελτιστοπόιησης και του αλγορίθμου προεκβολής ποσοτικοποιήθηκε μέσα από ένα σύνολο εννέα προβλημάτων. Από αυτά, τα πέντε είναι προβλήματα μαθηματικής βελτιστοποίησης, τα οποία έχουν σχεδιαστεί με σκοπό να χρησιμοποιούνται ως αναφορά μεταξύ των διαφόρων αλγόριθμων βελτιστοποίησης. Τα επόμενα τέσσερα είναι προβλήματα βελτιστοποίησης που ανήκουν στον χώρο τον προβλημάτων της βέλτιστης σχεδίασης στοιχείων μηχανών. Πιο συγκεκριμένα, τα τρία επόμενα προβλήματα αφορούν την βέλτιστη σχεδίαση υδροφοβικής επιφάνειας εδράνου ύπο από διαφορετικές συνθήκες λειτουργίας, ενώ το τέταρτο πρόβλημα αφορά την βέλτιστη σχεδίαση της γεωμετρίας ωστικού μικρο-εδράνου με τεχνητή επιφανειακή τραχύτητα.

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF NAVAL ARCHITECTURE AND MARINE ENGINEERING
DIVISION OF MARINE ENGINEERING

Development of Optimization Software Using Evolutionary Algorithms. Applications in Optimal Machine Design.

Diploma Thesis by
Nanopoulos Sotiris-Alexandros

Supervisor: Christos I. Papadopoulos
Assistant Professor NTUA

Athens, October 2016

Abstract

Describing real-world phenomena with complicated models and simulations also gave birth to the need for development of optimization software that would be able to optimize the parameters of such models. To optimize such complicated models, evolutionary algorithms have shown to provide the best performance, in terms of exploration of the search space. However, evolutionary algorithms, in order to solve an optimization problem, generally require a high number of calculations. This weakness is of such importance that prevents them from being used widely as industrial optimization tools. Meanwhile, the development of another scientific field, that of artificial intelligence, provides optimization researchers with new tools to reduce the computational cost of evolutionary algorithms. An important tool builds on the concept of artificial neural networks, which can be utilized in various different ways to accelerate the convergence of evolutionary algorithms.

In the present thesis, a general purpose optimization software using evolutionary algorithms is developed. To make this software competitive in comparison to the existing academic/commercial optimization software, an algorithm has been developed to accelerate the convergence of evolutionary algorithms. This algorithm is based on the notion of extrapolating a given Pareto front. The extrapolation algorithm utilizes previous calculations and artificial neural networks to predict in a quasi-deterministic way new individuals that dominate previously non-dominated solutions. The standard optimization methods and the extrapolation algorithms have developed within the C++ framework. Meanwhile, to increase the usability of the software, a user interface that allows the user to graphically monitor (real time) the optimization procedure has been developed in C++ utilizing the Qt framework.

The efficiency of the developed optimization software and the Pareto extrapolation algorithm has been quantified with the use of a set of benchmarking optimization problems. The optimization problems include five mathematical optimization test cases designed to

benchmark the performance of different approaches on evolutionary algorithms. Afterwards, the performance of the software has been monitored in real world optimization problems that include the optimal design of the hydrophobic surface of a journal bearing and the optimal design of the geometry of a textured three-dimensional micro- thrust bearing.

Table of Contents

Table of figures

Table of tables

Nomenclature

x	*vector* $x=[x_1,x_2,\ldots x_n]^T$
f(x)	*vector* $f(x)=[f_1,f_2,\ldots f_n]^T$
EA	*Evolutionary algorithm*
GA	*Genetic algorithm*
HEA	*Hierarchical evolutionary algorithm*
NSGA-II	*Non Sorting Genetic Algorithm II*
SOP	*Single-objective optimization problem*
MOP	*Multi-objective optimization problem*
SPEA-2	*Strengthen Pareto evolutionary algorithm 2*
MLP	*Multilayer perceptron*
NN	*Neural network*
RBF	*Radial basis function*
PEA	*Pareto extrapolation algorithm*
ev	*every how many generations extrapolations should occur*
ex	*extrapolation factor*
ag	*aggressiveness of the extrapolation procedure*
angle	*angle of the extrapolation procedure*
GUI	*graphical user interface*
k	*Convergence ratio*
l_{uo}	*Non-dimensional non-textured length*
s	*Relative dimple length*

Chapter 1: Introduction

1.1. Introduction

An optimization problem can be defined as a problem with the purpose of finding solutions that fit certain criteria with the most efficient manner. As a result, the first necessary step before solving an optimization problem is creating a function that correlates the solutions with a metric of "how much" a criterion is fitted. The criteria of the optimization problem are modelled as mathematical functions, called objective functions, bounded by the possible solutions called decision variables. There are many ways for solving an optimization problem and it is the responsibility of the solver to pick the right optimization method. Optimization methods can be cluster in many different ways. They can be cluster as analytical or iterative, deterministic or stochastic and population-based or individual-based. The optimization method that is implemented in the present thesis is evolutionary algorithms which belong in the cluster of stochastic-population optimization algorithms. Although evolutionary algorithms can adapt and solve almost any kind of optimization problem the high number of objective calculations required makes them unaffordable for general industrial use.

The outline of this thesis is as follows. In chapter 2, the basic concepts of evolutionary optimization are presented. In chapter 3, artificial neural networks core principle as well as the training procedure is presented. In chapter 4, the most successful algorithms and the algorithm proposed in the present thesis for enhancing the performance of evolutionary algorithm when facing computationally expensive problems are presented. In chapter 5, an overview of the optimization software functionalities, architecture and features are presented. In chapter 6, the optimization software performance tests and the results of those performance tests are presented. Finally, in chapter 7, main conclusions are drawn and some research topics are proposed for further work.

1.2. Literature review

Optimization plays an important role in any scientific or engineering field. As a result, optimization itself as a research field always advances to keep in pace with the scientific needs of its time. Shortly after calculus was established by Isaac Newton and Gottfried Leibniz, the first mathematical optimization concepts arise from Pierre de Fermat. Those concepts are analytical methods for calculating the optima of analytic functions. In particular, Fermat theorem for stationary points can be considered the first mathematical optimization theorem. The scientific needs for optimization methods could not be met only with Fermat theorem, which obviously failed to locate the optima of complex analytical function. The optimization methods developed to satisfy those needs came from the field of numerical calculus, with iterative methods for calculating the optima of analytic functions (Gauss and Newton iterative optimization methods). The next breakthrough in the optimization field was the linear programming method that calculates the optimal values of a linear function that is

linearly constrained. This method was originally proposed by L. V. Kantorovich in his book "Mathematical Methods in the Organization and Planning of Production" (Thie and Keough, 2008). Although expressing real world problems with linear mathematical models with linear constrains was successful, it was not enough, as many engineering and science models describing the world have non-linear components. As a result, optimization had to evolve once again to be able to cover the scientific needs for optimization of non-linear models. Many algorithms have been developed for solving non-linear functions, with the most robust and commonly used is the gradient descent algorithm (Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms, 2007).

All the optimization methods described above, although quite successful, share one or more of the following weaknesses: Firstly, they are suitable for optimizing a single function and thus are unable to optimize multiple functions simultaneously. Secondly, they require the objective function(s) and the constrain(s) of the problem to be differentiable. Finally, there are unable to provide multiple optimal solutions and thus several optimization runs are needed for finding the entire set of the optimal solutions. Those weaknesses become critical in complex engineering problem that require the minimization of multiple non-differentiable, functions that can only be calculated through numerical methods and have a large set of optimal solutions. A class of optimization methods called evolutionary algorithms was developed to tackle optimization problems such as the latter. In particular, evolutionary algorithms suitable for solving optimization problem with multiple goals are called multi-objective optimization evolutionary algorithms. Although the notion of using evolutionary algorithms to solve optimization problems with multiple goals existed since mid-20^{th} century, the first documented multi objective evolutionary algorithm was proposed in 1983 by David Schaffer (Schaffer 1984). In the next decade, several implementations of multi objective evolutionary algorithms were published with the most popular being NSGA (Srinivas and Deb, 1994), NPGA (Horn, Nafpliotis and Goldeberg, 1994) and MOGA (Genetic algorithms in search, optimization, and machine learning, 1989). Lastly, the evolutionary algorithms SPEA-II and NSGA-II (Deb et al., 2002) were developed both in 2002. These algorithms are up to now the core of multi objective evolutionary algorithms.

Multi objective evolutionary algorithms have been one of the most popular optimization methods for high dimensional, complex engineering problems. Although their ability to approximate the Pareto front is widely accepted, their weakness, high number of evaluations, prohibits them from being used as an industrial optimization method. As a result, the scientific research interest is shifted from finding better optimization algorithms for complex engineering problems to enhancing the standard evolutionary algorithms (SPEA-II and NSGA-II). In the recent years, many researchers have focused on finding ways to reduce the number of evaluations need for solving optimization problems with various manners.

- Island models: The idea of using multiple smaller populations instead of a single larger population was proposed by Muhlebein in order to increase the diversification of individuals and avoid premature convergence.
- Metamodel Assisted EA: The notion behind metamodel assisted EA is incorporating knowledge gained from previous exact fitness evaluations to create a function

approximating the fitness function of the problem with low computational cost.

- Hierarchical evolutionary algorithm: The core concept of HEA is organizing the fitness function models hierarchically and using the different models on different areas of the design space. HEA was initially proposed by Herrera et al. in 1999.
- Nash genetic algorithm: For an optimization problem with n objectives a Nash strategy is a symmetric game which consists in having n players, each optimizing its own objective, while respecting the other player criteria. The complement of the methods is achieved when each player is optimizing his objective using a genetic algorithm. The method was originally proposed by Sefrioui and Periaoux in 2000.

In the present thesis, a general purpose optimization software using evolutionary algorithms is developed in order to provide the tools necessary for solving complex optimization problems that come up frequently in the maritime industry. Taking into consideration the limitations and the weakness of the EA, a new algorithm is proposed in order to boost the performance of the standard evolutionary algorithms. This algorithm has been developed and benchmarked in terms of its performance. Finally, the optimization software has been tested in both benchmarking and real world optimization problems to evaluate its robustness.

Chapter 2: Optimization Theory

2.1. Introduction

According to the oxford dictionary (Oxforddictionaries.com, 2016) optimization is defined as "the action of making the best or most effective use of a situation or resource". The notion of effectively using available resources is as old as civilization itself. But until recently the approach to that end was based mostly on hunches and trial and error like processes. In today's world a more effective approach was required in order to deal with the various and very complex problems arising in such areas as engineering, machine design and finance. That approach had to be a quantifiable mathematical process with strict definitions and exact algorithms. These definitions are presented in this chapter as they are considered mandatory for understanding how optimization works. Next, the optimization algorithms used in this thesis, in particular evolutionary algorithms, are analysed in terms of their core elements and how they work. Lastly, one of the most popularly used genetic algorithm, NSGA II, is presented including how its core features enable it to outperform its competitors both in terms of performance and in terms of robustness for complex problems.

2.2. Definitions

In order to develop an understanding of the optimization algorithms that follow certain non-ambiguous definitions are required. These definitions provide all the symbols and tools necessary to understand and analyse the optimization concepts that will follow. Without loss of generality all definitions will be presented in terms of minimization.

Definition 1 (Coello Coello et al., 2007) Single-objective optimization problem

A general single-objective optimization problem is defined as minimizing f(**x**) subject to

$g_i(\mathbf{x}) \leq 0$, i= {1, 2, 3…m} and $h_j(\mathbf{x}) = 0$, j= {1, 2, 3 … p} $x \in \Omega$.

Definition 2 (Coello Coello et al., 2007) Multi-objective optimization problem

A general multi-objective optimization problem is defined as minimizing **f**(**x**) subject to

$g_i(\mathbf{x}) \leq 0$, i= {1, 2, 3…m} and $h_j(\mathbf{x}) = 0$, j= {1, 2, 3 … p} $x \in \Omega$.

Note that in both definition 1 and definition 2 functions g_i, h_j represent the constrains of the problem that must be fulfilled during the minimization of f(x) and **f**(**x**) respectively.

Definition 3 (Coello Coello et al., 2007) Convex Function

A function f(x) is called convex over the domain of $\mathbb{R}$ if for any two vectors $x_1, x_2 \in \mathbb{R}$,

$f(kx_1 + (1-k)x_2) \leq kf(x_1) + (1-k)f(x_2)$ where k is a scalar in the range $0 \leq k \leq 1$

In contrast to single-objective optimization problems, where there is a unique solution, in a non-trivial multi objective optimization problem there is no single solution that simultaneously minimizes all of the objectives. In that case, the objectives are called conflicting and there exist a higher number of optimal solutions. As a result, when solving MOPs, the goal is to find the optimum trade-offs between the objectives.

Definition 4 (Coello Coello et al., 2007) Pareto Optimality

A solution $\mathbf{x} \in \Omega$ is said to be Pareto Optimal with respect to Ω if and only if the is no $\mathbf{x'} \in \Omega$ for which **v=F(x')** dominates **u=F(x)**

Definition 5 (Coello Coello et al., 2007) Pareto Dominance

A vector **u** is said to dominate another vector **v** if and only if **u** is partially less than **v**

Pareto dominance is graphically illustrated in Figure 1:

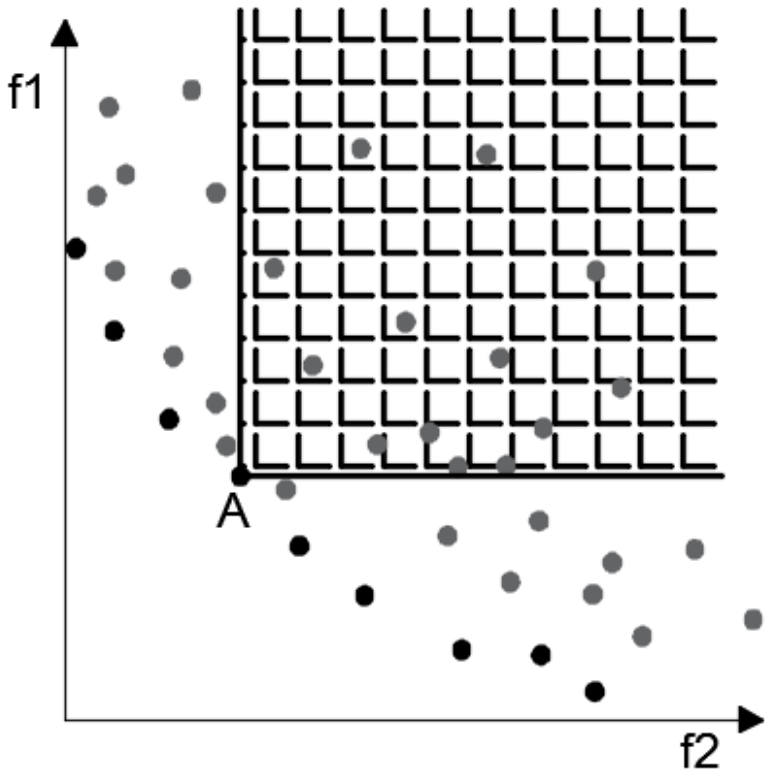

Figure 1: Pareto dominance graphical representation. Solution A is partially less than/dominates any other solution inside the grid

Based on the definitions in the frame of a MOP, comparison of two vectors of variables **v** and **u** can be performed following the relations presented on Table 1.

Table 1: Relation between two decision vectors based on Pareto dominance

Relation	Symbol	Meaning
Strictly Dominates	**v≺≺u**	**v** has less value than **u** in every objective
Dominates	**v≺u**	**v** has less value than **u** in at least one objective while the rest are not higher
Weakly Dominates	**v≼u**	**v** is not higher than **u** in every objective
Incomparable	**v‖u**	Neither **v≼u** nor **u≼v**
Indifferent	**v~u**	Both vectors result in the same objective values

Definition 6 (Coello Coello et al., 2007): Pareto Optimal Set

For a given MOP, **f(x)**, the Pareto Optimal Set, P* is defined as:

P* :={ **x** ∈ Ω|−∃ **x'**∈ Ω **f**(**x'**) dominates **f**(**x**)}

Definition 7 (Coello Coello et al., 2007): Pareto Front

For a given MOP, **f(x)** and Pareto Optimal Set, P*, the Pareto Front is defined as:

PF* = {**u**= **f(x)** | **x** ∈P*}

Based on the definitions above it can be concluded that a multi-objective optimization problem is equivalent to an optimal Pareto front approximation problem and thus any optimization algorithm is an algorithm that has two goals: to minimize the distance to the Pareto front and to maximize the diversity of the solutions. These two objectives are the fundamental goals of any optimization algorithm.

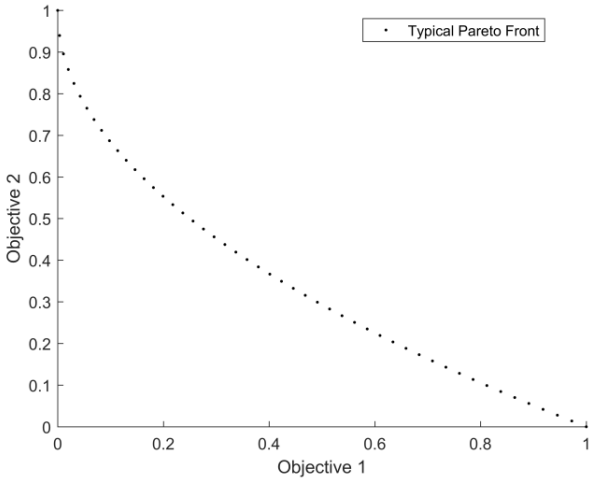

Figure 2: Typical example of a Pareto Front for a two objective minimization problem

Finally, expanding on the Pareto domination definition and the Pareto front definition given above, the relations between Pareto fronts A and B shown in Table 2 have been proposed by Zitzler et al. in 2007. One should note that, since the solution of a multi-objective optimization problem is a Pareto front, the relations shown in Table 2 allow to prematurely compare different solutions of a multi-objective optimization problem. These comparisons are of such necessity that an entire scientific field research focuses on quantifying and comparing Pareto fronts.

Table 2: Table of relations between Pareto Fronts

Relation	Symbol	Meaning
Strictly Dominates	$\mathbf{A \prec\prec B}$	Every $\mathbf{v \in B}$ is strictly dominated by at least one $\mathbf{u \in A}$
Dominates	$\mathbf{A \prec B}$	Every $\mathbf{v \in B}$ is dominated by at least one $\mathbf{u \in A}$
Better	$\mathbf{A \rhd B}$	Every $\mathbf{v \in B}$ is weakly dominated by at least one $\mathbf{u \in A}$ and **A** is different than **B**
Weakly Dominates	$\mathbf{A \preccurlyeq B}$	Every $\mathbf{v \in B}$ is weakly dominated by at least one $\mathbf{u \in A}$
Incomparable	$\mathbf{A \| \| B}$	Neither $\mathbf{A \preccurlyeq B}$ nor $\mathbf{B \preccurlyeq A}$
Indifferent	$\mathbf{A \sim B}$	$\mathbf{A \preccurlyeq B}$ and $\mathbf{B \preccurlyeq A}$

2.3. Genetic Algorithms

One genetic biologist once said “Evolution is an optimization process” (Mayr 1988). Evolution is the nature’s way to discover high precision solutions to complex problems using natural selection and mutation and thus slowly optimizing the gene pool in every generation. The basic concepts of evolution inspired many researchers in the field of computer science and computational engineering to create adaptive artificial algorithms, which mimic the evolution, to solve complex problems. The act of creating algorithms that mimic the nature created an entire new subset of algorithms called evolutionary algorithms that belong to the field of evolutionary computation. The field of evolutionary computation had huge success in the optimization field with the development of the evolutionary optimization algorithms. Although, according to the no-free-lunch (NFL) theorem (Wolpert and Macready 1996), there cannot exist an optimization algorithm that can solve all optimization problems superior to any other optimization algorithm, algorithms that mimic the natural selection outperform their competitors (e.g. gradient descent, Hill climbing , random search algorithm) in complex engineering problem described by lack of linearity and chaotic components.

2.3.1Genetic algorithm representation

Although there is no generally accepted definition for genetic algorithms one can define them as adaptive search algorithms inspired by the biological ideas of natural selection and genetics. As such, they represent an artificial intelligent variation of a random search algorithm used to solve optimization problems. The primary reason for using such algorithms is their ability to find a high number of Pareto optimal solutions in one simulation run. Finally, in order to analyse how an evolutionary algorithms work it is critical to understand the following core elements of an evolutionary algorithm: population of individuals, selection, crossover and random mutation of new offspring.

An individual (or chromosome) of the population is a set of values (decision vector) that define a proposed solution to the optimization problem an evolutionary algorithm strives to solve. The individual is represented through a variety of ways with the most common being binary representation, string representation and real code representation. A set of individuals with size $k \in \{2,3,\ldots n\}$ is called the population of the evolutionary algorithm. Generally, algorithms that use populations are also called population-based algorithms.

The selection operator mimics the natural process of mating. This operator is responsible for selecting individuals based on their fitness value. The higher the fitness of an individual the higher the chance it has of reproducing. The fitness value of an individual is calculated by a specific function called “fitness function”, which maps the individuals with their respective fitness values.

The crossover operator mimics the natural process of reproducing. Let A and B be two parent individuals in the population, then the crossover operator creates the offspring of A and B with a value that arises from how the operator is being implemented. The most common crossover types are single and two point crossover, which are presented in Figure 3 and Figure 4.

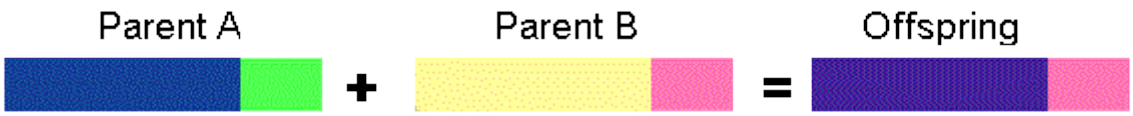

Figure 3: Single Point Crossover Operator

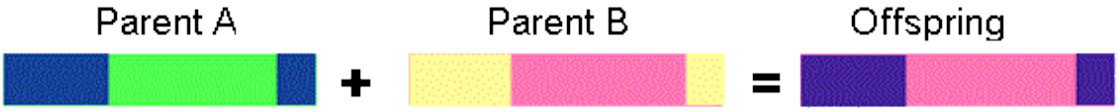

Figure 4: Double Point Crossover Operator

The mutation operator is a genetic operator used to maintain genetic diversity from one generation of population to the next and it is analogous to biological mutation After the crossover operator the offspring are subjected to the mutation operator which alters one or more gene values in an individual from their initial state according to a given mutation probability. The purpose of mutation in an evolutionary algorithm is to preserve and introduce diversity. Mutation allows the algorithm to avoid local minima by preventing the population from becoming too similar with each other and thus stopping the evolution. On the other hand, a high mutation probability has the undesirable effect of transforming the algorithm into a random search algorithm.

Knowing all the basic elements of genetic algorithm, the genetic algorithm and the genetic algorithm flowchart is shown below:

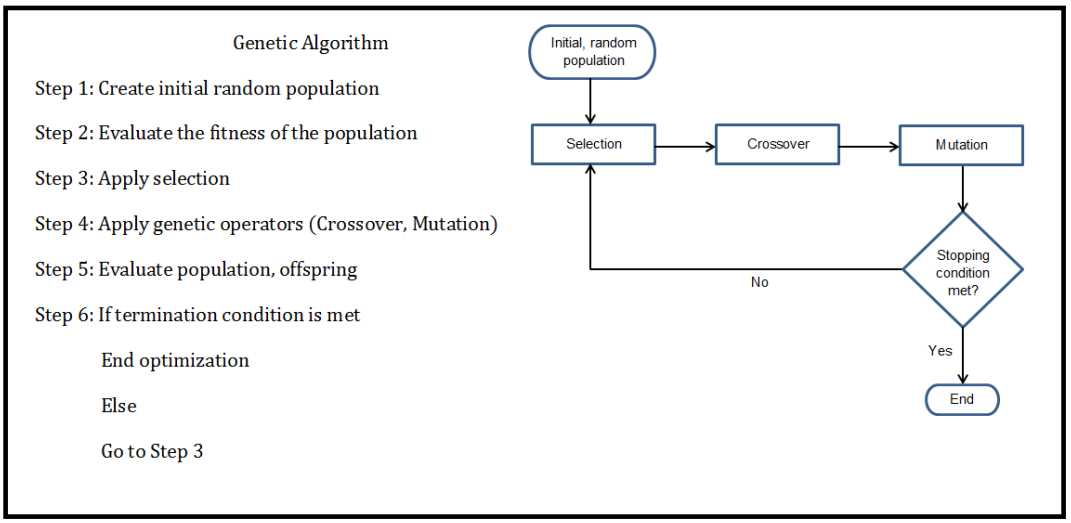

Figure 5: Genetic algorithm flowchart

2.4. Non dominated sorting Genetic algorithm II (NSGA II)

Given the fact that genetic algorithms outperform their competitor algorithms in optimization problems, when there is no previous knowledge of the problem, a high number of genetic algorithms have been suggested. In 2002 , the non-dominated sorting genetic algorithm II (Deb et al., 2002), NSGA II, was suggested featuring the "fast non dominating sorting approach" algorithm responsible for reducing the computational complexity of the sorting and the crowding distance assignment algorithm responsible for keeping high diversity of the solutions. As a result, NSGA II outperformed its competitors to become one of the most widely used optimization algorithms up to now, along with the strengthen pareto evolutionary algorithm 2 (SPEA 2).

NSGA II was proposed to solve the problems his predecessor non dominated sorting genetic algorithm (NSGA) (Srinivas and Deb, 1994).

The criticism NSGA faces is the following:

1) The use of suboptimal non dominated sorting algorithm. The computational complexity of NSGA is $O(MN^3)$ where M is the number of the objectives of the optimization problem and N is the size of the population used. This complexity makes the algorithm insufficient for any high complexity problem that involves a large population, and as a result NSGA met with little commercial success.
2) The absence of elitism during selection. Elitism in genetic algorithms is a feature that acts in such a way as to retain a number of the best individuals of the population after each generation. Elitism makes sure that a good proportion of the best individuals will reproduce and will not be destroyed from the crossover and mutation operators. It has been shown that elitism is a crucial component for improving the performance of a genetic algorithm (Zitzler, Deb and Thiele, 2000), since it ensures that good solutions will not be accidentally destroyed by the algorithm.
3) Lastly, NSGA utilized a subpar algorithm for keeping diverse solutions. As it was mentioned in the previous chapter, one of the main objectives in any multi-objective optimization algorithm is to find a highly diverse set of solutions approximating the Pareto front. NSGA, in order to keep diversity, used a sharing parameter that required previous knowledge of the optimization problem and there was no way to either know the parameter value beforehand or adjust its value at runtime.

Before presenting the fast non dominated sorting algorithm, a simpler suboptimal approach will be presented for the reader to become familiar with the concept and get a better grasp of the fast sorting algorithm. In order to find the non-dominated solutions in a population size N, each individual of the population must be compared with every other individual to examine whether or not it is dominated. This comparison requires each individual to be compared with another individual a number of times equal to the number of objectives of the optimization problem. This procedure needs to be iterated a number of times equal to the size of the population. As a result, the computational cost for each individual is O(MN) where M is the number of objectives. If this procedure is repeated

for the entire population it results in a computational cost of $O(MN^2)$ for finding the non-dominated solutions in the population. Finally, in the worst case scenario, there exist N fronts with only one solution in each front, therefore the algorithm described above has a computational cost of $O(MN^3)$ and a memory requirement of O(N) (Deb et al., 2002). In order to achieve lower computational complexity in sorting non-dominated solutions, the fast non dominated sort algorithm was proposed (Deb et al., 2002) as follows:

Fast Non Dominated sort (P)

For each p ∈P

S_p:=0

n_p:=0

for each q ∈P

if p dominates q then

S_p:= S_p U {q} (add q to the set of solutions dominated by p)

else if q dominates p then

n_p := n_p +1 (Increment the domination counter of p)

if n_p =0 then (p belongs to the first front)

p_{rank}:=1

F_1:=F_1 U {p}

i=1 (front counter)

while F_i ≠0

Q:=0

For each p ∈F_i

For each q ∈ S_p

n_q:=n_q-1

if n_q =0 then (q belongs to next front)

q_{rank}:=i+1

Q:=Q U {q}

i:=i+1

F_i=Q

Figure 6: Fast non-dominated sorting algorithm (Deb et al., 2002)

The algorithm above offers computational complexity of $O(MN^2)$, which arises from the following calculations: The first loop is calculated exactly N times since each individual can be the member of at most one front while the second nested loop (for each q) is executed at most (N-1) times for each individual. Therefore the calculated computational complexity of the algorithm is reduced to $O(MN^2)$ while the memory requirement is increased to $O(N^2)$.

In order to keep the solutions diverse NSGA II uses the crowding distance assignment algorithm. Before presenting the algorithm a definition must be given. Distance is defined as the absolute normalized difference in the function values of two adjacent solutions for

intermediate solutions, while the distance is defined as infinite for the boundary solutions. The crowding distance assignment algorithm (Deb et al., 2002) is shown in Figure 7:

Crowding distance assignment (I)

a := |I|

for each I, set I[i]:=0

for each objective m

I=sort(I,m)

I[a]:=∞ ,I[1]=∞

for i=2 to (a-1)

$I[i]_{distance} := I[i]_{distance} + (I[i+1].m - I[i-1].m)/(f_m^{max} - f_m^{min})$

Figure 7: Crowding distance assignment algorithm (Deb et al., 2002)

In the algorithm above, I[i].m stands for to the m^{th} objective function of the individual i.

The last core element of NSGA II is the crowded comparison operator which is a selection operator responsible for keeping the Pareto front evenly distributed. The operator can be defined as:

Let every individual of the population characterized by the two following attributes: nondomination rank and crowding distance. Then partial order $\prec_n$ can be defined as

$i \prec_n j$ if ($i_{rank} < j_{rank}$)

or ($i_{rank}=j_{rank}$)

and ($i_{distance}>j_{distance}$)

After all of the above algorithms were discussed, the NSGA II algorithm is presented:

NSGA II

$R_t := P_t \cup Q_t$ combine parent and offspring population

F=fast-non-dominated-sort(R_t) F represent all the non-dominated fronts of R_t

P_{t+1} :=Ø and i:=1

Until $|P_{t+1}| + |F_i| \leq N$ add elements to the population until it is filled

Crowding-distance-assignment(F_i) calculate crowding distance of F_i

$P_{t+1}:=P_{t+1} \cup F_i$ include ith nondominated front in the parent population

i:=i+1

Sort($F_i, \prec_n$) sort in terms of crowded comparison operator

$P_{t+1}:=P_{t+1} \cup F_i[1:(N-|P_{t+1}|)]$ choose the first $N-|P_{t+1}|$ elements of F_i

Q_{t+1}=make-new-pop(P_{t+1}) Use genetic operators to create new population

t=t+1 increment the generation counter

Figure 8: NSGA II algorithm (Deb et al., 2002)

Chapter 3: Machine Learning

3.1. Introduction

With the advent of the digital era (say early 80's) the amount of data stored has grown exponentially. The rise in the amount of data created the need to abandon traditional data analysis and create new algorithms that have the ability to learn from the available data on the fly and draw conclusions from them. This phenomenon also occurs when solving an optimization problem. All the previous calculations made from the optimization algorithm are a huge amount of data available that can be incorporated using appropriate algorithms to accelerate the optimization procedure. In order to solve these problems, another cluster of algorithms that mimic the biological procedures, and more specifically the human brain, arise. Those algorithms are called artificial neural networks. Artificial neural networks mimic the way the neural system of the human brain transmits and processes input signals. In this chapter two of the most commonly machine learning algorithms used in optimization problems: multilayer perceptron feedforward artificial neural networks, will be presented

3.2. Artificial Neural Network

3.2.1. Multilayer Perceptron Feedforward Artificial Neural Network

One of the most important of all the implementations of the neural networks is the multilayer perceptron feedforward artificial neural network (MLP). Firstly, a classification of the problems that machine learning in general is solving will be presented. In short, problems can be classified as supervised or unsupervised learning problems, based on the type of data supplied for solving the problem, and as classification or regression problems, based on the function neural networks are trying to produce. Furthermore, MLPs will be presented both in terms of their architecture and in terms of their core principle. Finally, the core dilemma of neural network interpolation will be shown.

Machine learning problems can be classified in two main categories: supervised learning and unsupervised learning problems. Supervised learning is the task of creating a function that corresponds **x** to **y** based on a dataset that contains m number of examples (also referred as training examples) of corresponding values of **x** and **y**. For example given a dataset of points (x, y), performing linear regression in the dataset is a prime example of supervised learning problem. On the other hand, unsupervised learning is the task of clustering the data of a large dataset. For example, given the news of the day, the act of grouping the news in terms of their content is an example of an unsupervised learning problem.

It has been already mentioned that supervised learning is the task of learning the mapping between **x** and **y**. Supervised learning problems can be split to two categories: classification and regression problems. In classification problems, the goal is to create a decision function based on a data set of **x** and **y** where **y= {0 or 1}** that estimates the probability of a vector **x**’

to result in a vector **y**'=1. The most common example of a classification problem is as follows: Given a set of emails classified as either spam or not spam, to create a decision function that, given a new email calculates the probability of this mail being a spam. On the other hand, in regression problems, the objective is to create a function based on a data set of **x** and **y**, that given, a vector **x**', the function corresponds a value of **y**'. For example, given house's size in square meters and the price they were sold, the regression problem is the problem of creating a function that given a new house's size in square meters to predict the price it will be sold. In the present thesis, all the problems being solved using machine learning techniques are supervised learning regression problems.

In order to understand the working principles of MLPs, one must first understand how the core structural element of MLPs (the neuron) works. An artificial neuron receives a number of numerical input signals(**x**) and then, with a set of parameters, transforms the input signal to an output signal (y). The transformation process is presented both in graphical and analytical representation. (Figure 9)

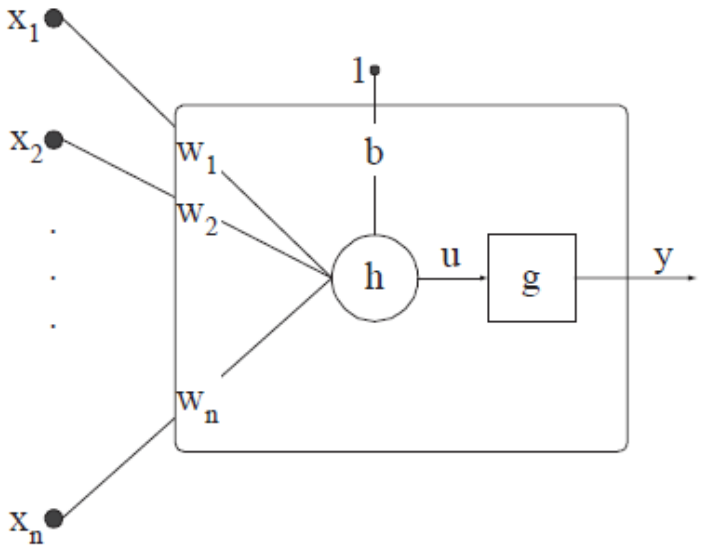

Figure 9: Neuron representation (Gonzalez 2008)

$$y = g(\boldsymbol{x} \cdot \boldsymbol{w}^{\boldsymbol{T}} + b)$$

Here g is the activation function, **w** is the weight vector and b is the bias unit.

An MLP is an algorithm for solving supervised learning problems. The structural components of MLP architecture are: the number of input, the input layer, the number of hidden layers, the number of neurons in each hidden layer, the number of output and the output layer. An MLP representation is presented in Figure 10:

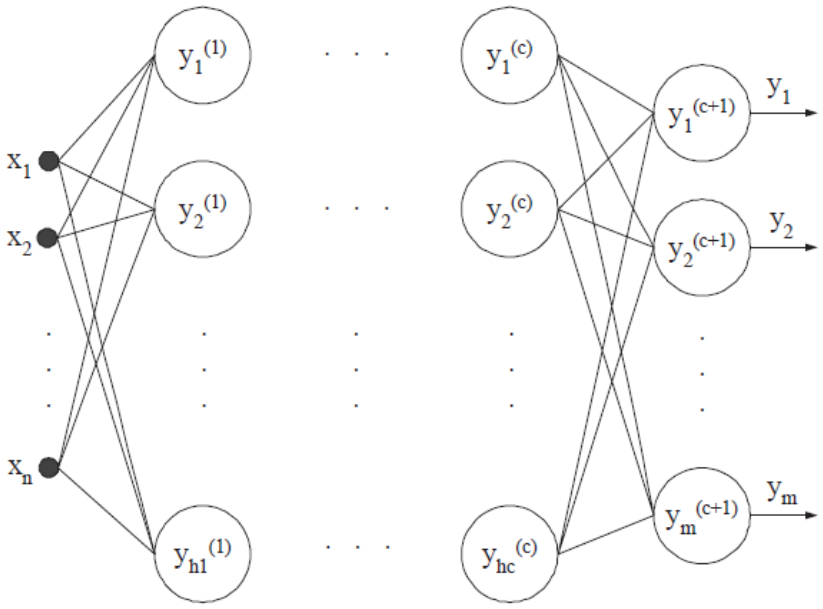

Figure 10: MLP architecture diagram (Gonzalez, 2008)

In Figure 10, n represents the number of input signals, the large black dots represent the input layer, circles represent neurons, number c represents the number of hidden layers in the MLP and lastly the number m represents the number of outputs of the neural network. Also, it is important to observe that (a) MLPs are fully connected, meaning that every neuron from a layer k connects with every neuron in layer k+1, and (b) MLPs are feed forward neural networks meaning that no neuron in a layer k can feed a neuron in a previous layer. Summing up the above, an MLP creates the following network function:

$$y_i^{(k)} = f\left[\sum_j^N g(w_{ij}^{(k,k-1)} y_j^{k-1}) + b_i^k\right]$$

where f is the activation function of the output layer and g is the activation function of the hidden layer.

The last core concept of MLPs is the training of the neural network. Training can be defined as the minimization of the error between the value of the approximation function created from the MLP and the actual value for every training example in the dataset with the decision variables being the weights of the neurons and the bias units. The error function is commonly known as cost function. The most common error functions are the logistic regression and the mean square error function. However, minimizing the error function is not a necessary condition for having a good approximation function. The approximation function might predict perfectly the examples in the training set, but fails to give good predictions in examples outside the training set. The problem described above is called overfitting the data.

3.2.2. Training algorithm

As it was mentioned before, training a neural network is an act of solving the following optimization problem:

Minimize: $C = error[(f(\boldsymbol{x}|\boldsymbol{w}) - \boldsymbol{y})^2]$ where C is the cost function to be minimized, error is the error function and f($\boldsymbol{x}|\boldsymbol{w}$) is the approximation function produced by the neural network bounded by the decision variable **w**

This optimization problem is solved using the backpropagation algorithm to calculate the values needed to perform the gradient descent optimization algorithm (Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms, 2007).

Backpropagation algorithm is not a single algorithm but a cluster of algorithms used to perform the training of a neural network. Although various implementations of the backpropagation algorithm may differ from one another, they all have the same core concepts. Those concepts are the following:

Any backpropagation algorithm can be divided in to two phases:

(a) The propagation phase, which consists of forward propagating the training examples in order to calculate the output of all the activation units
(b) The backpropagation phase, where using the calculated values of all the activation units the difference between the predicted value and the actual value is calculated.

Finally, the weights are updated according to the following. Firstly, the gradient of the error in terms of the weight is calculated. Then the error and the gradient of error in terms of the weight are used within a gradient descent optimization algorithm, which updates the gradient of the weight values by subtracting them a given ration. This procedure is called iteration or epoch and is repeated until the stopping criterion, most common criterions being the number of epochs or a desirable error value. The percentage of the gradient of the weight subtracted is called the learning rate and it expresses how much the algorithm “trusts” each training example. Thus, the greater the learning rate, the faster the training of the network but with downside that the predictions function might be less accurate. In most cases the weight values are randomly initialized between [-1,1] at the beginning of the optimization procedure.

One key limitation of the gradient descent algorithm is that it is not guaranteed to find the global minimum of the cost function but a local minimum and thus having a limited success in non-convex cost functions. This limitation arises from the gradient descent algorithm. In most engineering problems though, local minima are good enough solutions for most purposes. Lastly, although not mandatory, data normalization increases the performance of the algorithm.

The other three backpropagation algorithms that will be used in this thesis are batch backpropagation, quickprop and rprop. Batch backpropagation is the same as the standard

backpropagation algorithm but the weights are updated after calculating the error of the whole training set thus resulting in a slower algorithm that calculates more accurately the error function and as a result offers better results in many applications. The rprop algorithm (Riedmiller and Heinrich, 1992) is adapting the learning rate based on the partial derivatives of the weights in every dimension. Lastly the quickprop learning algorithm (Fahlman, 1988) promises significantly better results than standard backpropagation by modifying the update rule in such a way that the update takes into consideration the value of the weight at the previous iteration.

3.2.2. Activation Functions

The different activation functions of the neurons change not only the approximation function created by the neural network but also the error function used during training. The activation function itself affects the approximation function model while the derivative of the activation function is the deciding factor for the convexity of the cost function. In Table 3: Table of activation functions the activation functions that will be used in this thesis for both the hidden layer and the output layer are presented.

Table 3: Table of activation functions

Function name	Equation
Sigmoid	$f(x) = \frac{1}{1 + e^{-2sx}}$
Sigmoid symmetric	$f(x) = tanh(sx)$
Gaussian	$f(x) = e^{-x^2 s^2}$
Elliot	$f(x) = \frac{\frac{xs}{2}}{1 + \lvert xs \rvert} + 0.5$
Sin	$\mathrm{f(x)} = \frac{\sin(\mathrm{xs})}{2} + 0.5$
Cos	$f(x) = \frac{cos(xs)}{2} + 0.5$

Chapter 4: Optimization for Computationally Expensive Problems

4.1. Introduction

Evolutionary algorithms have been one of the most popular optimization methods for high dimension, complex engineering problems. Their ability to approximate the Pareto front with a single run is widely accepted. Their weakness, which is the high number of evaluations needed to achieve the abovementioned Pareto front approximation, may result in certain cases in unacceptable computational cost. This computational cost prohibits EA from being used as an industrial optimization method. The following definition (Tenne et al. 2010) was given for computationally expensive problems:

A problem will be considered computationally expensive when it is described by the following attributes:

1) The objective function is computationally expensive in terms of time or in terms of CPU usage (i.e. CFD problems may require many hours to be solved)
2) There can be limited amount of objective calculations (i.e. optimization for F1 where CFD solvers can be used by the manufacturing companies a limited number of times due to regulations)

In last years, many researchers focus on finding ways to reduce the number of evaluations needed for solving optimization problems. The most commonly used methods that will be described below are surrogate models and fitness inheritance. Lastly, in this thesis a new approach is proposed where, based on the previous calculations, an artificial neural network is trained in order to predict solutions that would in a quasi-deterministic way, dominate previous solutions. All the methods mentioned above although different from one another, share the same concept of incorporating knowledge gained from previous calculations made during optimization and using it in either deterministic or stochastic algorithms to accelerate the converge towards the Pareto front.

4.2. Surrogate Models

In many cases, in real world optimization problems, the fitness function does not come in analytic form but as a result of a simulation or a finite element solution and is considered a “black box” for the optimization procedure. These objective functions may take hours to be solved for a single individual, let alone for the entire population. The surrogate model (or metamodel) approach is used to create an approximation function which can be solved with relatively low computational cost. The approximation function of the objective function is created with machine learning techniques. The objective of this approach is to create a metamodel of the objective function that can reduce the number of evaluations needed while

maintaining the quality of the results.

Metamodels ability to approximate the objective function well with the minimum amount of training relies heavily on the type of neural network and the training set used. In terms of the training, metamodel assisted evolutionary algorithms can be separated in to two classes. In the first class of metamodel assisted evolutionary algorithms, a metamodels is trained in advance of the optimization procedure when the sufficient data are gathered. This metamodel is then used during optimization procedure to approximate the objective function. The metamodel is constantly monitored in terms of its associated error; the metamodel is updated if the associated error is unacceptable. The second class is based on 'on-line training' meaning that the metamodel is constantly retrained based on the most recent data available. In the second class of metamodel assisted optimization although the metamodel might be more accurate the computational cost of training is getting significantly higher since the metamodel has to be retrained a high number of times and thus being the determining factor for choosing the neural network type.

One of the most common approximation methods is using a second order polynomial model with the following form:

$$\boldsymbol{y} = b_o + \sum b_i x_i + \sum b_{n-1+i+j} x_i x_j$$

The polynomial model is trained using the previous calculations using the least square cost function and the gradient descent algorithm. One notable application of polynomial models as surrogates can be found in (Goel et al., 2007) where polynomial models assisted the standard NSGA-II to approximate the Pareto front of a liquid-rocket engine design problem

Another commonly used approximation algorithm is Kriging models which is a global model plus a deviation term:

$$y(\boldsymbol{x}) = g(\boldsymbol{x}) + Z(\boldsymbol{x})$$

where g is the global model and Z is the deviation term. The deviation term is modelled as Gaussian random function with zero mean and non-zero covariance that represent the local deviation of the global moment. The main advantage of Kriging models is the low computational cost of the model. However, due to the fact the training for Kriging model is done with deterministic methods (requiring matrix inversions) rather than optimizing a cost function with the gradient descent algorithm, the computational cost of the algorithm increases when the dimensions of the problem become too high.

Another, commonly used approximation model used is Radial Basis function Networks and their normalized form kernel ridge regression:

RBF Neural Network Function: $y(\boldsymbol{x}) = \sum_{j=1}^{n} w_j \, \varphi(\|\boldsymbol{x} - \mu_j\|)$

RBF regularized Neural Network Function (Kernel ridge regression): $y(\boldsymbol{x}) = \frac{\sum_{j=1}^{n} w_j \varphi(\|\boldsymbol{x}-\mu_j\|)}{\sum_{j=1}^{n} \varphi(\|\boldsymbol{x}-\mu_j\|)}$

RBF neural networks either alone or accompanied with self-organizing maps for calculating the centers have shown good results in assisting evolutionary optimization (Karakassis and Giannakoglou, 2006).

Finally, MLPs is also a really good approach for approximation model:

$$y_i^{(k)} = f\left[\sum_{j}^{N} g(w_{ij}^{(k,k-1)} y_j^{k-1}) + b_i^k\right]$$

Their ability to approximate generic functions along with their simplicity makes them a compelling method for approximating complex engineering problems. For example, Poloni et al., 2000 used MPLs as an approximation model for 3D Navier-Stokes simulation of the fin keel for the optimization of the design of a sailing yacht.

4.3. Fitness Inheritance

Another approximation method developed for speeding up the optimization procedure is "fitness inheritance" where, instead of approximating the objective function of the problem, the fitness value of every individual is assigned to them based on their parents fitness values. Fitness inheritance was originally proposed at 1995 and had quite a success since then. In fitness inheritance, all the individuals in the initial population have their fitness value calculated with the use of the fitness function. Thereafter, only for a limited number of individuals their fitness value is calculated using the exact functions and for the rest their fitness value is assigned with the fitness inheritance algorithm.

The classic implementation of fitness inheritance proposed by Smith in 1995 has been successful in accelerating the convergence of evolutionary algorithms. The classic fitness inheritance equation is the following:

$$f(\boldsymbol{x}^h) = \begin{cases} f(\boldsymbol{x}^p)\,, if\ d(x^h, x^p) = 0 \\ \dfrac{s(x^h, x^{p1}) f(x^{p1}) + s(x^h, x^{p2}) f(x^{p2})}{s(x^h, x^{p1}) f(x^{p1}) + s(x^h, x^{p2}) f(x^{p2})}, otherwise \end{cases}$$

Where x^h is the generated individual from x^{p1} and x^{p2}. The equation above describes the fitness value of an offspring as the weighted average of his parent's fitness values. The similarity function, s, is a metric highly correlating with the Euclidian distance of the offspring with their parents.

An advanced implementation of this algorithm is clustering individuals into several groups and for each cluster of solutions only calculating one representative individual and assign fitness values for the rest based on the classic fitness inheritance equation described above

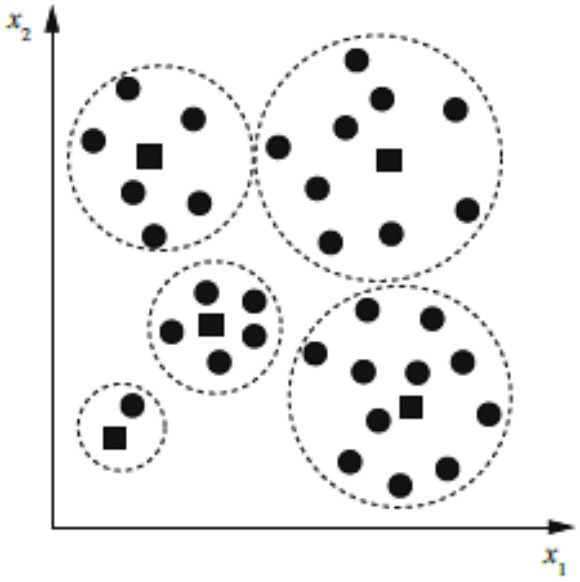

Figure 11: Example of fitness inheritance with clustering where the black circles represent individuals whose fitness has been assigned with a fitness inheritance equation while the black boxes represent individuals whose fitness has been exactly calculated. (Fonseca, Barbosa and Lemonge, 2009)

4.3. Pareto Extrapolation Algorithm

In the present thesis, the Pareto extrapolation algorithm is being proposed for accelerating the optimization procedure. The fundamental idea of this algorithm is to create a procedure that can seed into the population individuals that would in a quasi-deterministic way dominate previously non-dominated individuals. The aforementioned idea is implemented through extrapolating a given Pareto front of an optimization problem. Then based on the knowledge incorporated from previous calculations estimate individuals that correspond to the extrapolated points and seed them into the population. The Pareto extrapolation algorithm will be described below. Finally, in a later chapter the algorithm will be plugged in to the standard NSGA-II and will be tested against it in terms of performance.

In order to predict the individuals that would result in the extrapolated points the reverse function of the fitness function has to be approximated using MLPs. The first major problem that has to be overcome is the limitation of neural networks in general to produce a function with less neuron in the input layer than the output layer. This problem was solved by considering as constants all the variables up until the sum of the number of objectives plus the number of constant variables is at least equal or one more than the number of the rest of the variables. Finally, in order to reduce the error, only the calculations made during the previous generation were used. Regarding the training, all the activation functions mentioned above for the hidden and the output layer and all the training algorithms will be tested for a limited number of training iterations and the ones resulting in the least error will be used. That, in order to keep the algorithm as less biased as possible. Afterwards, after a generation has been completed and following the extrapolation protocol suggested below, the Pareto front will be extrapolated. The extrapolated points, combined with the constant values coming from the individuals, will then be used as an input for the neural network function to

estimate the individuals that correspond to the extrapolated points. Finally, those individuals will be merged with the original population and will be evaluated in order to replace the members of the population they dominate.

The algorithm will be shown below in a step-by-step format for clarity:

Step 1[Initialization of a standard optimization problem]: Let the following optimization problem be solved with NSGA-II

$$\begin{cases} \text{minimize } (f_1, f_2, \dots f_n) \text{ where n is the number of objectives} \\ \text{Bound to the desicion vector} : \mathbf{x} := [x_1, x_2, .., x_m] \text{where m is the number of variables} \end{cases}$$

With the following user defined parameters for the Pareto extrapolation algorithm:

1) After how many generations the extrapolations occur, denoted: *ev*
2) The extrapolation factor, denoted: *ex*
3) The aggressiveness of the extrapolation, denoted: *ag*
4) The angle of the extrapolation, denoted: *angle*

Step 2 [Selection of training data]: If the difference between the current generation number and the generation number where the next Pareto extrapolation algorithm will occur is 5 or less and the point is not already added to the training database then add the following vectors as a training example:

$X := [f_1, f_2, \dots f_n, x_1, x_2 \dots, x_k]$, $Y := [x_k x_{k+1} \dots, x_m]$ where k is select in such manner that either the size of vector X and vector Y are equal size or the size of vector X is greater than vector Y size by one

Step 3 [The training procedure]: At the end of every generation where the Pareto extrapolation algorithm should occur, an MLP is trained in the following manner to ensure the minimum error in the cost function. Firstly, a neural network with a single neuron in the hidden layer is being trained with the data created in step 2 with all the available training algorithms (standard backpropagation, batch backpropagation, quickprop, rprop) for 2000 epochs and the algorithm that results in the minimum value for the cost function is selected as the training algorithm. Afterwards, for the selected training algorithm all possible combination of activation functions for both hidden layer and output layer are tested again with the same procedure and again the combination of activation functions resulting in the minimum value for the cost function are selected.

Step 4 [The extrapolation protocol]: Let the current Pareto front and the individuals corresponding to that Pareto front after the end of a generations to be the following matrixes:

$$F := \begin{bmatrix} y_{11} & \cdots & y_{1n} \\ \vdots & \ddots & \vdots \\ y_{g1} & \cdots & y_{gn} \end{bmatrix} \qquad X := \begin{bmatrix} x_{11} & \cdots & x_{1m} \\ \vdots & \ddots & \vdots \\ x_{g1} & \cdots & x_{gm} \end{bmatrix}$$

Where F is the Pareto Front matrix, X is the individuals corresponding to the Pareto Front matrix and g is the number of non-dominated solutions. Then depending on the size, the following extrapolation sub protocol is followed:

Step 4a: The non-dominated solutions with an index contained in the following set will be extrapolated:

$$i_{n+1} := \begin{cases} i_n + 1, \ if\ ag + (times\ PEA\ occured - 1) > g \\ i_n + div\left(\frac{g}{ag + (times\ PEA\ occured - 1)}\right)\ in\ other\ cases \end{cases} \text{ and } i_0 = 0$$

Where n:={0,1,…g}

Step 4b: The extrapolation factor will be calculated for each objective with the following equation:

$$extrapolation_i := (ex + (times\ PEA\ occured - 1) \cdot 0.01) \cdot \sin(angle + \frac{\pi}{2} \cdot i)$$

Where i:={1,2,…n}

Finally, for each extrapolated point we create the following vector containing the extrapolated points and the corresponding individual decision variables:

$$F_i := [extrapolation_1 \cdot f_1, extrapolation_2 \cdot f_2, \ldots extrapolation_n \cdot f_n, x_1, x_2 \ldots, x_k]$$

Step 5 [Reversing the problem]: For every vector F_i mentioned above, using the MLP function the following vector is calculated:

$$F_i := [\text{extrapolation}_1 \cdot f_1, \text{extrapolation}_2 \cdot f_2, \ldots \text{extrapolation}_n \cdot f_n, x_1, x_2 \ldots, x_k]$$
$$\overset{NN}{\Rightarrow} X_i' := [x'_k x'_{k+1} \ldots, x'_m]$$

The set of vectors X_i' will be combined with the vector, $X_i := [x_k x_{k+1} \ldots, x_m]$ to create a set of individuals X= [$X_i' | X_i$] which will be merged and evaluated with the original population in terms of their fitness value and thus replace dominated solutions kept in the population with individuals that correspond to new Pareto dominant solutions.

Step 6 [Repeat]: If the termination condition hasn't been met then go to step 2

The algorithm flow described above is illustrated in Figure 12 below:

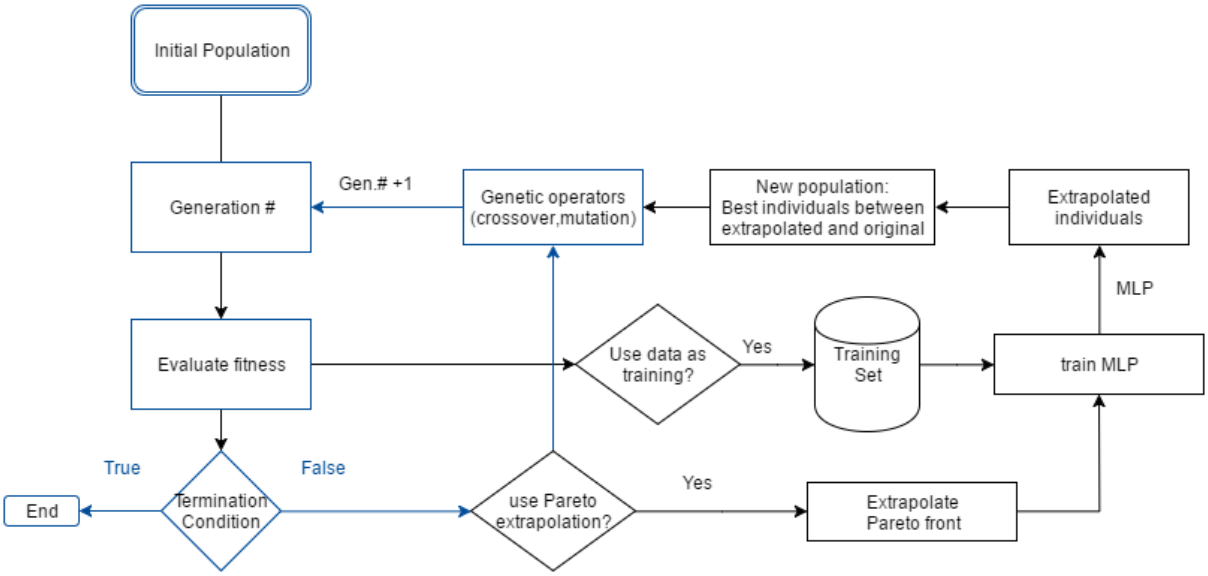

Figure 12: EA-assisted with Pareto extrapolation algorithm. The standard EA is denoted with blue colour while the Pareto extrapolation algorithm is denoted with black colour

The algorithms shown above although tested as a plug-in to NSGA-II can be incorporated in any population based optimization approach not only to accelerate the optimization procedure but also increase its robustness. The Pareto extrapolation algorithm will be tested in terms of its performance both in analytic optimization functions as well as real world applications in the following sections.

Chapter 5: Optimization Software Overview

5.1. Introduction

The optimization software developed in the scope of this thesis consists of two major components. The first component, “optimizer GUI” is written in C++ in the Qt framework and its task is the user's interface of the software. “Optimizer GUI” subtasks include setting up the user defined parameters, creating the optimization database and visualizing the optimization procedure with proper graphs. The second component of the optimization software: “optimizer core” is developed in plain C++ for performance reasons. The optimization procedure is initialized after the user defines a set of parameters necessary to run the algorithms mentioned above. After the user settings have been defined, a local database is created in order to store both the parameters and all the calculations made by the “Optimizer Core”. Simultaneously, while “optimizer core” is implementing the standard NSGA-II and the Pareto extrapolation algorithm, the database is updated with the calculations and “optimizer GUI” using the available data graphically represents the optimization progress.

All of the above are shown in Figure 13:

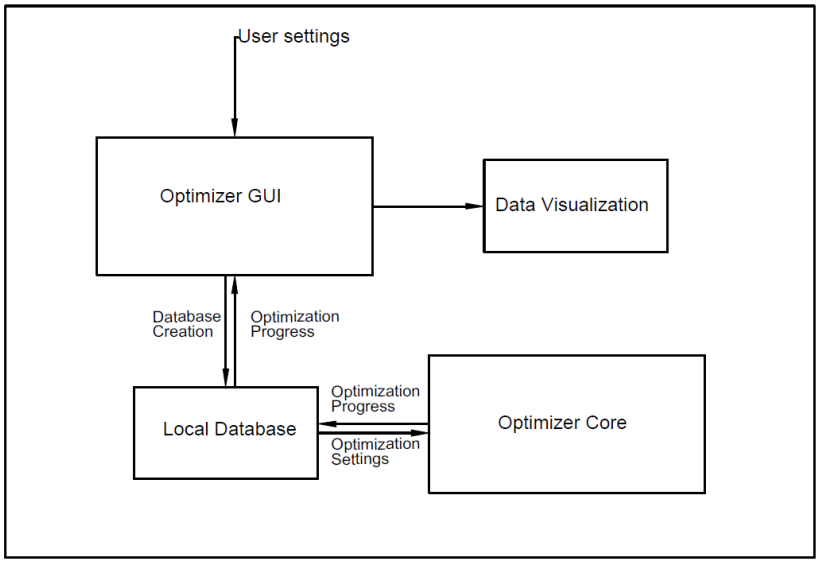

Figure 13: Optimization software flowchart overview

In the following chapters, every box of the flowchart above will be analysed in more depth. The first block that will be analysed is “Optimizer Core” regarding the structure required to solve problems with the optimization software, the libraries used and the code architecture. Finally, “Optimizer GUI” along with his core components, data visualization and data input, will be presented.

5.2. Optimizer Core

The “optimization core” component of the optimization software can be further decomposed in to the following procedures that will be analysed further in this section:

- Database
- Run 1 generation of NSGA-II
- Regularize data
- Evaluate fitness
- Extrapolate population

The existence of a local database plays an important role in the optimization procedure, not only for keeping the software structure cleaner, but also for enhancing the optimization procedure. In the database the user defined optimization parameters are stored and fetched by the “optimizer core” to perform the optimization procedure. All the calculations, parameters, Pareto fronts and calculations are stored inside the database for further analysis from the “optimizer GUI”, thus reducing the amount of information lost. Meanwhile, in order to eliminate unnecessary fitness function evaluations, each individual, before calculated is examined as to whether it already exists in the database, and consequently there is no need to calculate again its fitness value. The database selected for this optimization software was SQLite.

The second core procedure that was developed in the scope of this thesis is the execution of a single generation of the NSGA-II algorithm. The development of the single generation of the NSGA-II algorithm was based on ParadisEO library (Cahon, Melab and Talbi, 2004) with a slight modification due to the fact that ParadisEO was unable to perform a single generation of NSGA-II algorithm without modifications. Performing the NSGA-II one generation at a time is critical in order to be able adjust the flow of the program in case of underwhelming results and also apply the Pareto extrapolation algorithm which needs to be applied at the end of a generation. The following code snippet (Figure 14) is an example of how to initialize the NSGA-II algorithm with the modified version of the ParadisEO library.

```
1  OptimProblemEval eval;
2  inexactEvaluator ipe;
3  eoQuadCloneOp < OptimProblem > xover;                                    // Crossover Class
4  eoUniformMutation < OptimProblem > mutation(mutationEpsilon);            // initialize mutation Class
5  eoRealVectorBounds bounds(optimParam.numberOfVariables, -1., 1.);        // Set scaled Bounds [-1,1]
6  eoRealInitBounded < OptimProblem > init(bounds);                         // Set the above bounds for the initial Pop
7  eoPop < OptimProblem > pop(optimParam.PopulationPerGeneration_multi, init);  // Create Random Population
8  // Run NSGA II until max number of generations are reached //
9
10 for (generationNo = 0; generationNo < optimParam.MaxNumberofGeneration_multi; generationNo++){
11     checkIfPaused();
12     metamodelAssistedNSGAII < OptimProblem > moAlgo(0,eval,ipe,xover,crossoverPropability,mutation,mutationPropability);
13     moAlgo(pop,generationNo==0,false);
14     setUpExtrapolation(pop, init, generationNo == timesOfExtrapolation*EveryHowManyGenerations);
15     updateDbWithCurrentArch(pop,query, generationNo + 1);
16     generationChanged=true;
17 }
18
```

Figure 14: ParadisEO NSGA-II example

Variable regularization is another component that, although not necessary, improves the robustness and the performance of an optimization algorithm. For the reasons mentioned

above, all the values of the individuals are regularized within [-1,1] in the optimization algorithm scope. The program has the task to reverse the regularization of the values of the individuals before their fitness is evaluated. This is accomplished with one of the following functions:

Let an individual X with regularized decision variables values x_i ={1,2,..n},where n is the number of variables, bounded by a lower bound denoted LB_i and an upper bound denoted UB_i then the available equations to express their real world values are:

Equation 1:

$$real\ world\ value_i = \begin{cases} UB_i\ if\ x_i > 0.999 \\ LB_i + {}^{x_i+1}/_{2} \cdot (UB_i - LB_i), if\ -0.999 \le x_i \le 0.999 \\ LB_i, if\ x_i < -0.999 \end{cases}$$

Equation 2:

$$real\ world\ value_i = \begin{cases} UB_i\ if\ x_i > 0.999 \\ \frac{10^{{}^{x_i}/_{k}}}{m}, if\ -0.999 \le x_i \le 0.999 \\ LB_i, if\ x_i < -0.999 \end{cases}$$

Where $m = \sqrt{{}^{1}/_{(UB_i LB_i)}}$ and $k = {}^{1}/_{\log(UB_i \cdot m)}$

The user has the option to select between the equations above to be used by the optimization software while the default option is equation 1. A subtle detail that should be noticed in the equations above is that both of them prevent the evolutionary algorithm from sending values outside the variable bounds to the fitness evaluator. This feature reduces the chance of encountering non-feasible individuals that may result in a runtime error for the fitness function.

Probably, the most important aspect of a general purpose optimization software like the one developed for this thesis is to be able to be used for any optimization problem. In order to achieve the high usability needed, the optimization software gives the user two problem types that can be solved with it.

- Any use-defined fitness function that complies with the following requirements:
    1) Can be executed under the Microsoft Windows operating system
    2) Take as an execution argument the directory of a text file named “variables.txt”
    3) After runtime the fitness function creates in its local directory a text file named “run.txt”, where the fitness values are output.

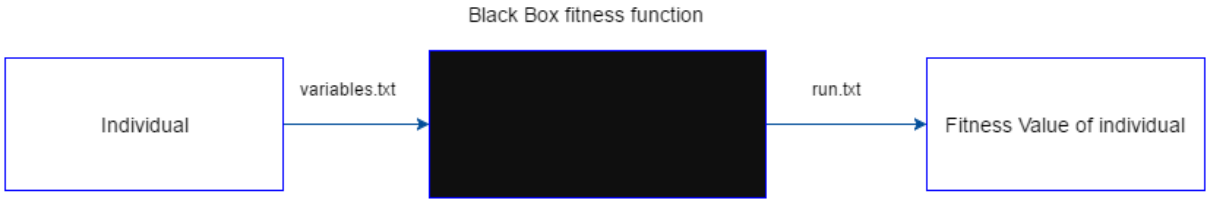

Figure 15: Problem representation of user-defined fitness function

- Integrated ANSYS solver that developed originally by Dr Christos I. Papadopoulos and was integrated to the optimization software as an industrial optimization solver.

Finally the last core component of the "optimizer core" is the Pareto extrapolation algorithm that was presented in section 4.3. Pareto Extrapolation Algorithm of this thesis. The development of the Pareto extrapolation algorithm required the usage of MLP that were implemented with the FANN library (Nissen, 2003). The following code snippets are examples of how to initialize and train (Figure 16: Example of training a neural network with Fann library) a neural network and how to predict values from the neural network approximation function (Figure 17: Example of using a neural network to predict values with Fann library) with the FANN library.

```
float neuralNetwork::train(int epochs){
struct fann_train_data * data = NULL;
struct fann *ann = fann_create_standard(num_layers, num_input, num_neurons_hidden, num_output);
fann_set_activation_function_hidden(ann, activationFunctionHidden);
fann_set_activation_function_output(ann, activationFunctionOutput);
fann_set_training_algorithm(ann, trainingMethod);
data = fann_read_train_from_file(trainingFile.c_str());
fann_set_scaling_params(ann,data,-1,1,-1,1);
fann_scale_train(ann, data);
fann_train_on_data(ann, data, epochs, epochs_between_reports, desired_error);
float returnValue = fann_get_MSE(ann);
fann_destroy_train(data);
fann_save(ann, "optiNN.net");
fann_destroy(ann);
return returnValue;
}
```

Figure 16: Example of training a neural network with Fann library

```
void NeuralNetwork::predictResult(vector<double> inputData_,vector<double>& outputData_,
const int size_){
fann_type *calc_out;
fann_type input[size_];
struct fann *ann = fann_create_from_file("optiNN.net");
for (unsigned int i = 0; i < inputData_.size(); i++){
    input[i] = inputData_[i];
}
fann_scale_input(ann, input);
calc_out = fann_run(ann, input);
fann_descale_output(ann, calc_out);
for (unsigned int i = 0; i < outputData_.size(); i++){
    outputData_[i] = calc_out[i];
}
fann_destroy(ann);
}
```

Figure 17: Example of using a neural network to predict values with Fann library

5.3. Optimizer GUI

Continuing on the decomposing approach that was used on the "optimizer core" block, "optimizer GUI" will be decomposed as well to its main components. The user interface was designed in a manner that would allow both users unfamiliar with optimization theory to use it without being overwhelmed by the optimization parameters as well as more experienced users to fine tune the optimization algorithm for their needs. The starting work environment is shown in Figure 18:

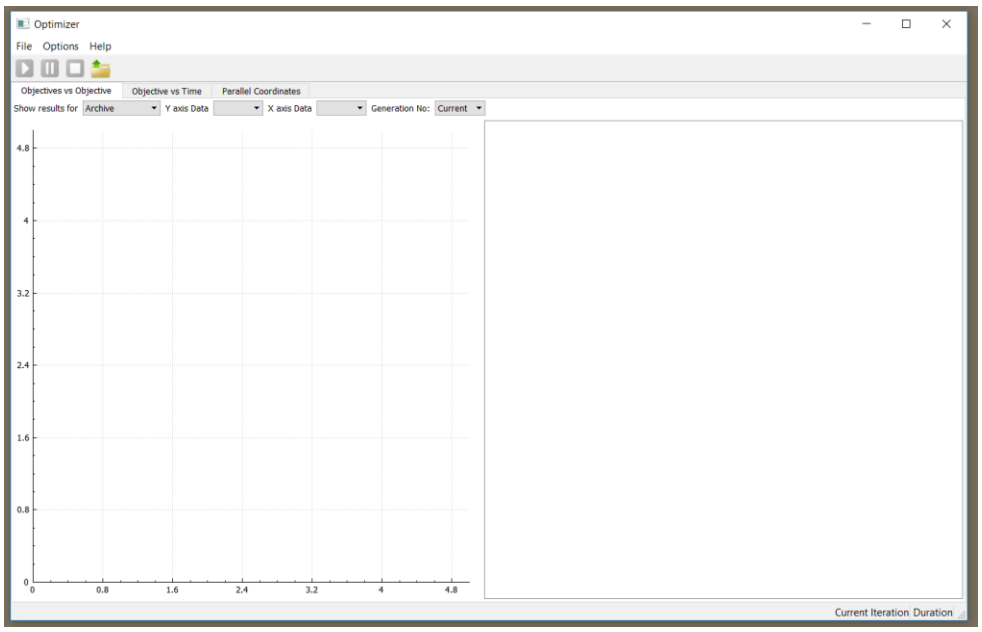

Figure 18 Optimizer GUI stating work environment

In order to create a new optimization problem instance the user has to press:

- Menu->file->New Optimization for generic optimization

- Menu->file->New ANSYS Optimization to use the integrated ANSYS solver

The user goes through an optimization set up procedure, where he sets up the directory of the fitness function, the number of objectives, the number of variables and the bounds of the variables. In case of selecting the ANSYS solver then the optimization software parses the model files and the user decides if a parameter of the ANSYS model selected is an objective or a design variable. For selecting the optimization parameters the user has three types of default choices (standard, passive, aggressive) that will be explained below and the "advanced" option for more experienced users that want to tune the optimization procedure to their exact needs.

- **Standard** option for parameters values
  The standard settings involve selecting parameters that would allow the algorithm to perform well (not optimal) in a high variety of optimization Problems.
  i. Generation number: 150
  ii. Population size: 60
  iii. Crossover probability: 0.9
  High crossover probability has shown to accelerate the optimization procedure when the algorithm is assisted with the Pareto extrapolation algorithm
  iv. Mutation probability: 1/n where n is the number of decision variables

The mutation probability rule used is an acknowledged empirical rule commonly used in the evolutionary optimization bibliography (see e.g. Deb et al. 2002) .

For the Pareto extrapolation algorithm, the parameters have been selected as follows:

i. ev: 20
ii. ex : 1.1
iii. ag : 3
iv. angle: $\pi/4$

- **Passive** option for parameters values
  The passive settings involve selecting parameters that would allow the algorithm to always converge to the Pareto front even if more fitness function evaluations than necessary are needed.
  i. Generation number: 400
  ii. The number of generations ensures that the Pareto optimal set will be found and the evolutionary algorithm will not converge into a local Pareto front.
  iii. Population size: 80
  iv. Crossover probability: 0.8
  High crossover probability has shown to accelerate the optimization procedure when the algorithm is assisted with the Pareto extrapolation algorithm
  v. Mutation probability: 2/n where n is the number of decision variables

The modified version for mutation probability gives a higher mutation rate thus the algorithm will explore higher percentage of the design space.

For the Pareto extrapolation algorithm the parameters have been selected as follows:

i. ev: 25
ii. ex : 0.9
iii. ag : 5
iv. angle: π/4

- **Aggressive** option for parameters values
  The aggressive settings involve selecting parameters that would allow the algorithm to optimize the problem with the least amount of exact fitness function evaluations, but the algorithm faces the risk of converging in to a local Pareto front.
  i. Generation number: 40
     The low generation number and as a result the low number of objective calculations reduces the total computational cost of the algorithm.
  ii. Population size: 40
  iii. Crossover probability: 0.9
     High crossover probability enables the algorithm to repopulate individuals of the Pareto extrapolation algorithm faster.
  iv. Mutation probability: 0.5/n where n is the number of decision variables
     The modified version for mutation probability gives a lower mutation rate thus the algorithm will converge faster but will not be very efficient in escaping local optima.

  For the Pareto extrapolation algorithm the parameters have been selected as follows:

  i. ev: 10
  ii. ex : 0.9
  iii. ag : 10
  iv. angle: π/4

5.3.1. Data visualization

Visualizing the optimization process with suitable graphs is an important feature in any general purpose optimizer. In the optimizer developed for the present thesis various graph types have been implemented to assist the user during optimization. Every graph offers different information regarding the optimization process. In this section the graphical representation implemented will be explained for the reader to understand both why these graphs have been selected and the information the graphs provide. The graphs implemented were:

- Pareto front graph
- Cloud graph

- Parallel coordinate graph

The first and probably the most famous graph that was implemented was the Pareto front graph. This graph represents the compromises that one has to make in one objective to achieve a desirable goal in another objective. Simultaneously this graph also provides the information of optimality since it demonstrates a set of solutions to an optimization problem. The Pareto front diagram has the same dimension as the number of objectives of the optimization problem. Thereafter, in case of an optimization problem with more than two goals the Pareto front cannot be represented in a graph. In that case, the Pareto front is represented as set of planes intersecting the multidimensional space. The following figure illustrates a Pareto front graph as shown at the optimization software.

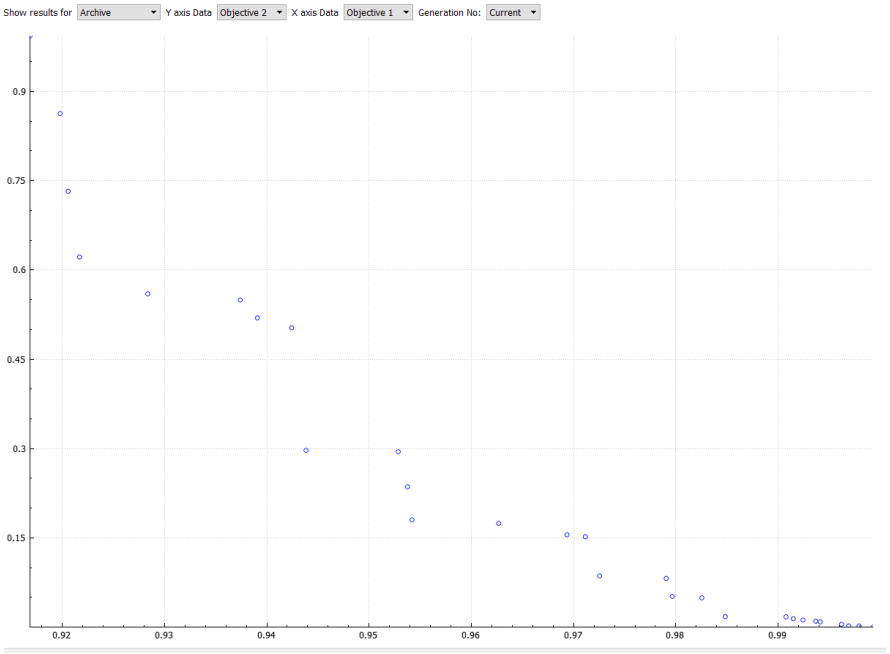

Figure 19: Pareto front graph. Proposed solutions are denoted with blue circle

Another optimization graph that provides a lot of information regarding the optimization progress is the cloud graph. Cloud graph shares the same principles with the Pareto front graph regarding its structure but illustrates all the calculation made during the optimization procedure instead of only the optimal solutions. This difference allows the user to examine the percentage of the search space examined by the evolutionary algorithm as well as see the areas of the search space that the objective function is non-feasible.

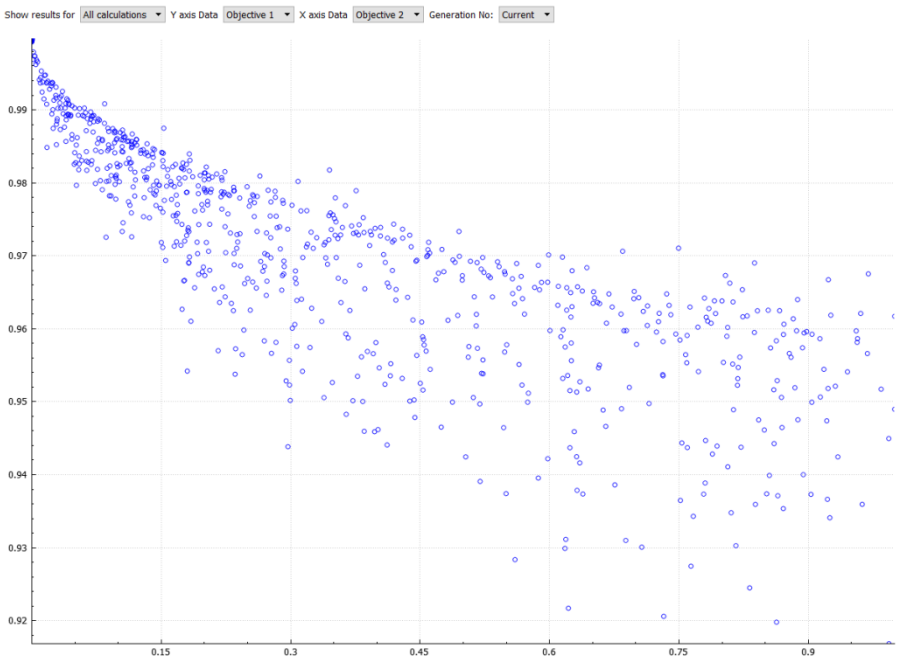

Figure 20: Cloud graph. All the calculations made by the evolutionary algorithm

The last graph implemented in the developed optimization software is the parallel coordinates plot (Ocagne, 1885). Parallel coordinates plot allows the user to visualize the optimization as a whole even when the dimension of the problem is higher than 3. The graph is created with a grid of n lines parallel to the y axis, where n equals the sum of the number of objectives and the number of variables. Then, every solution in the final Pareto front that has the following form:

$Solution_i$:= ($objective_1$,$objective_2$,…,$objective_n$,$variable_1$,$variable_2$,….$variable_n$)

Every solution is represented as a polyline with vertexes on the parallel axes that denote the value of the solution in that dimension. Due to the fact that different objectives or variables might not share the same class size all the objective values used in the parallel coordinates plot are regularized within [-1-1] to ensure proper visualization. Meanwhile, variable values that are already regularized within the scope of the "optimizer core" are not regularized for the second time. As a result, variable values might not belong in [-1,1]. Parallel coordinates plot is a graphical representation of the trade-offs between the variable and the objective values of the Pareto front. Parallel coordinates plot is one the few possible ways to graphically demonstrate the entire Pareto front solutions and their corresponding decision vectors with a single plot. Its importance is increasing as the dimensions of the problems increase. In an optimization problem with three objectives for example parallel coordinates offers a way to present all the Pareto optimal solutions in respect to all three objectives and all the objectives with a single graph.

A parallel coordinates plot from the optimization software is presented in Figure 21

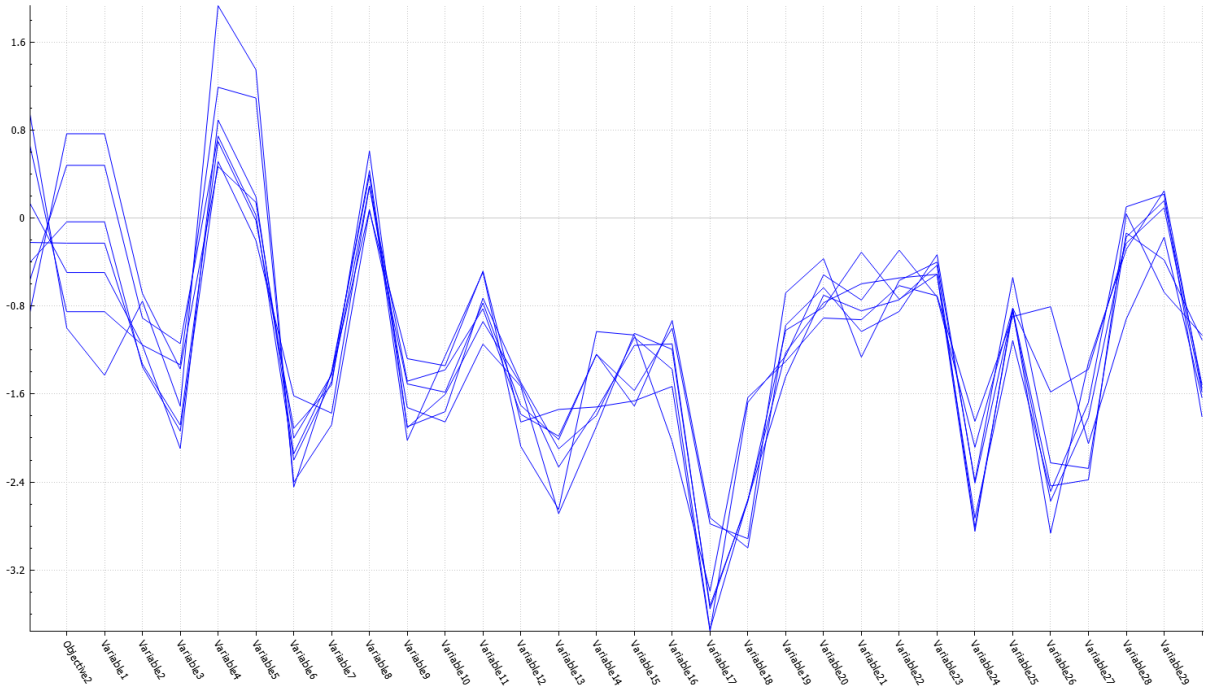

Figure 21: Parallel coordinates plot

Chapter 6: Performance Evaluation of the Developed Optimization Software

6.1. Introduction

This chapter focuses on evaluating the optimization software performance when it is used to solve different optimization problems. In order to measure the performance of the optimization software the following methods have been applied; the hypervolume indicator proposed by Zitzler et al. in 2007 and set convergence proposed by Deb et al. in 2002. Then, a set of benchmarking optimization problems have been defined and solved both with the standard NSGA-II and with the Pareto Extrapolation algorithm plugged in the standard NSGA-II and the results have been analysed and compared. The optimization software will be tested in a set of analytic functions proposed in the evolutionary optimization literature for benchmarking evolutionary algorithms and two "real-world" applications relevant to the marine engineering. The mathematical optimization problems have been selected to evaluate the performance of the optimization algorithms. They have been designed in such a manner as to challenge an optimization algorithm and to server as benchmarking between optimization algorithms. The, two sets of optimization problems belonging in the field of optimal machine design that have been chosen correspond to the different possible solvers of the optimization software: the generic solver and the integrated ANSYS workbench solver. The optimization problems that will be used to evaluate the performance are:

1) ZDT problems proposed by Zitzler, Deb and Thiele, 2000 and implemented by the author of this thesis
2) The problem of optimizing the design of hydrophobic journal bearings under different operational conditions.
3) The problem of optimizing the geometry of three-dimensional micro-thrust bearings.

6.2. Performance Metrics

Although the relations amongst Pareto fronts described in Chapter 1 give a sense of ordering, in most cases there is a need to quantify the quality of a Pareto front. For the quantification of the quality of the Pareto front the first method that will be presented is the Hypervolume Indicator (I_H) which is based on the idea of weak Pareto dominance. The hypervolume indicator is a metric that expresses the volume of the objective space dominated by a Pareto front A and it can be defined based on a reference set R (Auger et al., 2009) as

$$I_H := \lambda\big(H(A,R)\big)$$

Here

- the set $H(A,R) := \{(z_1,z_2) \in \mathbb{R}^2 ; \exists\ x \in A\ , \exists\ (r_1,r_2) \in \mathbb{R} : \forall\ 1 \le i \le 2 : F_i(x) \le z \le r_i\}$ denotes the set of objective vectors that are enclosed by the front $F(A):=\{F(x)|x \in A\}$ given by

A and the reference set R.

- the symbol λ stands for the Lebesgue measure of the set H(A,R).

Analysing further the definition above the following can be concluded:

1) The dimension of the hypervolume indicator dimension is equal to the number of the objectives. Figure 22 shows an example of a hypervolume for a two objective optimization problem.
2) Point R is called reference set and in most case it is simplified as a single point called reference point. This point is dominated by every solution of the Pareto front.
3) The following equivalence describes the relation between the hypervolume indicator and the Pareto front dominance $\mathbf{A} \preccurlyeq \mathbf{B} \Leftrightarrow I_H(\mathbf{A}) \geq I_H(\mathbf{B})$.
4) The hypervolume indicator can be expressed equivalently as the hypervolume of objective space dominated, as the percentage of the objective space dominated and finally as the percentage of the non-dominated objective space. In this thesis the exact metric that will be used is the percentage of the objective space dominated. The percentage of dominated space is calculated through Monte Carlo simulation.

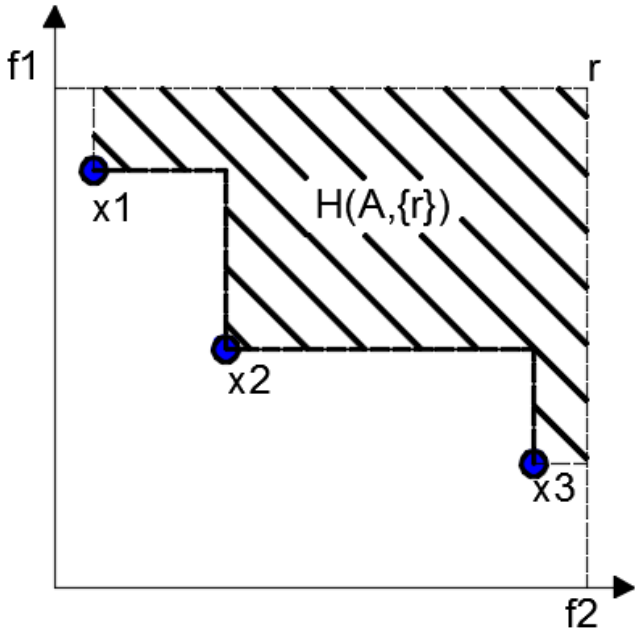

Figure 22: Example of the hypervolume indicator for the Pareto front $X=\{X_1,X_2,X_3\}$

The second metric to quantify the quality of a Pareto front that will be used in this thesis is the set convergence (Deb et al. in 2002), a metric that was used to evaluate the NSGA-II when it was originally proposed. This metric expresses the convergence of Pareto front to a set of true Pareto optimal solutions. Although this metric expresses the quality of a Pareto front really well it has a serious downside; it cannot be used for any problem where the true Pareto-optimal front is not known in advance. Subsequently, this metric will be used as a performance index only for the first set of problems (analytic functions), where the true Pareto-optimal front is known. In order to calculate the performance metric, a set of 1000

uniformly spaced solution from the true Pareto-optimal front are selected. Then, for each solution in a Pareto front the minimum Euclidian distance of it from the known set of Pareto-optimal solutions is calculated. Finally, the set convergence of the Pareto front is the average of the distances calculated for every individual. When every solution lies exactly in the true Pareto optimal set then the set convergence metric value is zero. This performance metric is of higher fidelity than the hypervolume indicator because it takes in to consideration the true Pareto optimal front. However, set convergence performance metric can only be used in benchmarking optimization problems where the true Pareto optimal front is known. In real world applications where the true Pareto optimal front is unknown this metric cannot be used to evaluate the convergence of an evolutionary algorithm. Figure 23 illustrates the set convergence metric.

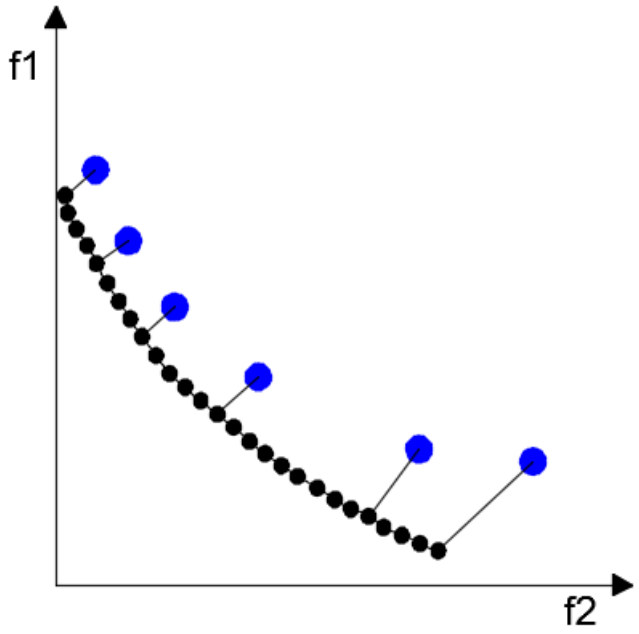

Figure 23: Set convergence metric graphical representation. Points in black colour represent the true Pareto optimal set while points in blue colour represent a Pareto front calculated with an optimization algorithm. The length of the lines connecting blue points with black points show is the minimum Euclidian distance needed to be calculated

6.3. Optimization Benchmarking Problems

6.3.1 Overview

The first set of optimization problems that will be solved with the optimization software developed in the scope of this thesis is the Zitzler-Deb-Thiele problems (Zitzler, Deb and Thiele, 2000). These problems are designed in such manner that each one has different characteristics that would challenge an optimization algorithm. In every problem the fitness functions in analytic form, the number of variables, the bounds of the variables, the characteristics of the Pareto front and the graph of the Pareto front will be presented. Hence, problem ZDT 5 is a binary optimization problem, it has been omitted. Without further ado the optimization problems along with characteristics and their true Pareto fronts are presented:

- ZDT 1 Problem

$$\text{Minimize}\begin{cases} f_1(x) = x_1 \\ f_2(x) = g(x)\left[1 - \sqrt{{x_1}/{g(x)}}\right] \\ g(x) = 1 + {9\left(\sum_{i=2}^{n} x_i\right)}/{(n-1)} \end{cases}$$

bound to $x_i \in [0,1]$ i=1,2,…,n

Here the number of decision variables (n) is 30. The Pareto front of this problem is convex.

The true Pareto front of the ZDT1 problem is illustrated in Figure 24:

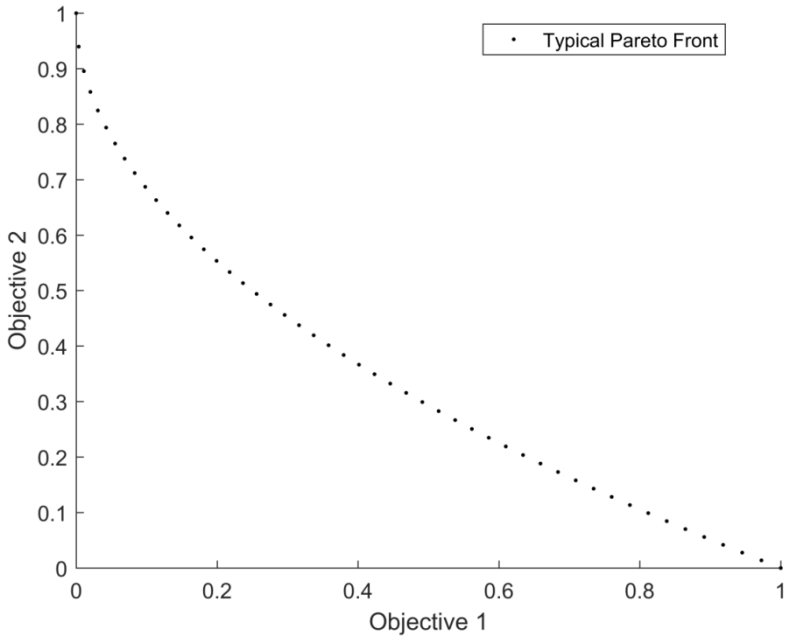

Figure 24: ZDT1 true Pareto front

- ZDT 2 Problem

$$\text{Minimize} \begin{cases} f_1(x) = x_1 \\ f_2(x) = g(x)\left[1 - \left({x_1}/{g(x)}\right)^2\right] \\ g(x) = 1 + {9\left(\sum_{i=2}^{n} x_i\right)}/{(n-1)} \end{cases}$$

bound to $x_i \in [0,1]$ i=1,2,…,n

Here the number of decision variables (n) is 30. The Pareto front of this problem is non-convex.

The true Pareto front of the ZDT2 problem is illustrated in Figure 25:

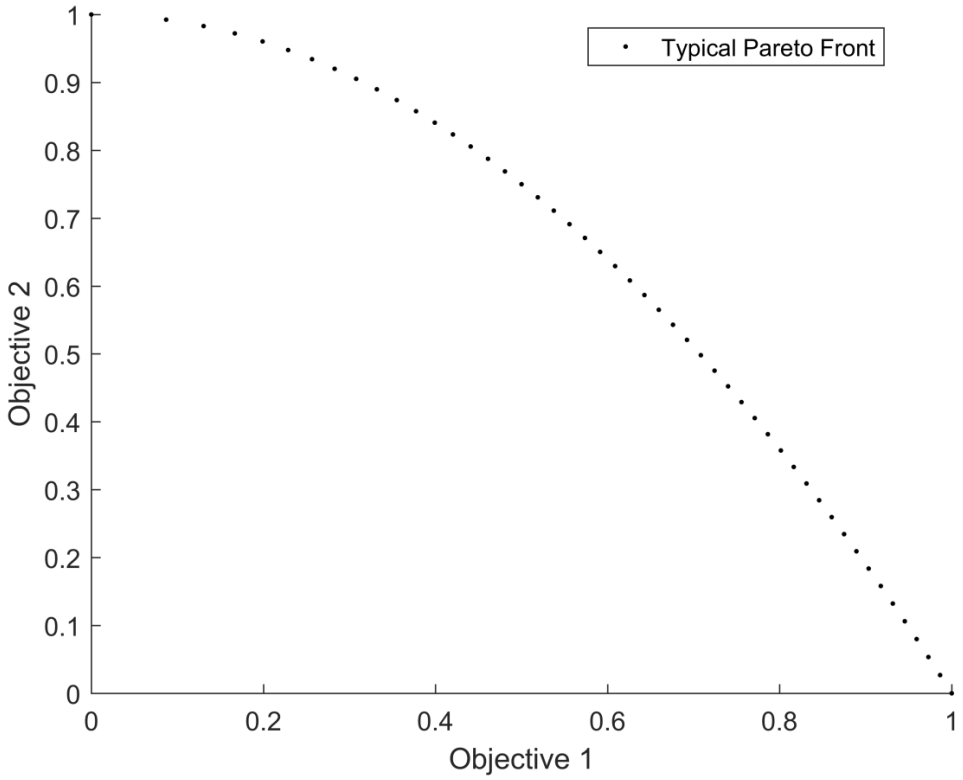

Figure 25:ZDT2 true Pareto front

- ZDT 3 Problem

$$\text{Minimize}\begin{cases} f_1(x) = x_1 \\ f_2(x) = g(x)\left[1 - \sqrt{x_1/g(x)} - \frac{x_1}{g(x)}\sin(10\pi x_1)\right] \\ g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right)/(n-1) \end{cases}$$

bound to $x_i \in [0,1]$ i=1,2,…,n

Here the number of decision variables (n) is 30. The Pareto front of this problem consists of discontinuous convex parts.

The true Pareto front of the ZDT3 problem is illustrated in Figure 26:

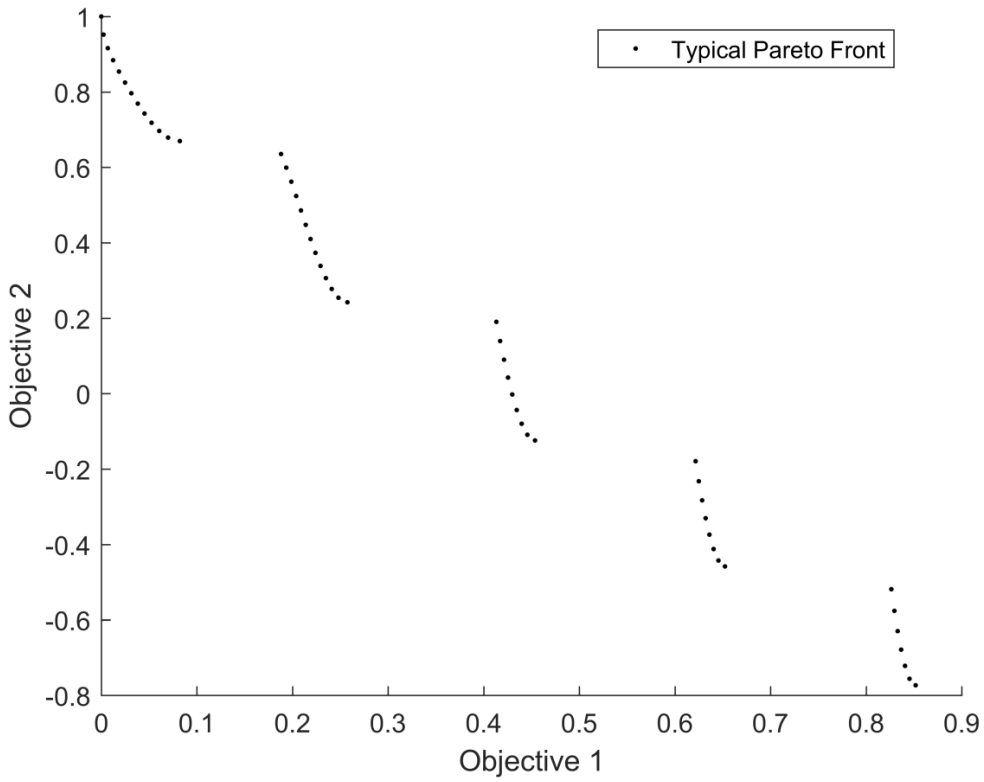

Figure 26: ZDT3 true Pareto front

- ZDT 4 Problem

$$\text{Minimize}\begin{cases} f_1(x) = x_1 \\ f_2(x) = g(x)\left[1 - \left(f_1(x)/g(x)\right)^2\right] \\ g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}[x_i{}^2 - 10\cos(4\pi x_i)] \end{cases}$$

bound to $x_1 \in [0,1]$ and $x_i \in [-5,5]$ i=2,3,…,n

here the number of decision variables (n) is 30. Although the Pareto front of this problem is similar to the ZDT1 Pareto front this problem has 21^9 local Pareto fronts and challenges EA in the aspect of multimodality.

The true Pareto front of ZDT4 problem is illustrated in Figure 27:

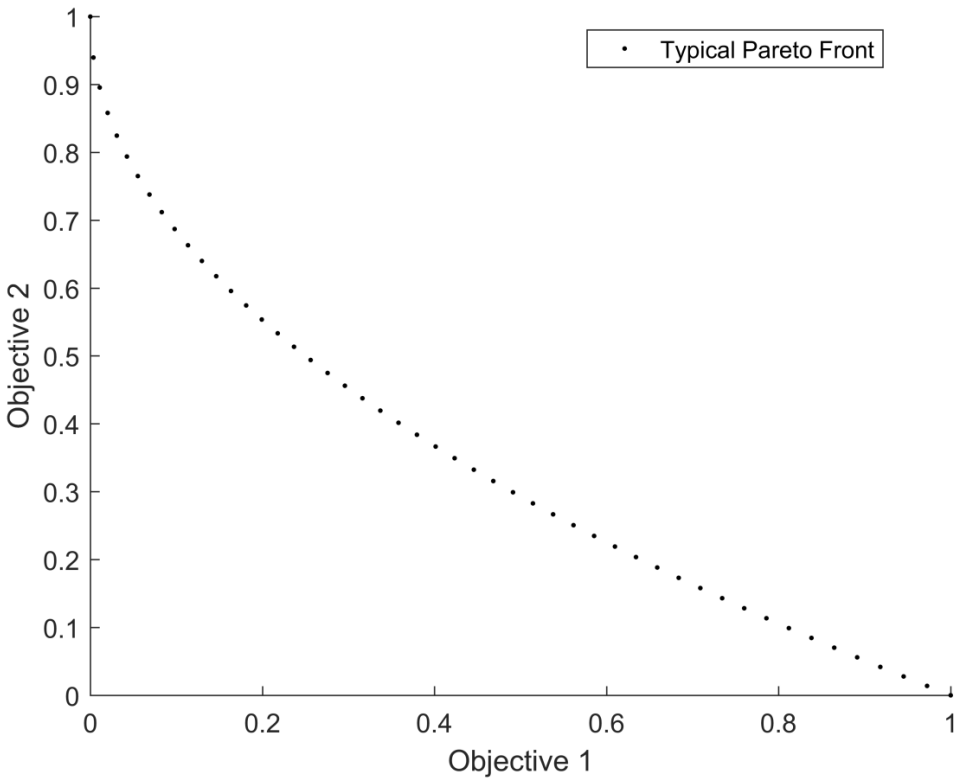

Figure 27:ZDT4 true Pareto front

- ZDT 6 Problem

$$\text{Minimize}\begin{cases} f_1(x) = 1 - \exp(-4x_1)\, sin^6(6\pi x_1) \\ f_2(x) = g(x)\left[1 - \sqrt{{x_1}/{g(x)}}\right] \\ g(x) = 1 + \left[{9(\sum_{i=2}^{n} x_i)}/{(n-1)}\right]^{0.25} \end{cases}$$

bound to $x_i \in [0,1]$ i=1,2,…,n

here the number of decision variables (n) is 30. Finding the Pareto front of this function is rather challenging because the points consisting the Pareto front are nonuniformly distributed along the front.

The true Pareto front of the ZDT6 problem is illustrated in Figure 28:

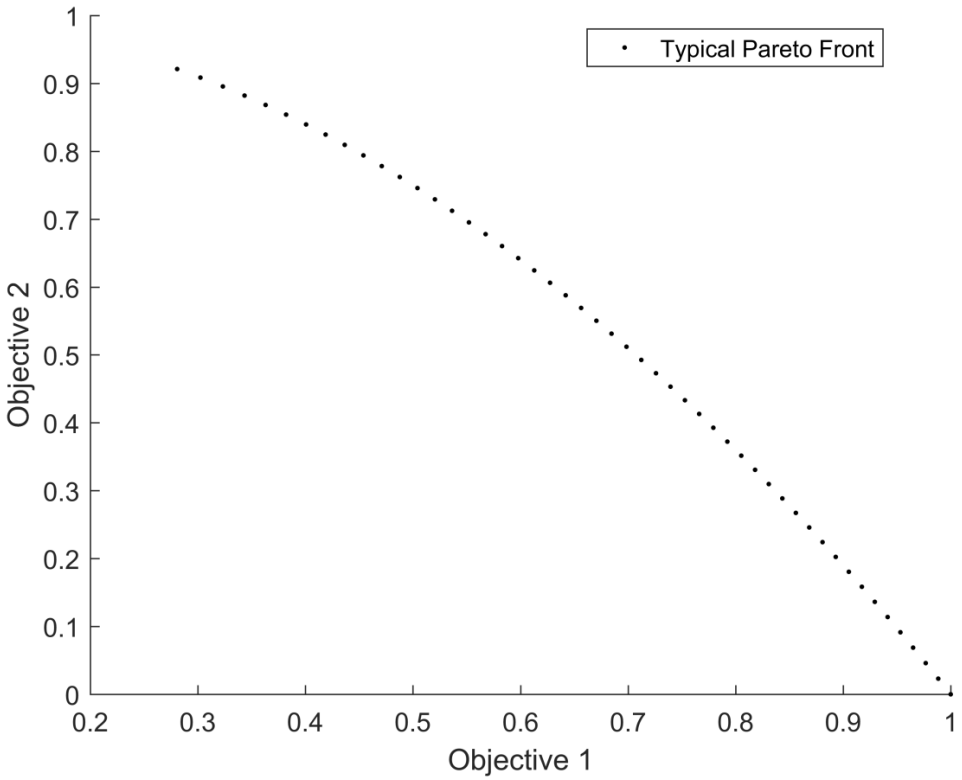

Figure 28: ZDT 6 true Pareto front

In order to assess the performance of the developed optimization software, the optimization problems mentioned above have been solved 30 times per problems both with the standard NSGA-II as well as with the NSGA-II assisted with the Pareto extrapolation algorithm.

The optimization parameters that have been selected to solve the problem are the following:

Both standard NSGA-II and PEA-NSGA-II have been used separately to solve the optimization problem. The evaluation of the Pareto fronts from both algorithms has resulted from 30 separate runs of every algorithm. The runs have been limited to 1100 objective calculations. As this problem is an analytic problem, true Pareto optimal has been used to evaluate the performance of the algorithms. The NSGA II and the PEA-NSGA II are tuned as follows:

NSGA-II parameters:

- Population size: 80
- Crossover rate 0.9
- Mutation probability 1-1/n where n is the number of decision variables

PEA parameters:

- Ev 20 (for problems with mutation probability of 0.03) and 8 (for problems with mutation probability 0.1)
- Ex 1.1
- Ag 3
- Angle 0.785 ($\pi/4$)

Reference points for calculating the hypervolume indicator:

- R=[11 11] for ZDT1,ZDT2 and ZDT3
- R= [1.1 160] for ZDT4
- R=[1.1 9] for ZDT6

6.3.2 Optimization Results

For every problem mentioned in the previous section the optimization results are presented. Results consist of:

- Table of results which illustrates the average value and the standard deviation of the performance metrics described in section 6.1. for every Pareto front every 100 objective calculations.
- The averaged hypervolume indicator and set convergence graphs which are the graphical representation of the abovementioned table.
- Complete hypervolume indicator graph and Complete set convergence graph. For clarity reasons the graphs with the metrics values for every “run” are presented.
- Finally, the Pareto front graphs with both algorithms are presented.

For the optimization problem ZDT1 the Pareto extrapolation feature will be graphically represented. The Pareto extrapolation feature is similar in all other optimization problems.

In evolutionary multi-objective optimization Pareto fronts do not appear in fixed number of objective calculations. In order to solve the problem and be able to compare the quality metrics of Pareto fronts every 100 objective calculations the following algorithm was implemented. For every optimization, the performance metrics where calculated for the Pareto front before and after the comparison points (every 100 objective calculations in this case). The performance metrics were then calculated for comparison points interpolating the results before and after. Finally, the average and standard deviation of the performance metrics are calculated for every comparison point.

ZDT1 Problem

Table 4 and Table 5 demonstrate the values of the performance metrics (set convergence and hypervolume indicator) for ZDT1 optimization problem every 100 objective calculations.

Table 4: PEA NSGA-II results for the ZDT1 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.76282	0.01348	2.64630	0.16542
200	0.81062	0.02412	1.98042	0.39683
300	0.86152	0.02546	1.31383	0.37311
400	0.89088	0.02568	0.94012	0.28638
500	0.91574	0.02284	0.65821	0.22523
600	0.93203	0.02163	0.42955	0.17802
700	0.94649	0.01511	0.29656	0.13333
800	0.95847	0.01551	0.22967	0.11967
900	0.96660	0.01235	0.17129	0.09129
1000	0.97342	0.00983	0.13560	0.07617
1100	0.97724	0.01019	0.10678	0.06835

Table 5: NSGA-II results for the ZDT1 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.766210	0.016051	2.622257	0.185175
200	0.800967	0.018350	2.145309	0.207267
300	0.831607	0.019340	1.752261	0.215571
400	0.858234	0.016640	1.443577	0.208893
500	0.879759	0.015865	1.215535	0.218559
600	0.897627	0.017876	0.995901	0.242339
700	0.913064	0.018880	0.833318	0.228668
800	0.925017	0.020003	0.712502	0.224479
900	0.939606	0.017831	0.582112	0.193322
1000	0.948501	0.018848	0.479207	0.187592
1100	0.956307	0.017946	0.407935	0.179530

The results presented in Table 4 and Table 5 are illustrated in Figure 29 and Figure 30

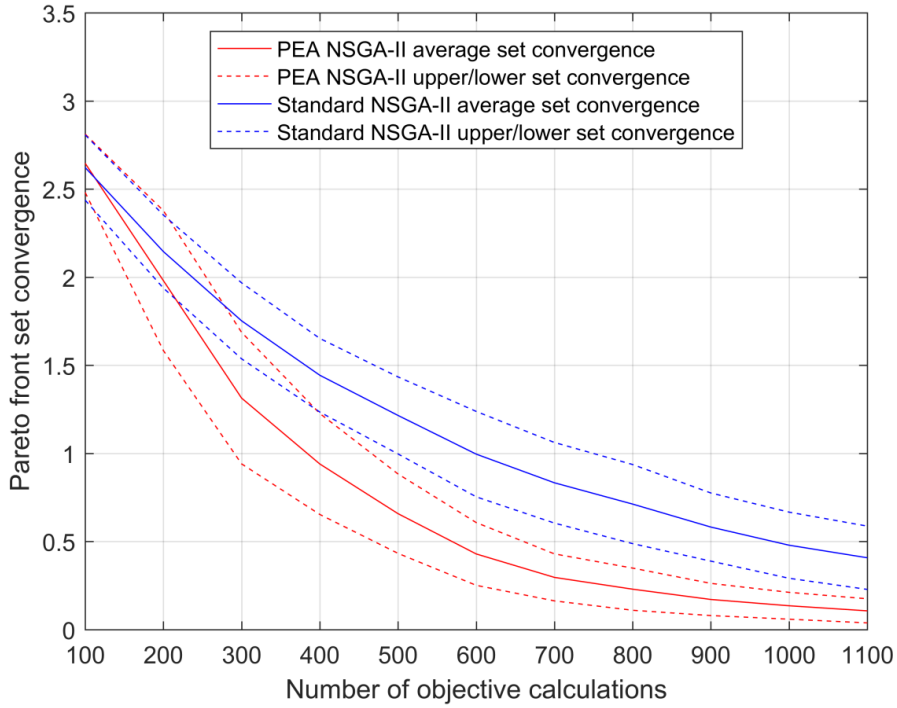

Figure 29: ZDT1 average set convergence graph

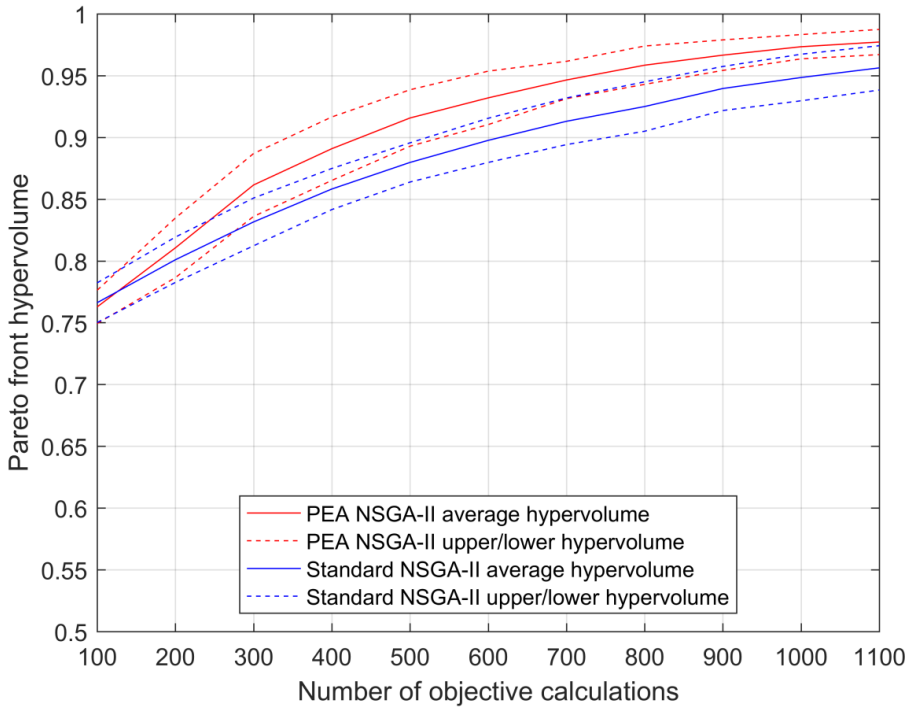

Figure 30: ZDT1 average hypervolume indicator graph

For clarity reasons, the performance metrics values from every independent optimization run are presented in Figure 31 and Figure 32

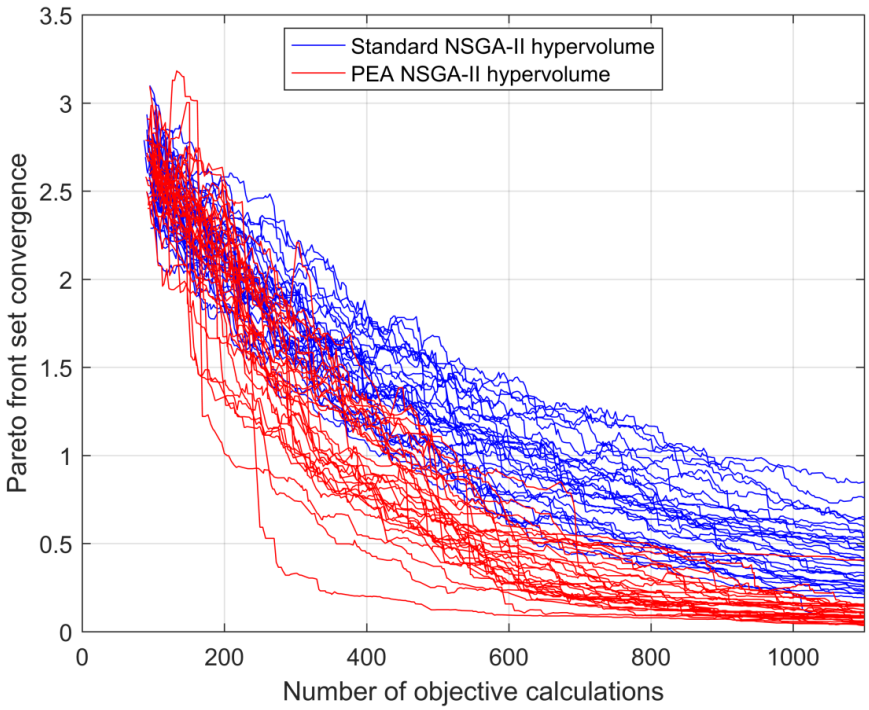

Figure 31: ZDT1 set convergence results for 30 independent runs

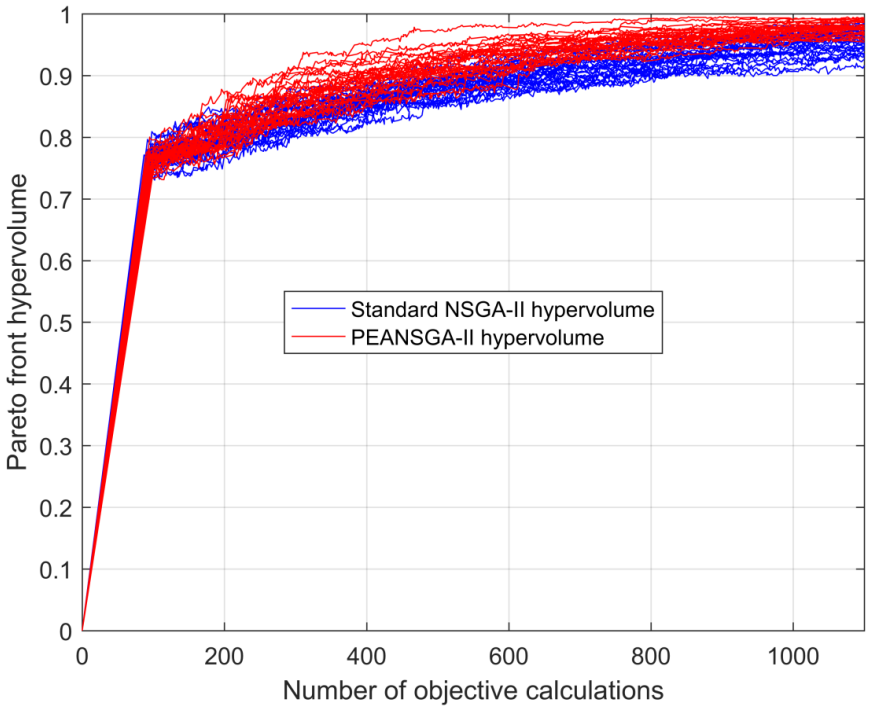

Figure 32: ZDT1 hypervolume indicator for 30 independent runs

Figure 33 illustrates a typical Pareto front at 1100 objective calculations. The Pareto fronts chosen for graphical representation are the fronts with the absolute minimum difference in performance metrics values from the average values at 1100 objective calculations. This allows the reader to understand how different performance metrics values correspond to different fronts. Also it is clearly demonstrated that PEA NSGA-II outperforms NSGA-II in terms of Pareto front convergence for a given number of objective calculations.

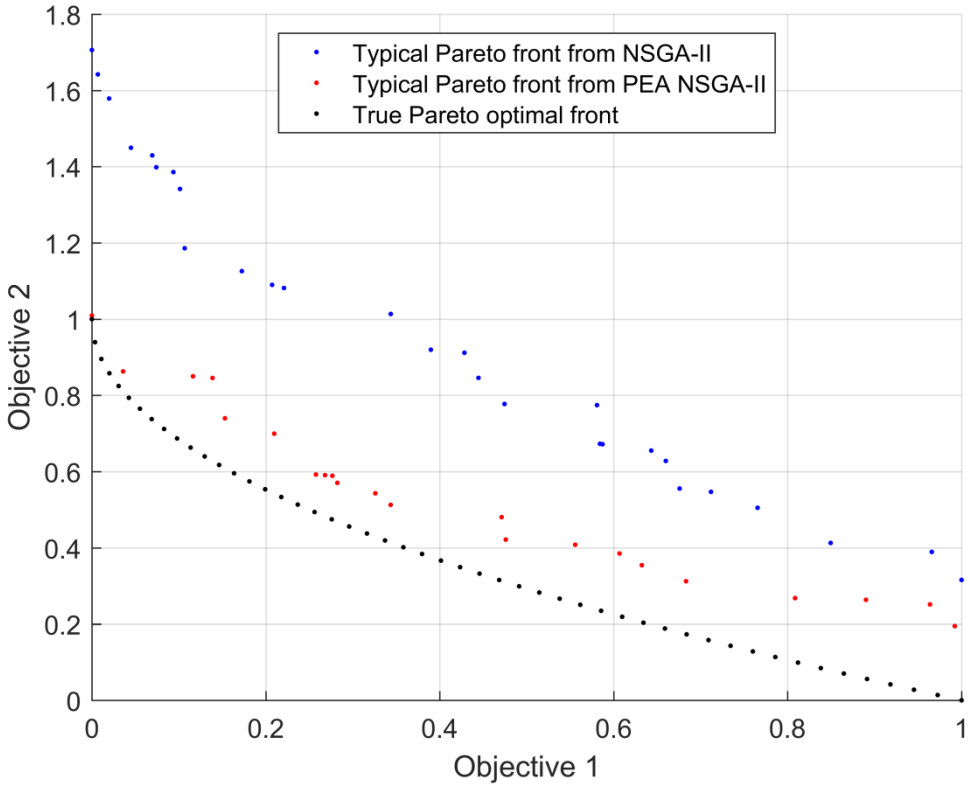

Figure 33: Typical Pareto fronts for the ZDT1 problem by the NSGA-II and the PEA NSGA-II algorithms after 1100 objective calculations

In Figure 34 to Figure 39 the extrapolation feature of PEA NSGA-II is illustrated graphically. To achieve the following information is plotted:

a) Black points illustrate the Pareto front before the application of the PEA algorithm
b) Red points illustrate extrapolated points calculated by PEA algorithm
c) Blue circled points illustrate the Pareto front after the application of the PEA algorithm

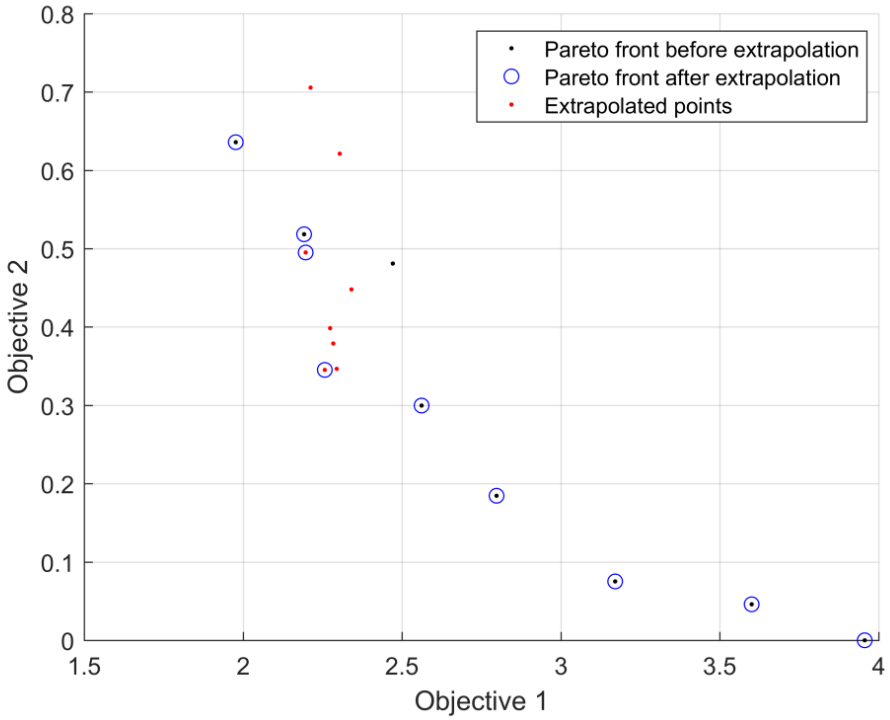

Figure 34: ZDT1 Pareto front before and after extrapolation (40 generations)

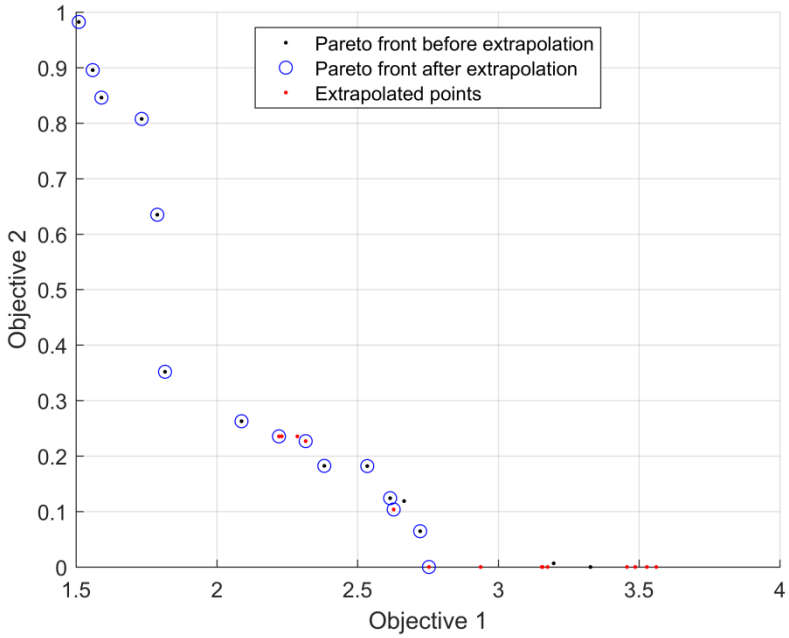

Figure 35: ZDT1 Pareto front before and after extrapolation (80 generations)

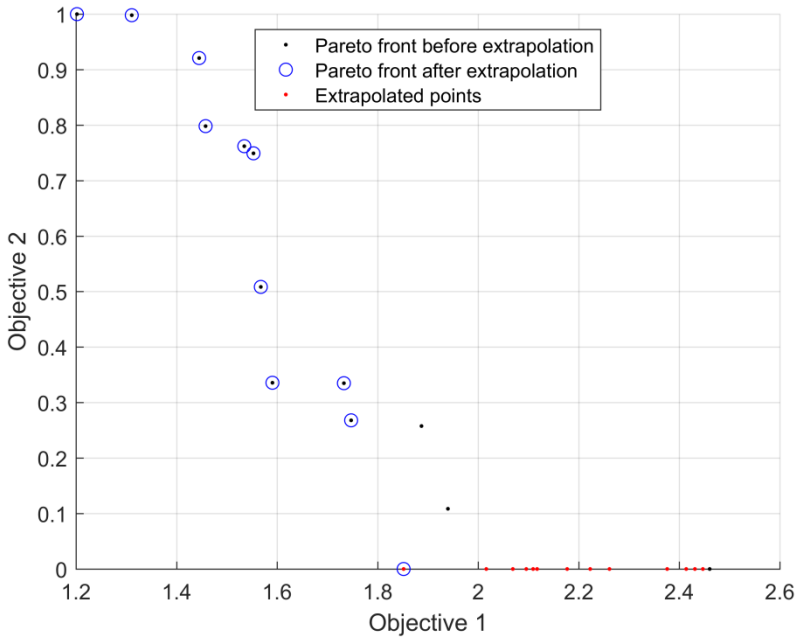

Figure 36: ZDT1 Pareto front before and after extrapolation (120 generations)

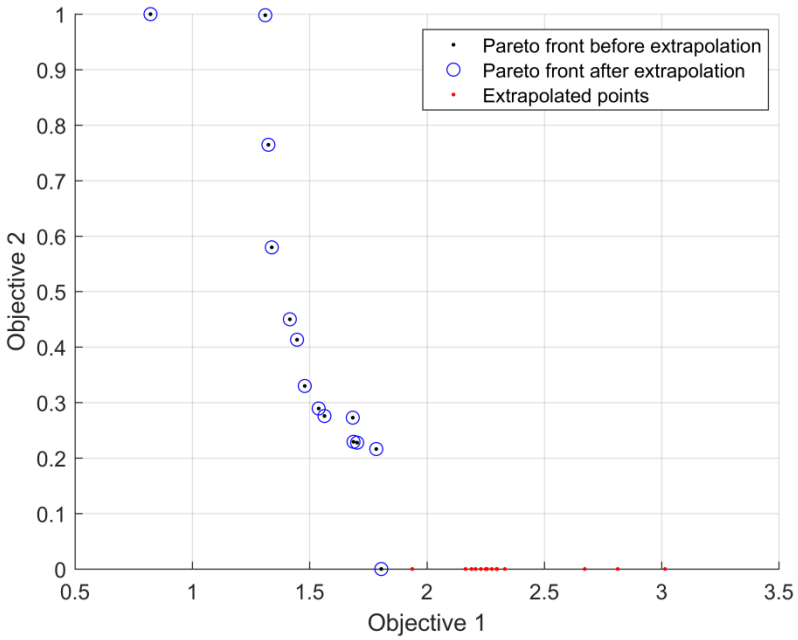

Figure 37: ZDT1 Pareto front before and after extrapolation (160 generations)

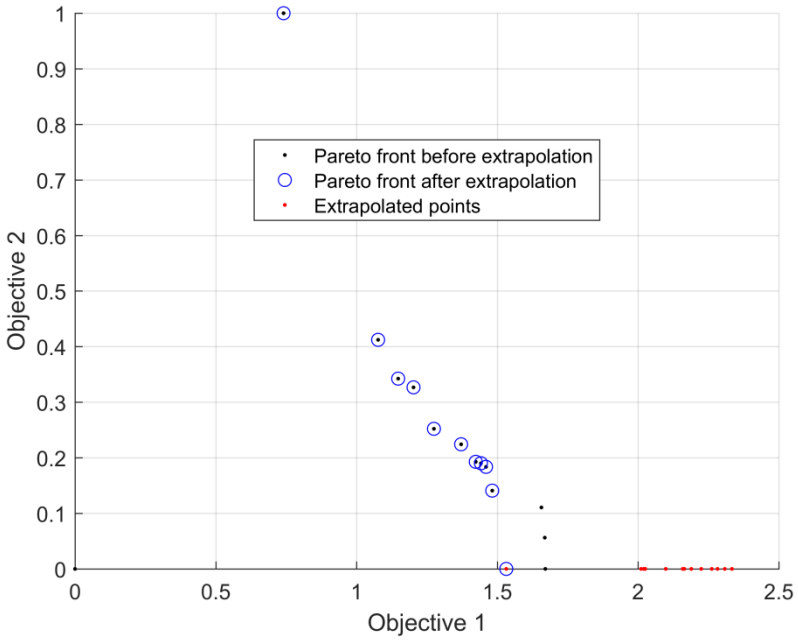

Figure 38: ZDT1 Pareto front before and after extrapolation (200 generations)

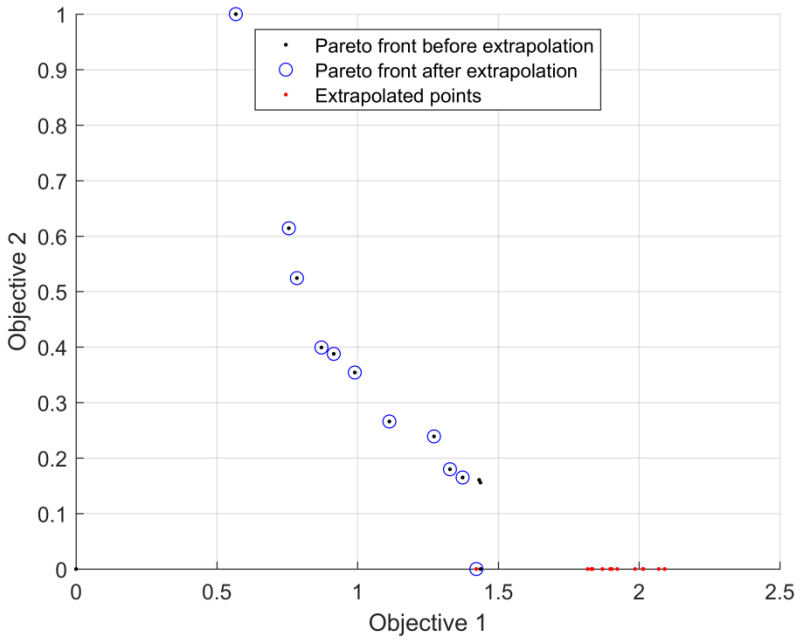

Figure 39: ZDT1 Pareto front before and after extrapolation (240 generations)

ZDT2 Problem

Table 6: and Table 7: Standard NSGA-II results for the ZDT2 problem demonstrate the values of the performance metrics (set convergence and hypervolume indicator) for ZDT2 optimization problem every 100 objective calculations.

Table 6: PEA NSGA-II results for the ZDT2 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.63447	0.01923	3.44296	0.23661
200	0.72616	0.02978	2.21833	0.38391
300	0.80941	0.03480	1.24735	0.40867
400	0.86002	0.03066	0.69267	0.32409
500	0.89750	0.02558	0.35547	0.25509
600	0.91970	0.02357	0.19703	0.15622
700	0.93808	0.02634	0.13367	0.11949
800	0.95278	0.02234	0.10057	0.09141
900	0.96441	0.02205	0.08946	0.09366
1000	0.97123	0.02156	0.07954	0.09333
1100	0.97672	0.02056	0.07253	0.09222

Table 7: Standard NSGA-II results for the ZDT2 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.64038	0.02138	3.40294	0.23109
200	0.71594	0.01907	2.40874	0.30181
300	0.77725	0.03295	1.71097	0.42722
400	0.82348	0.02733	1.13098	0.37786
500	0.85848	0.02737	0.76469	0.35926
600	0.88823	0.02398	0.49732	0.34478
700	0.90999	0.02726	0.36538	0.30034
800	0.92696	0.02983	0.26674	0.23928
900	0.94199	0.02810	0.19510	0.19333
1000	0.95448	0.03063	0.15059	0.16501
1100	0.96748	0.02514	0.11275	0.13872

The results presented in Table 6: and Table 7: Standard NSGA-II results for the ZDT2 problem are illustrated in Figure 40 and Figure 41

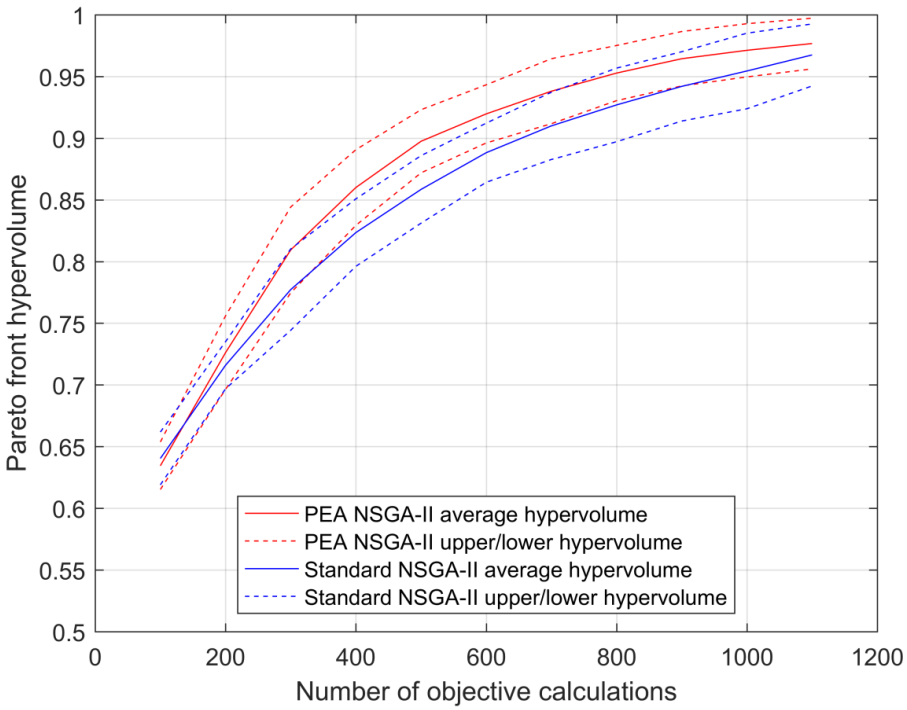

Figure 40: ZDT2 average hypervolume indicator graph

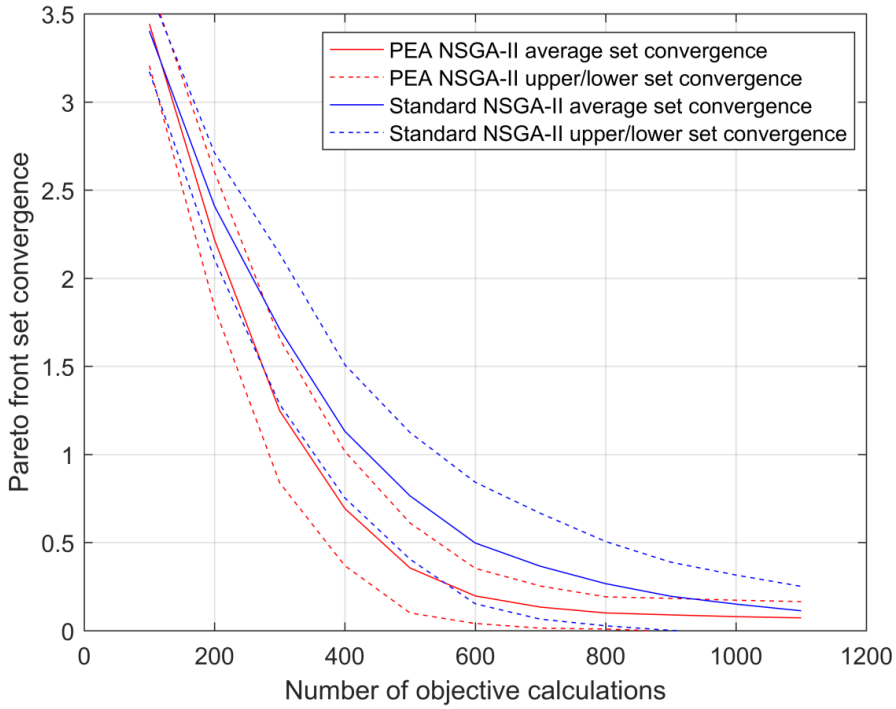

Figure 41: ZDT2 average set convergence graph

For clarity reasons, the performance metrics values from every independent optimization run are presented in Figure 42 and Figure 43

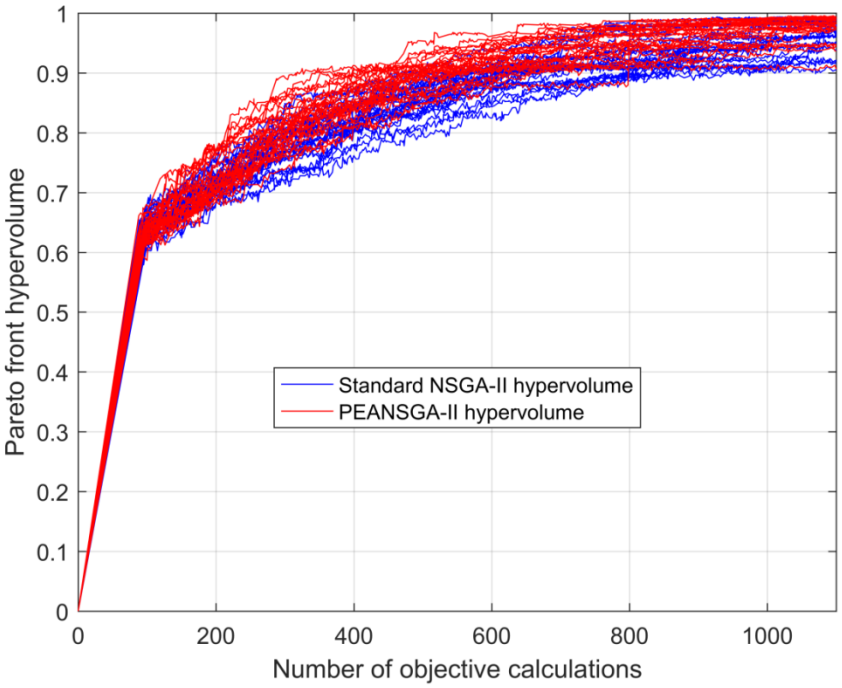

Figure 42: ZDT2 hypervolume results for 30 independent runs

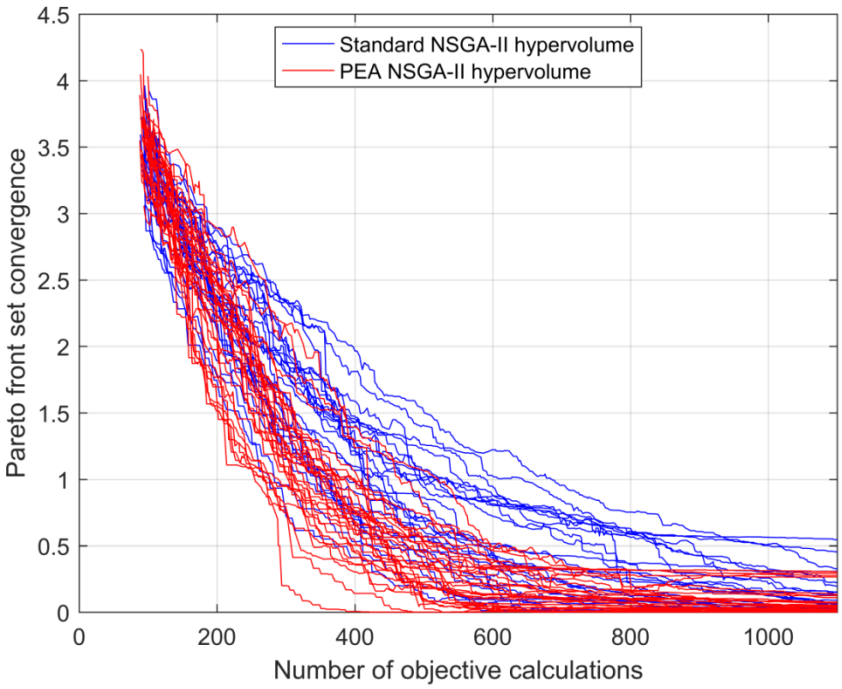

Figure 43: ZDT2 set convergence results for 30 independent runs

Figure 44 illustrates a typical Pareto front at 1100 objective calculations. The Pareto fronts chosen for graphical representation are the fronts with the absolute minimum difference in performance metrics values from the average values at 1100 objective calculations. This allows the reader to understand how different performance metrics values correspond to different fronts. Also it is clearly demonstrated that PEA NSGA-II outperforms NSGA-II in terms of Pareto front convergence for a given number of objective calculations.

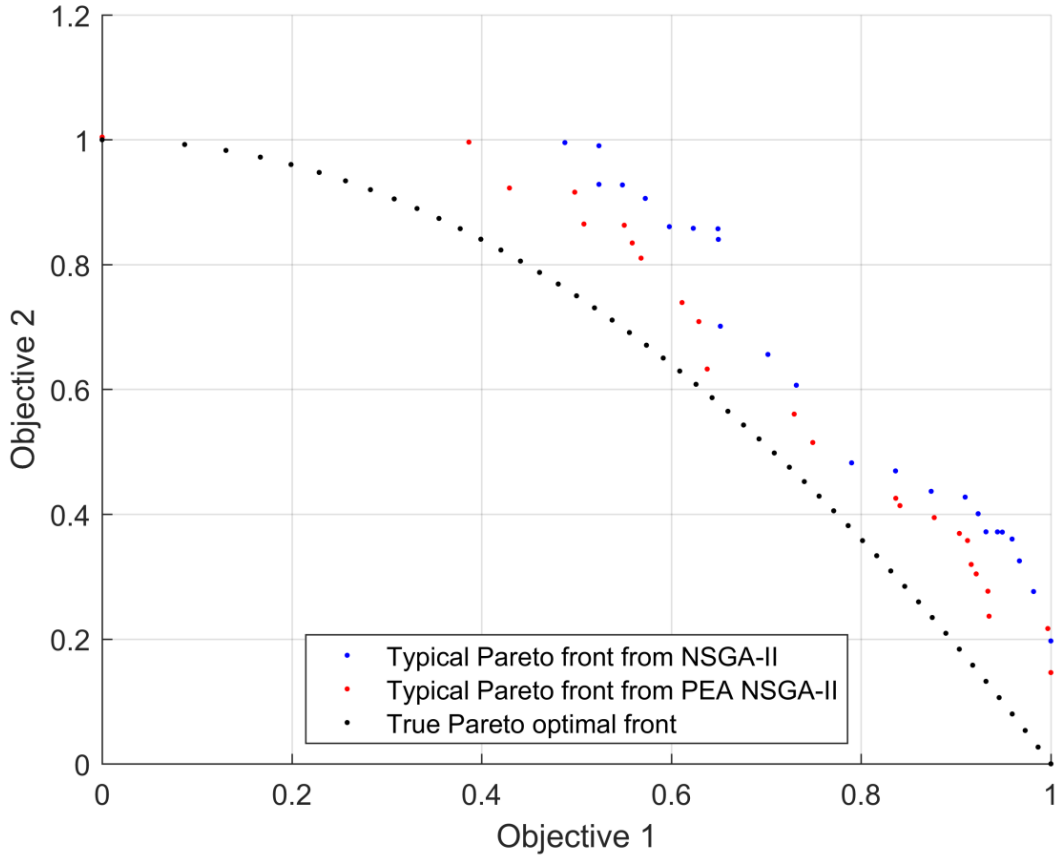

Figure 44: Typical Pareto fronts for the ZDT2 problem by the NSGA-II and the PEA NSGA-II algorithms after 1100 objective calculations

ZDT3 Problem

Table 6: and Table 7: Standard NSGA-II results for the ZDT2 problem demonstrate the values of the performance metrics (set convergence and hypervolume indicator) for ZDT3 optimization problem every 100 objective calculations.

Table 8: PEA-NSGA II results for the ZDT3

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.79839	0.02167	2.56208	0.17959
200	0.83005	0.03161	1.97914	0.39179
300	0.86516	0.03600	1.36316	0.45551
400	0.89440	0.03652	0.99240	0.46454
500	0.91226	0.03854	0.71532	0.39052
600	0.92722	0.03445	0.50823	0.34212
700	0.93989	0.03140	0.36676	0.30178
800	0.94846	0.02850	0.28697	0.26191
900	0.95484	0.02873	0.21110	0.22442
1000	0.95949	0.02514	0.15169	0.15849
1100	0.96499	0.01966	0.11878	0.13568

Table 9 Standard NSGA-II results for the ZDT3

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.79619	0.02049	2.49740	0.24179
200	0.81175	0.02136	2.09989	0.22926
300	0.82550	0.02369	1.82303	0.20151
400	0.83316	0.02473	1.62227	0.22711
500	0.84513	0.01837	1.40486	0.21025
600	0.86132	0.02078	1.19597	0.22467
700	0.87600	0.01817	0.97473	0.24001
800	0.89104	0.02296	0.82658	0.26553
900	0.90240	0.02570	0.69314	0.26341
1000	0.91213	0.02405	0.54622	0.22792
1100	0.92186	0.02343	0.45171	0.20129

The performance metrics values in Table 7and Table 8 are also in illustrated in Figure 45: ZDT3 average hypervolume graph and Figure 46: ZDT3 average set convergence graph below:

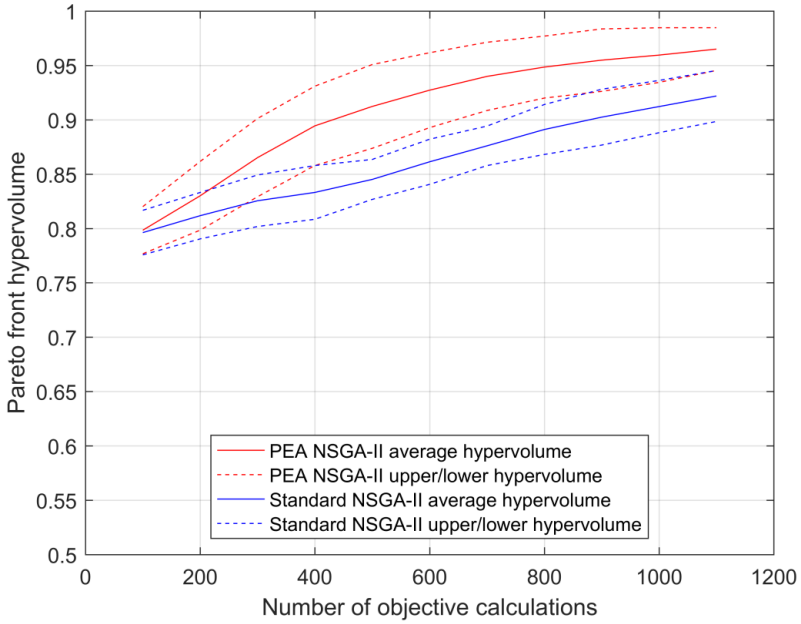

Figure 45: ZDT3 average hypervolume graph

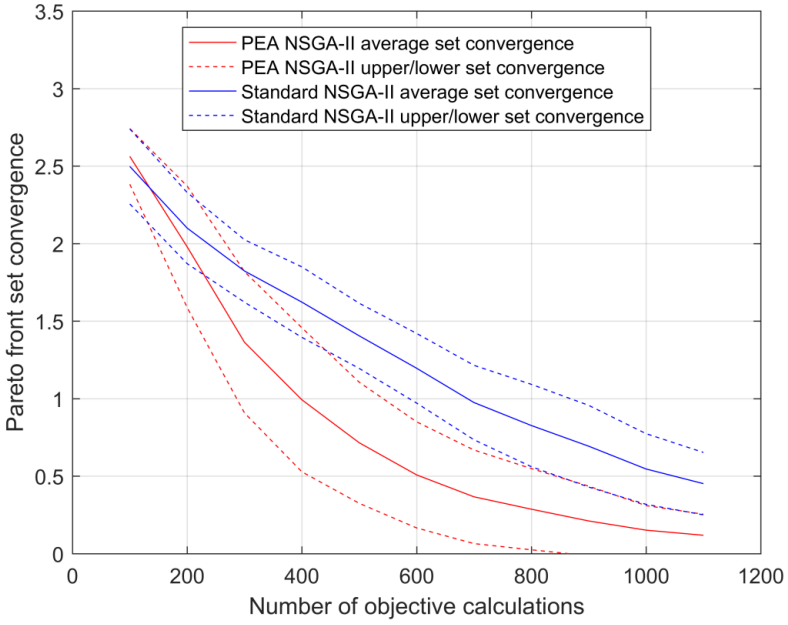

Figure 46: ZDT3 average set convergence graph

For clarity reasons, the performance metrics values from every independent optimization run are presented in Figure 47 and Figure 48.

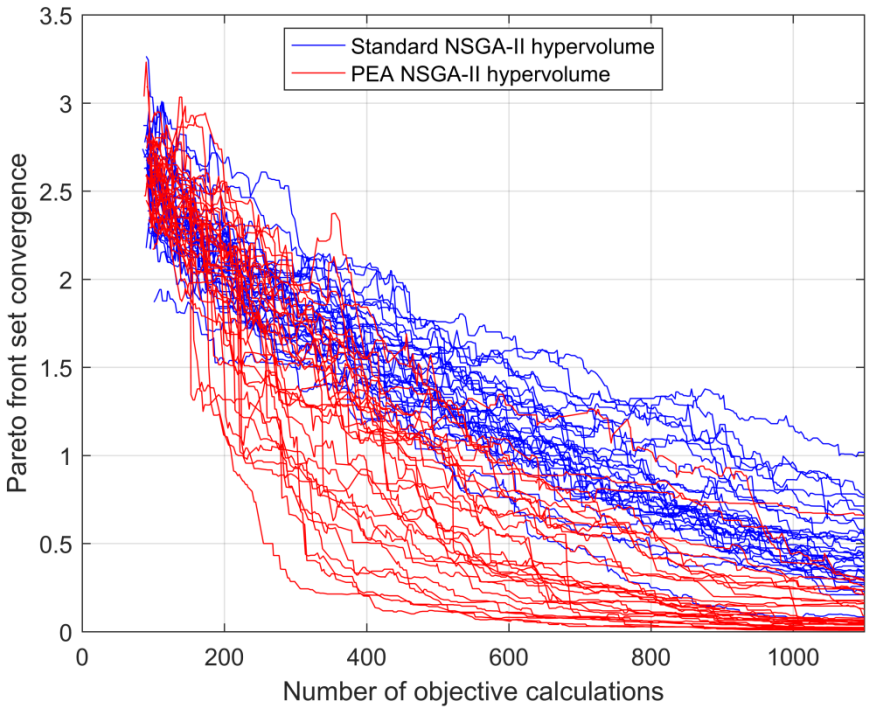

Figure 47: ZDT3 hypervolume results for 30 independent runs

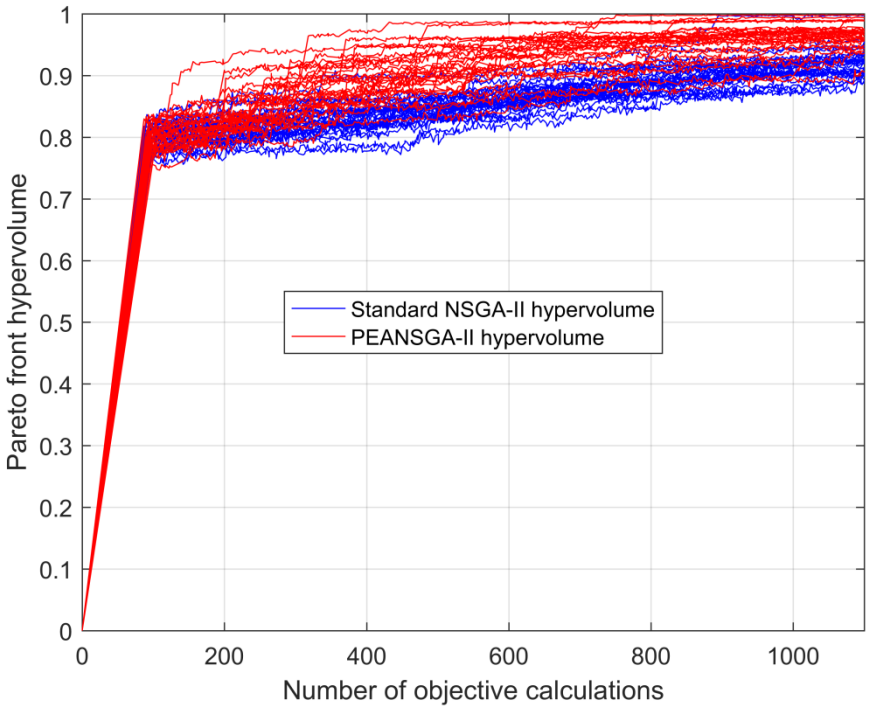

Figure 48: ZDT3 set convergence results for 30 independent runs

Figure 49: illustrates a typical Pareto front at 1100 objective calculations. The Pareto fronts chosen for graphical representation are the fronts with the absolute minimum difference in performance metrics values from the average values at 1100 objective calculations. This allows the reader to understand how different performance metrics values correspond to different fronts. Also it is clearly demonstrated that PEA NSGA-II outperforms NSGA-II in terms of Pareto front convergence for a given number of objective calculations.

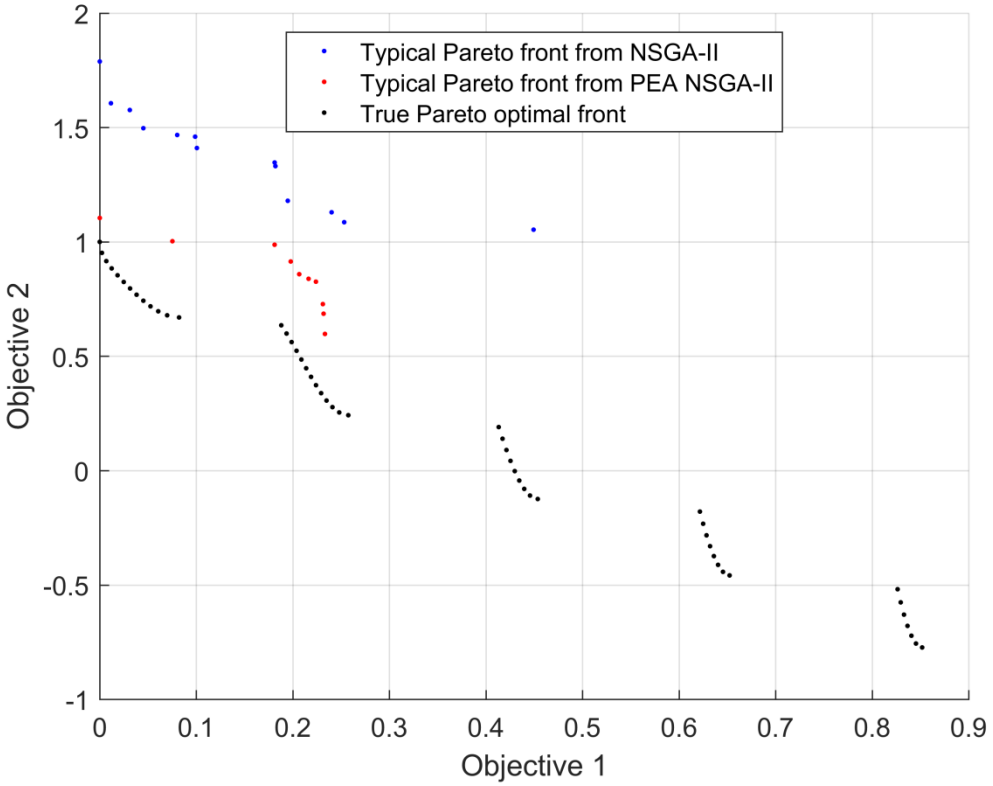

Figure 49: Typical Pareto fronts for the ZDT3 problem by the NSGA-II and the PEA NSGA-II algorithms after 1100 objective calculations

ZDT4 Problem

Table 10 and Table 11 demonstrate the values of the performance metrics (set convergence and hypervolume indicator) for ZDT4 optimization problem every 100 objective calculations.

Table 10: PEA NSGA-II results for the ZDT4 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.3548	0.0575	128.2643	71.5093
200	0.5255	0.0695	84.4790	10.5203
300	0.5771	0.0804	73.4152	12.5540
400	0.6078	0.0857	68.4086	11.8265
500	0.6366	0.0738	65.0280	10.0493
600	0.6404	0.0734	62.2351	10.7522
700	0.6525	0.0780	60.4197	10.7980
800	0.6646	0.0883	58.4014	11.8273
900	0.6754	0.0798	56.3030	11.6626
1000	0.6828	0.0771	54.5416	12.1086
1100	0.6863	0.0715	53.7078	11.3185

Table 11: NSGA-II results for the ZDT4 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.3566	0.0490	119.0563	43.1288
200	0.5150	0.0812	83.6560	12.0979
300	0.5566	0.0780	76.4200	12.9886
400	0.5982	0.0766	68.9573	14.0872
500	0.6124	0.0744	65.5943	12.2201
600	0.6308	0.0673	62.8785	11.2561
700	0.6399	0.0656	60.8432	10.8661
800	0.6440	0.0630	59.8745	10.5915
900	0.6461	0.0622	59.1446	10.0120
1000	0.6536	0.0611	57.3823	9.8624
1100	0.6618	0.0569	55.8345	9.4049

The performance metrics values shown in Table 12 and Table 13are also in illustrated graphically in Figure 50 and Figure 51:

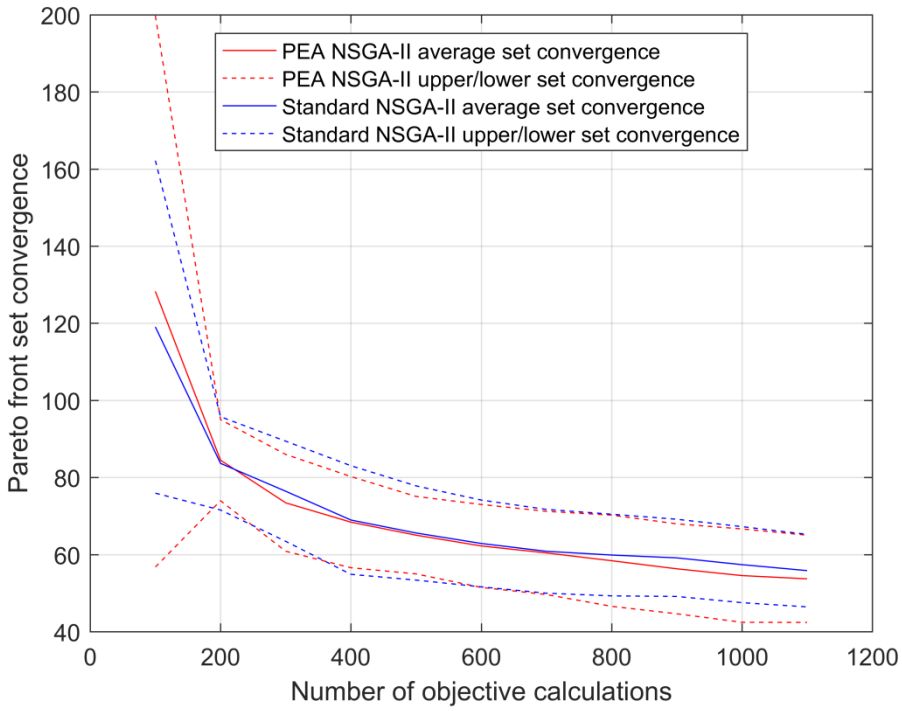

Figure 50: ZDT4 set convergence graph

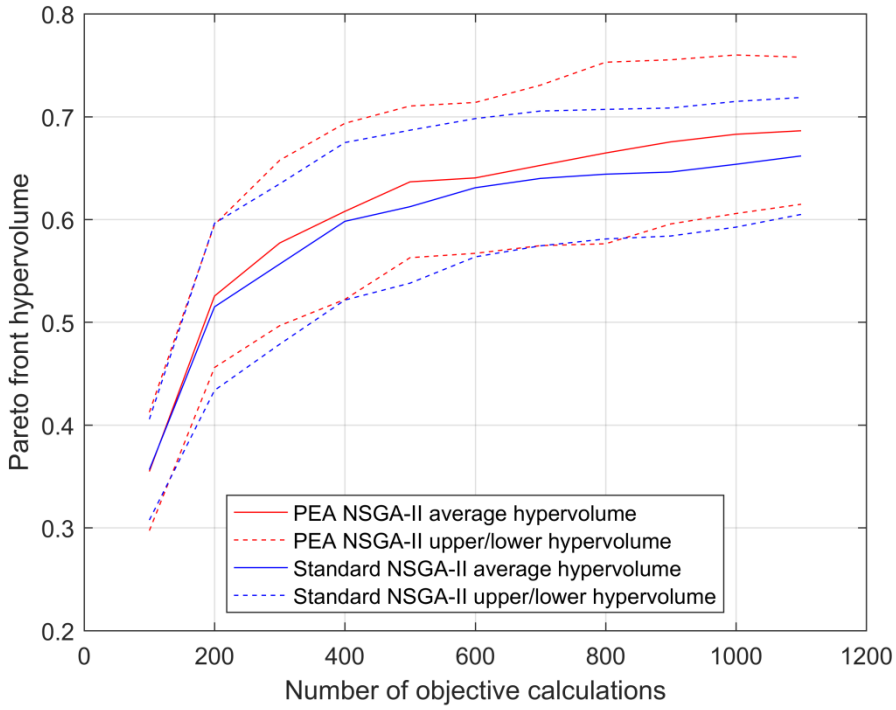

Figure 51: ZDT4 average hypervolume graph

For clarity reasons the performance metrics values from every independent optimization run are presented in Figure 52 and Figure 53.

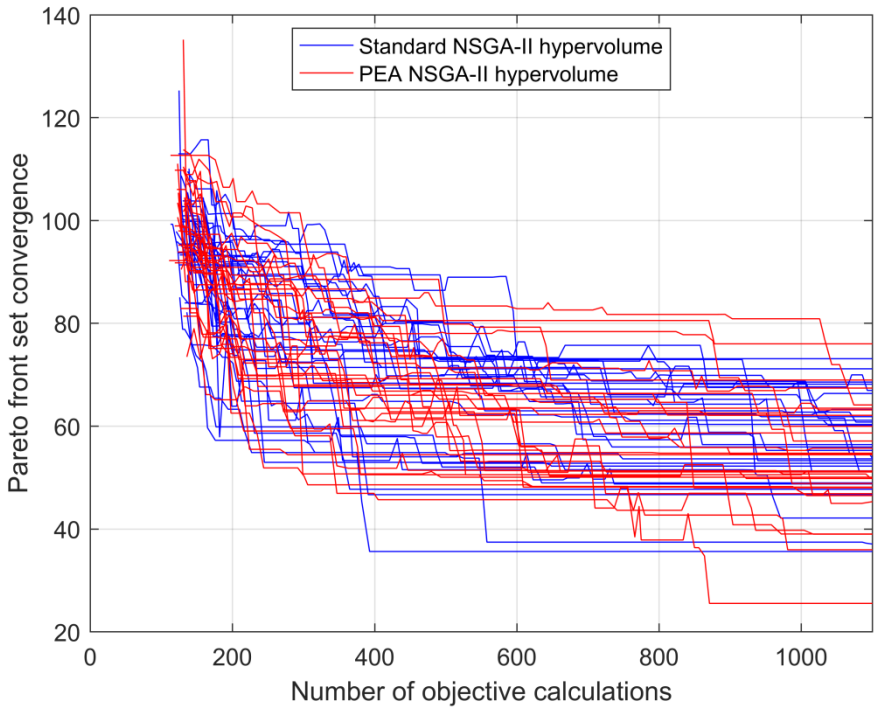

Figure 52: ZDT4 set convergence results for 30 independent runs

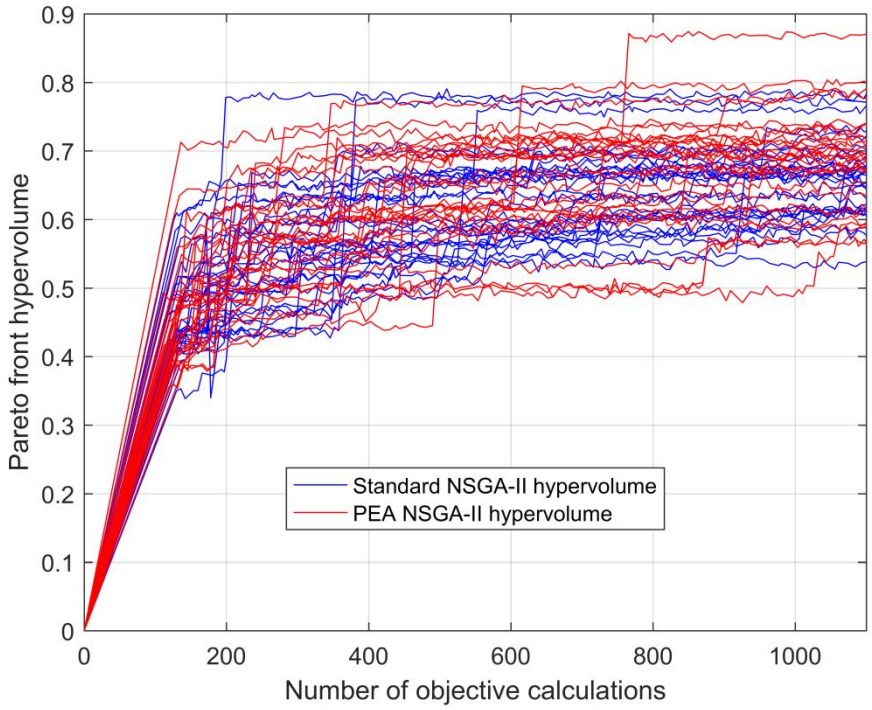

Figure 53: ZDT4 hypervolume indicator results for 30 independent runs

Figure 54 illustrates a typical Pareto front at 1100 objective calculations. The Pareto fronts chosen for graphical representation are the fronts with the absolute minimum difference in performance metrics values from the average values at 1100 objective calculations. This allows the reader to understand how different performance metrics values correspond to different fronts.

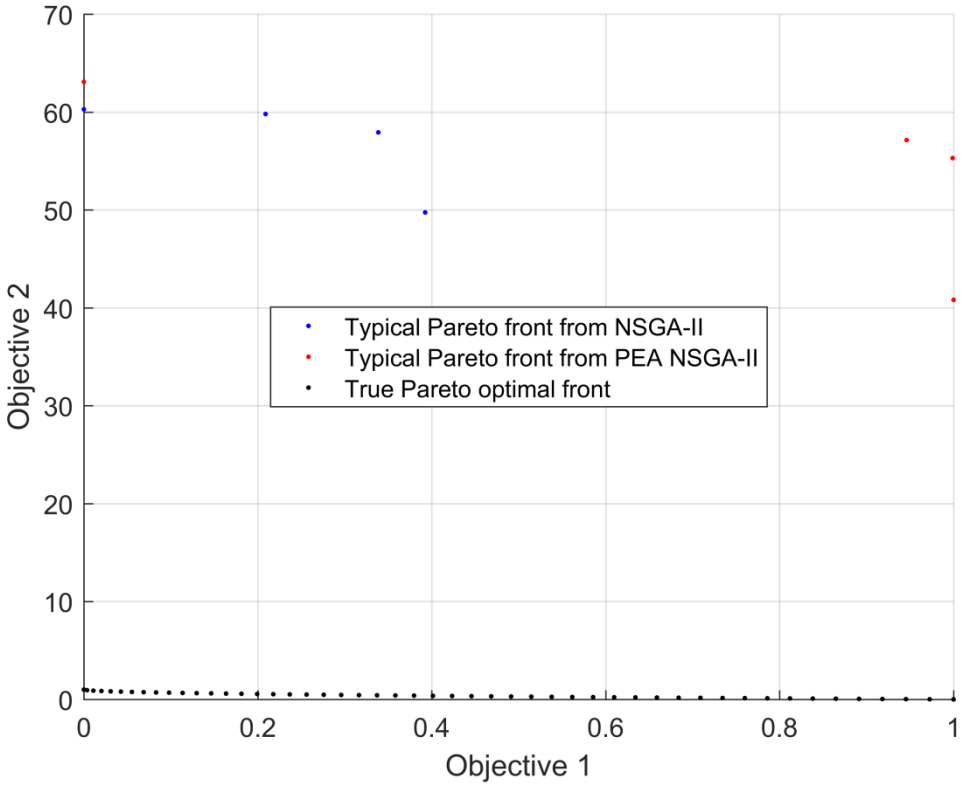

Figure 54: ZDT4 typical Pareto front after 1100 objective calculations

ZDT6 Problem

Table 12 and Table 13 demonstrate the values of the performance metrics (set convergence and hypervolume indicator) for the ZDT6 optimization problem every 100 objective calculations.

Table 12: PEA NSGA-II results for the ZDT6 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.07772	0.01493	7.00300	0.39293
200	0.14396	0.02494	6.38610	0.27681
300	0.19311	0.03045	5.83616	0.35613
400	0.24098	0.04581	5.34590	0.51163
500	0.30606	0.09102	4.70238	1.00694
600	0.38751	0.11404	3.69299	1.31310
700	0.48562	0.12949	2.53351	1.50129
800	0.56766	0.12189	1.46191	1.41124
900	0.61152	0.10276	0.85772	1.11342
1000	0.63639	0.08170	0.55419	0.84365
1100	0.65443	0.06160	0.43362	0.56534

Table 13: NSGA-II results for the ZDT6 problem

Objective calculations	Hypervolume indicator		Set convergence	
	Average	Standard deviation	Average	Standard deviation
100	0.08788	0.01790	6.88123	0.40163
200	0.14007	0.02825	6.46413	0.29523
300	0.17462	0.04105	6.12823	0.37282
400	0.21806	0.05444	5.71330	0.49962
500	0.26357	0.06894	5.24007	0.71507
600	0.32906	0.09902	4.68045	0.85242
700	0.41757	0.12311	3.89673	1.22388
800	0.49183	0.14037	3.00375	1.62196
900	0.55484	0.14595	2.19746	1.77949
1000	0.59352	0.12595	1.55263	1.57860
1100	0.61916	0.10646	0.98320	1.10427

The performance metrics values shown in Table 12 and Table 13are also in illustrated graphically in Figure 55 and Figure 56:

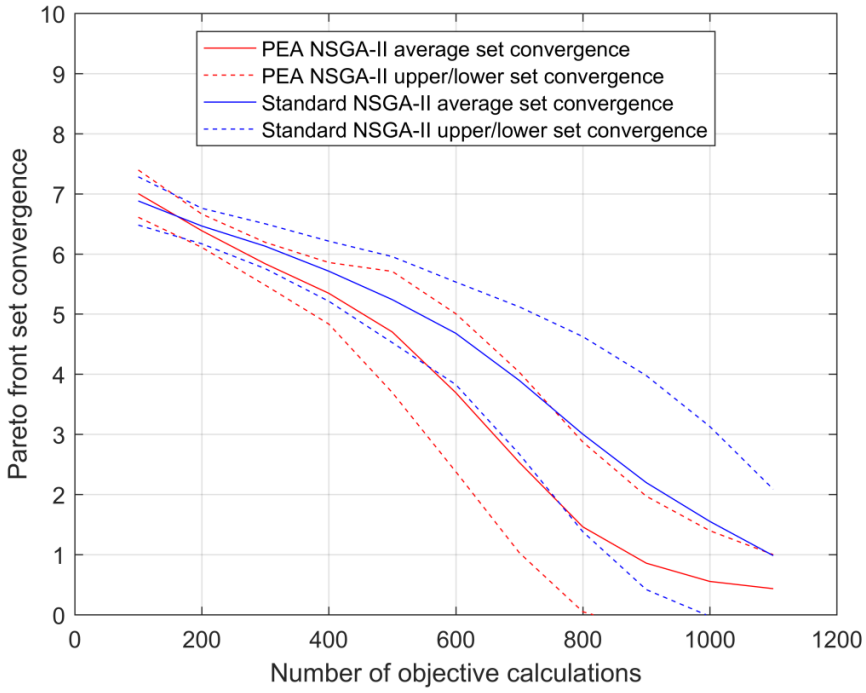

Figure 55: ZDT6 average set convergence graph

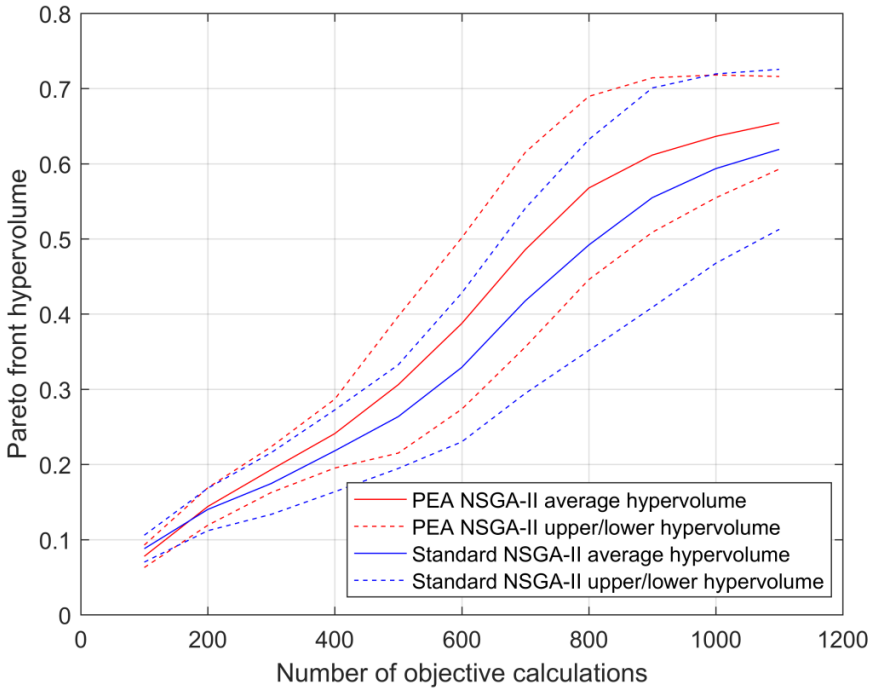

Figure 56: ZDT6 average hypervolume indicator graph

For clarity reasons the performance metrics values from every independent optimization run are presented in Figure 57 and Figure 58.

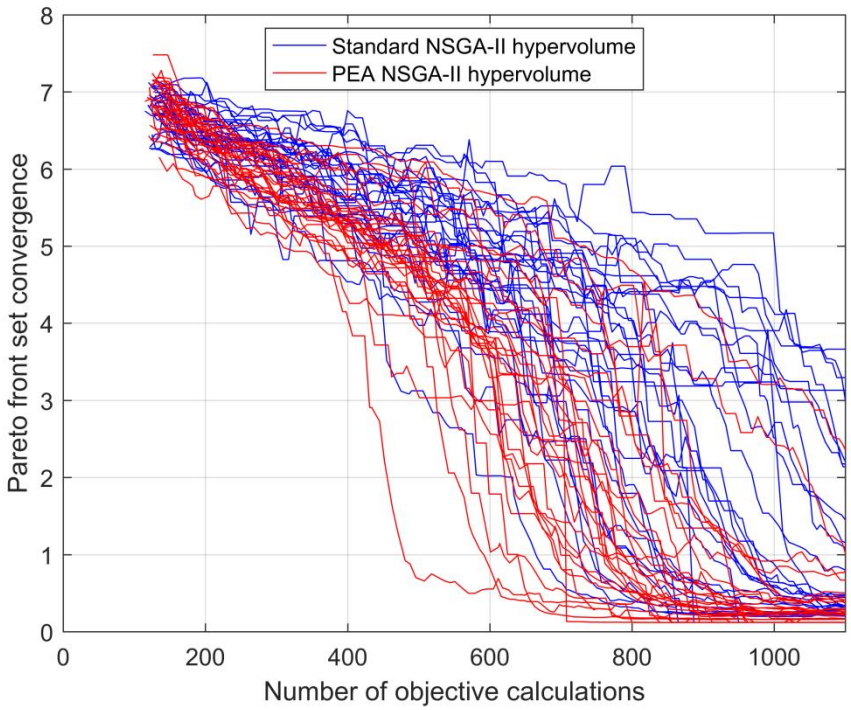

Figure 57: ZDT6 set convergence results for 30 independent runs

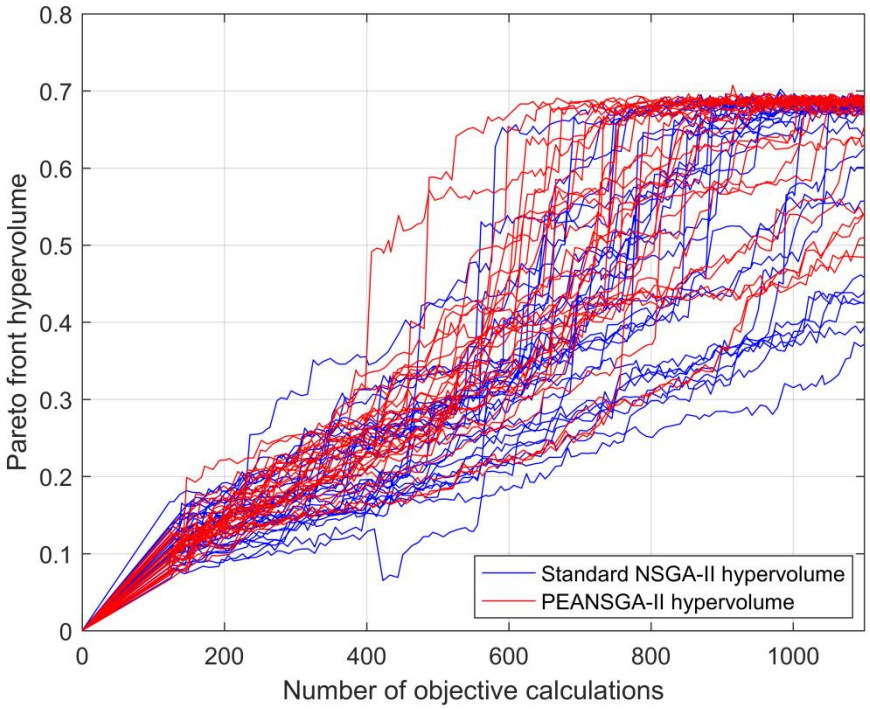

Figure 58: ZDT6 hypervolume indicator results for 30 independent runs

Figure 59 illustrates a typical Pareto front at 1100 objective calculations. The Pareto fronts chosen for graphical representation are the fronts with the absolute minimum difference in performance metrics values from the average values at 1100 objective calculations. This allows the reader to understand how different performance metrics values correspond to different fronts. Also it is clearly demonstrated that PEA NSGA-II outperforms NSGA-II in terms of Pareto front convergence for a given number of objective calculations.

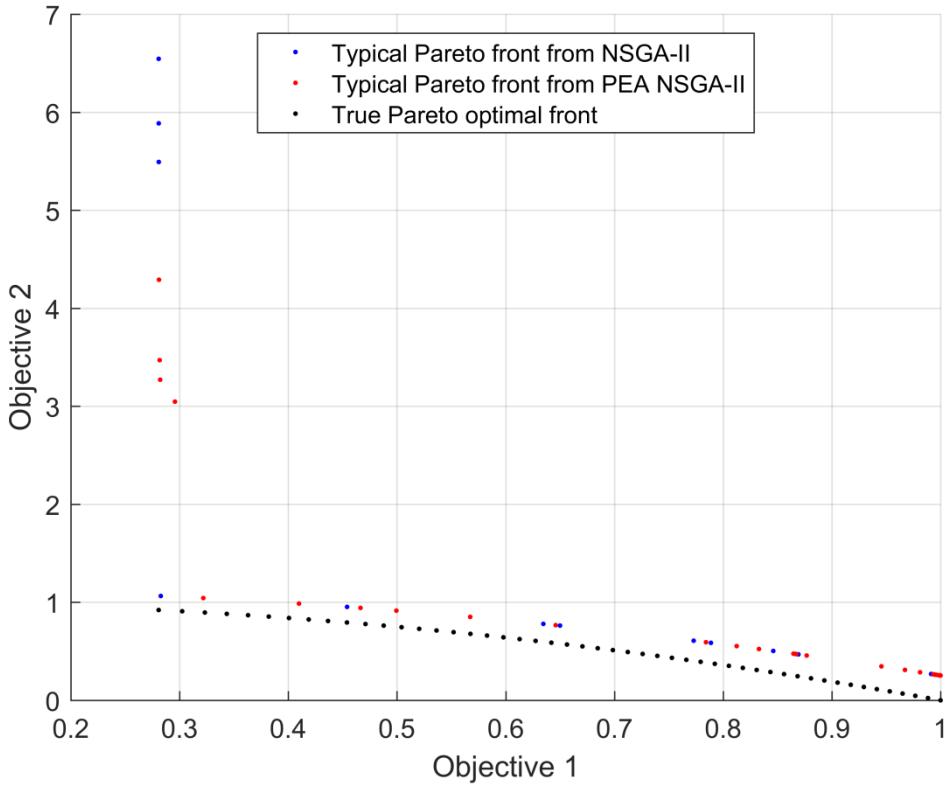

Figure 59: ZDT6 typical Pareto front after 1100 objective calculations

6.3.3 Conclusions

Overall, our work on the ZDT problems demonstrated that both the standard NSGA-II and PEA NSGA-II are performing well in a set of challenging optimization problems. Furthermore, it can be observed that PEA NSGA-II algorithm outperforms the standard NSGA-II algorithm in every optimization problem except ZDT4. In the following paragraphs the performance of the neural network and the optimization algorithms will be further discussed for every ZDT problem.

Overall it is clear that in every optimization problem the neural network was unable to predict the inverse objective functions successfully. However, it is not the goal of the PEA algorithm to predict the exact decision variables that result in the extrapolated points but rather train a neural network to understand the trends in the decision variables. Based on that, the neural network was able to predict correctly the trend of the inverse objective function regarding the decision variables values. As a result, the neural network was able to systematically find new individuals that would dominate previous non dominated solutions. One should also notice that, in many cases the solutions of the extrapolation algorithm gives the same objective 2 value. This odd behaviour can be explained by a combination of the following. Objective 2 is only dependant on the value of the first decision variable. Consequently, if the neural network predicts for two different extrapolated points the same value for the first decision variable then both extrapolated points would result in individuals that have the same objective 2 value. As a result, the extrapolation algorithm instead of predicting new uniformly distributed Pareto optimal individuals it predicted new individuals that share the same objective 2 value.

In the ZDT1 problem, NSGA-II was outperformed by the PEA NSGA-II algorithm at every point of the optimization procedure. At 1100 objective calculations the PEA NSGA-II algorithm dominates on average 2.1% more of the search space and has 4 times lower average distance from the true Pareto optimal front. PEA NSGA-II standard deviation is also less than the standard deviation of NSGA-II. As a result, PEA NSGA-II offers a more robust option for solving ZDT1 problem. The difference of performance can easily be observed in the typical Pareto fronts, shown in Figure 33 where the typical PEA NSGA-II Pareto front strictly dominated the typical NSGA-II Pareto front.

In the ZDT2 problem, NSGA-II was outperformed by PEA NSGA-II algorithm after 300 objective calculations objective calculations. At 1100 objective calculations the PEA NSGA-II algorithm dominates on average 0.1% more of the search space and has 1.57 times lower average distance from the true Pareto optimal front. The standard deviation of the performance metrics is also lower for the PEA NSGA-II algorithm. From that it can be concluded that the PEA algorithm is also improving the stability of the evolutionary algorithms. The difference of performance can easily be observed in the typical Pareto fronts, shown in Figure 44: where every solution but one found by the typical PEA NSGA-II Pareto front strictly dominates the solutions found by the typical NSGA-II.

In the ZDT3 problem, NSGA-II was again outperformed by PEA NSGA-II algorithm at any point during the optimization procedure after 200 objective calculations. In particular at 1100 objective calculations PEA NSGA-II dominates 0.43% more of the search space than standard NSGA II and the average Pareto front resulted from PEA NSGA-II had 4 times lower average distance than the average Pareto front resulted from NSGA-II. The standard deviation of the performance metrics is also lower for the PEA NSGA-II algorithm. The difference of performance can easily be observed in the typical Pareto fronts, shown in Figure 49 where the typical PEA NSGA-II Pareto front strictly dominated the typical NSGA-II Pareto front. The true Pareto optimal solutions of the problem are located in a set of discontinuous convex fronts. In order to examine the software ability to locate solutions across the entire true Pareto optimal front the problem was solved again with a limit of 800 generations (approximately 2000 objective calculations). The Pareto front of the abovementioned independent run is presented in Figure 60:

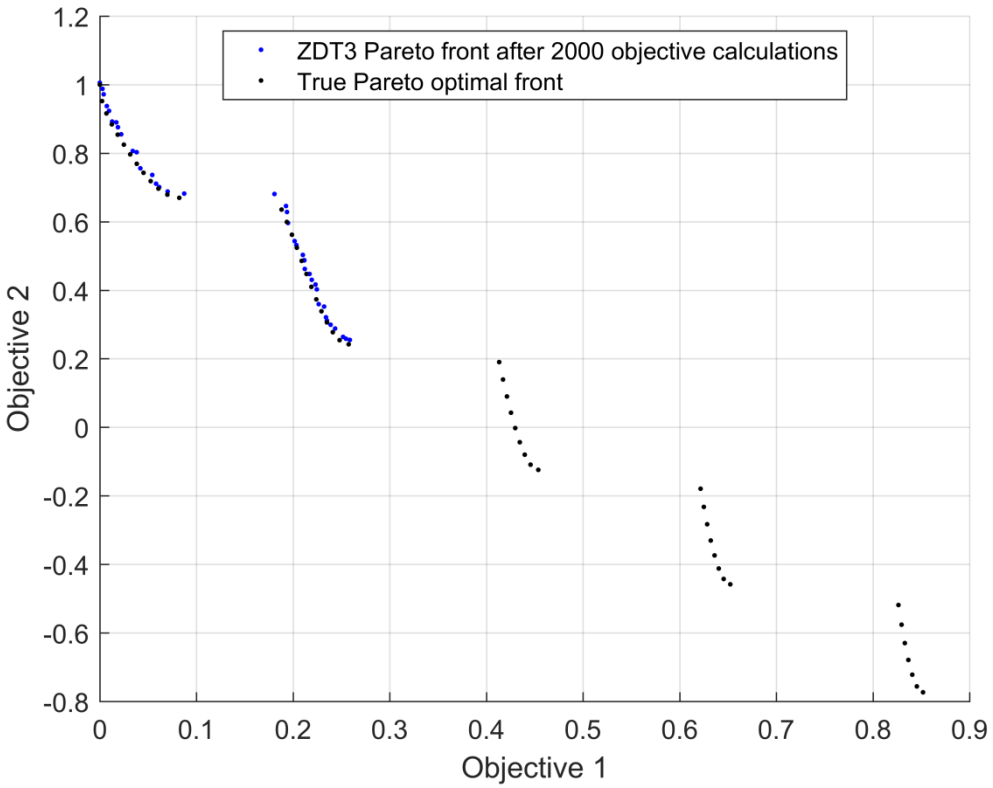

Figure 60: ZDT3 Pareto front after 2000 objective calculations

In the ZDT4 problem, NSGA-II was marginally outperformed by PEA NSGA-II. Although, the average values of the performance were higher for PEA NSGA-II the standard deviation of the metrics were high enough to show that the performance of both algorithm are performing similarly. This result is expected since the extrapolation feature is not an exploration feature. PEA thrives when the evolutionary algorithm can locate a number of Pareto dominant solutions. In ZDT4 most of the fronts have little amount of non-dominated solutions and thus PEA does not have sufficient number of points to extrapolate. Also in

ZDT4 the neural network has a hard time predicting the reverse function of the problem and thus adding new solutions with the extrapolation. The fact that both algorithms have similar performance can be observed from Figure 54 where the typical PEA NSGA-II Pareto front and the standard NSGA-II Pareto front have no clear relation between them.

In the ZDT6 problem, NSGA-II was again outperformed by PEA NSGA-II at any point of the optimization procedure after 200 objective calculations. In particular, PEA NSGA-II Pareto front after 1100 objective calculations dominates 0.04% more of the search space while having 2.27 times lower distance on average than standard NSGA-II. Simultaneously, standard deviation of the performance metrics is lower for PEA NSGA-II. As a result, PEA NSGA-II can achieve the abovementioned results in a more robust manner. It can be observed in Figure 58 that the hypervolume indicator increases steadily, up until it reaches 68%. This happens when the evolutionary algorithm finds the first solution with an objective 2 value near the minimum optimal value. The density of the solutions along the Pareto front is not constant and in particular it decreases when objective 1 value is reaching towards the maximum. Consequently, Pareto optimal solutions that have maximum objective 1 value are much scarcer and when found by the evolutionary algorithm they increase by a high amount the hypervolume indicator value. Figure 61 illustrates a Pareto front before and after the evolutionary algorithms finds a solution with an objective 2 value equal to 0.3 and an objective 1 value equal to 1.1.

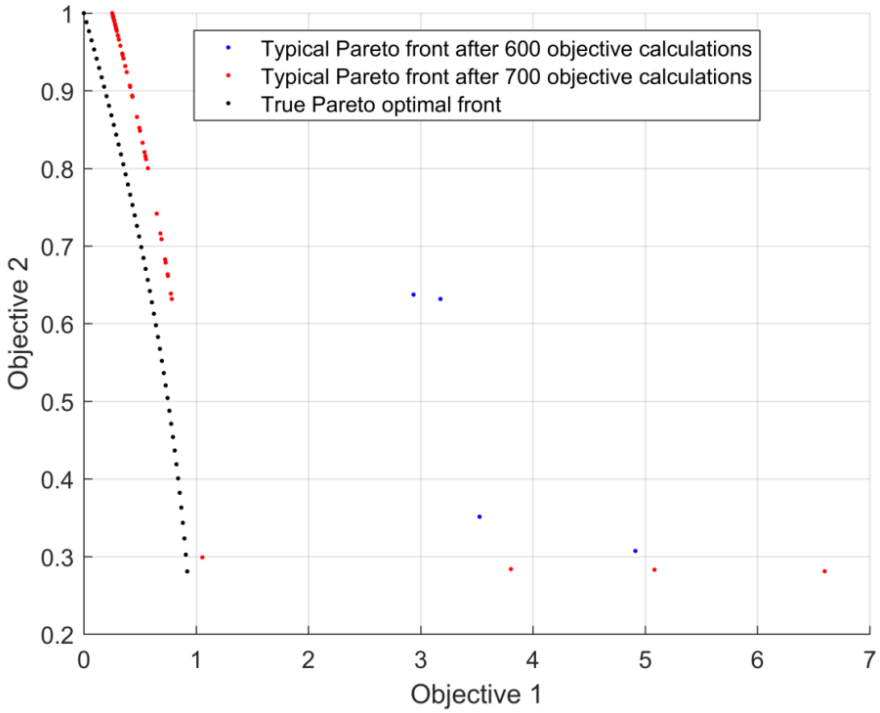

Figure 61: ZDT6 Pareto front after 600 and 700 objective calculations respectively

6.4. Optimal design of hydrophobic bearings

6.4.1. Hydrophobic bearing overview

The next optimization problems that have been solved with the optimization software developed within this thesis is the optimal design of the hydrophobic surface of a journal bearing. The optimization problem consists of minimizing the normalized friction coefficient and the dimensionless eccentricity ratio by selecting the optimal geometry for the hydrophobic surface. The design variables selected for the hydrophobic surface are:

1) Slip starting angle. This variable denotes the starting angle of the hydrophobic surface
2) Slip end angle. This variable denotes the ending angle of the hydrophobic surface
3) $Y_{slip\ start}$. This variable expresses the non-dimensional value of the bearing length where the hydrophobic surface starts.
4) $Y_{slip\ end}$. This variable expresses the non-dimensional value of the bearing length where the hydrophobic surface ends.

The decision variables of the optimization problem are also illustrated in Figure 62:

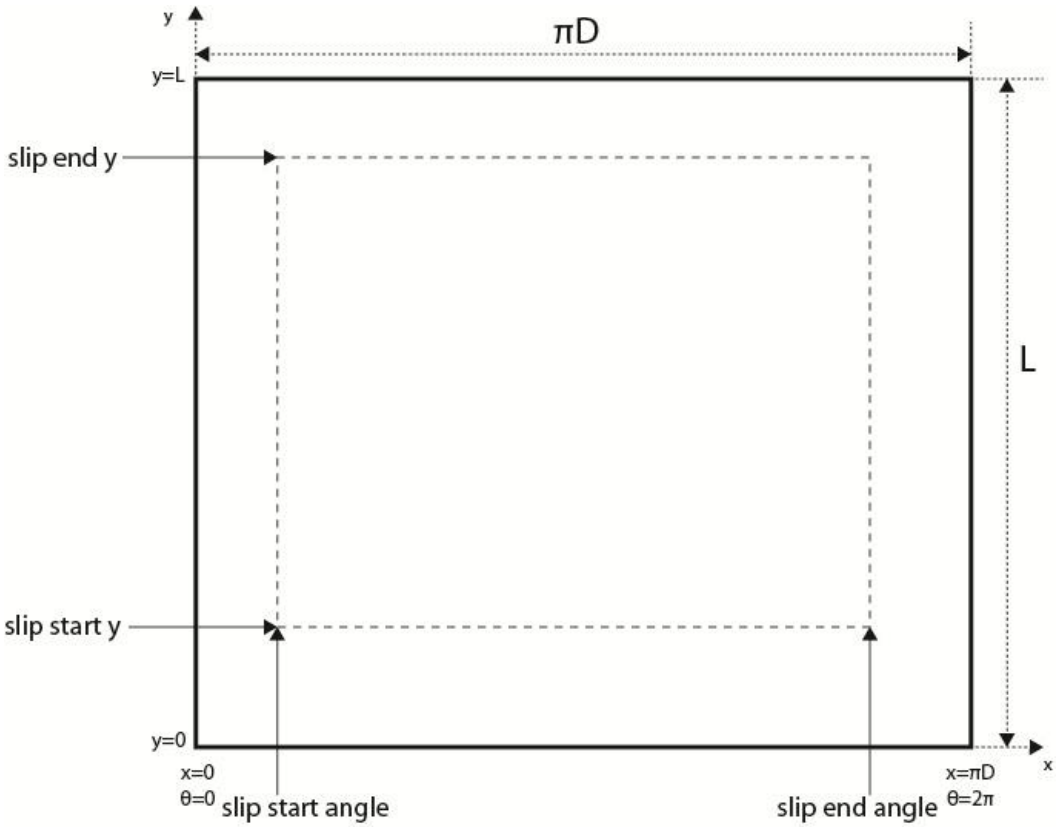

Figure 62: Geometry of journal bearing: Design variables for defining the hydrophobic part of the bearing

The decision variable bounds are shown in Table 14:

Table 14: Hydrophobic surface optimal design variable bounds

Decision variable	Lower bound	Upper bound
Slip starting angle	10	105
Slip end angle	105	200
$Y_{slip\ start}$	0.1	0.4
$Y_{slip\ end}$	0.6	0.9

The optimization problem will be solved for a journal bearing with fixed dimensions that is operating under three different conditions. The geometry and the operational parameters of the bearing are shown in Table 15

Table 15: Journal bearing geometry and operational condition

Case name:	Bearing case 1	Bearing case 2	Bearing case 3
RPM	1000	1000	1000
Applied load (vert.)[N]	200	500	1000
Bearing length[mm]	30	30	30
Bearing diameter[mm]	30	30	30
Radial clearance[mm]	0.05	0.05	0.05
Non-dimensional slip length[mm]	10	10	10

In this optimization problem the developed optimization software will select the decision variables and in house developed Bearing solver software will be used to calculate the values of the objectives. In case of an infeasible solution the optimization software assigns the value of 10^5 to both objectives.

The optimization parameters that will be selected to solve the problem are the following:

Both standard NSGA-II and PEA-NSGA-II will be used separately to solve the optimization problem. The evaluation of the Pareto fronts from both algorithms will result from 5 independent runs of every algorithm. The runs are limited to either 40 generations or 225 objective calculations whichever comes first.

NSGA II and PEA-NSGA II are tuned as follows:

NSGA-II parameters:

- Population size: 40
- Crossover rate 0.8
- Mutation probability 0.11

PEA parameters:

- Ev 8
- Ex 1.1
- Ag 3
- Angle 0.785 ($\pi/4$)

Reference points for calculating the hypervolume indicator:

- R=[0.489 4.13] for “Bearing case 1”
- R= [0.73 2.25]for “Bearing case 2”
- R=[0.85 1.46]for “Bearing case 3”

6.4.2. Optimization results

Due to the fact that the total selected numbers of independent optimization runs are 10 the results will not be interpolated and averaged. Instead the results will be presented for every run. This allows the reader to further understand the results of the optimization procedure and the comparison between PEA NSGA-II and standard NSGA-II. The present problem is a real world optimization problem thus no true Pareto optimal set of solutions is known in advance. As a result, the performance metric “set convergence” used in the ZDT problems will be omitted. As these sets of problems are “real world” optimization problems, it is expected that Pareto fronts will have clear relations between them. However, the final Pareto fronts are not sufficient information for evaluating the performance of an optimization algorithm, especially in “real world” optimization problems. The hypervolume indicator provides information about the initial point of the optimization and the optimization convergence rate. Because of that, the first metric that will be presented in those problems is the hypervolume indicator. Following the hypervolume indicator, the final Pareto front will be presented in two different ways. The first form of presenting the Pareto front is by creating a super-font consisting of all the non-dominated solutions from all the 5 independent runs. The second form is presenting all the Pareto fronts from every independent run of the problems for clarity reasons.

Hydrophobic bearing: Case 1

The first performance metric that will be presented for “Hydrophobic bearing: Case 1” is the hypervolume indicator.

In Figure 63 the hypervolume indicator for every independent run is presented:

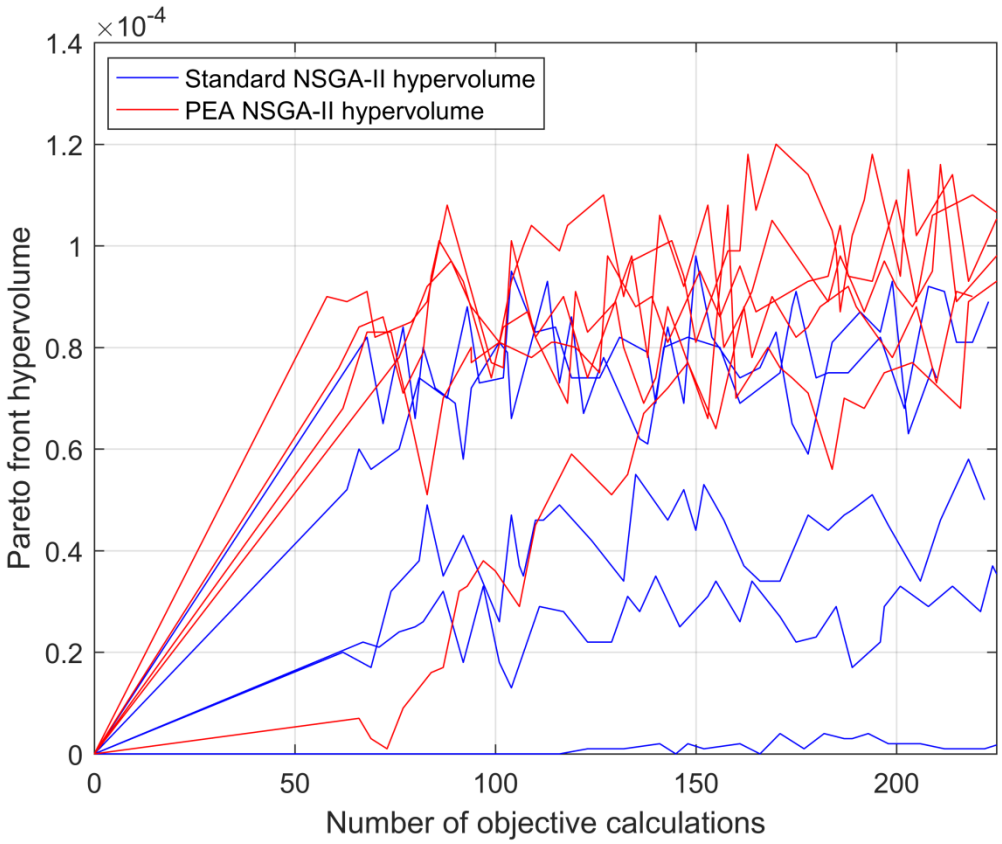

Figure 63: “Hydrophobic bearing: case 1” hypervolume indicator for 10 independent runs

In Figure 64 the super front of standard NSGA-II and PEA NSGA-II is presented:

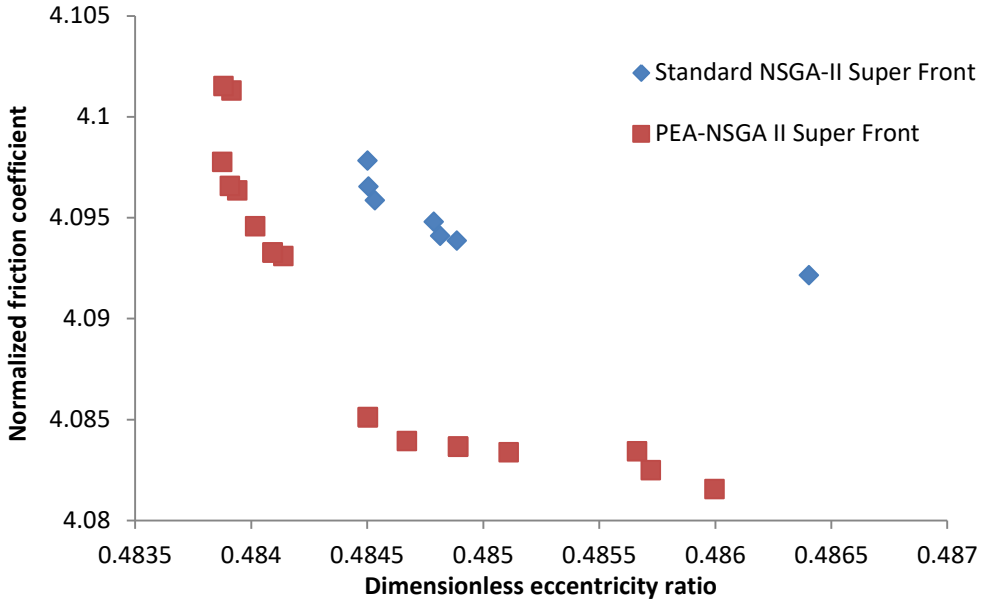

Figure 64: "Hydrophobic bearing: case 1"super-Pareto fronts.

The super front of ""Hydrophobic bearing: case 1"" arises from the Pareto fronts shown in Figure 65

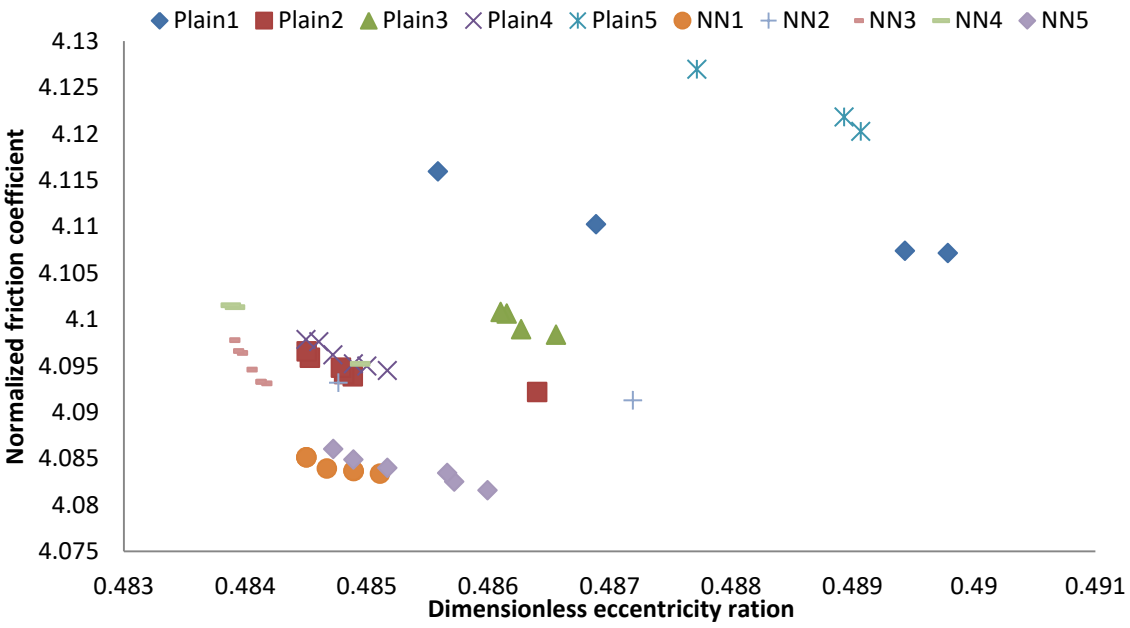

Figure 65: "Hydrophobic bearing: case 1" Pareto fronts from 5 independent runs with PEA NSGA-II and standard NSGA-II

Hydrophobic bearing: Case 2

The first performance metric that will be presented for “Hydrophobic bearing: Case 2” is the hypervolume indicator.

In Figure 66 the hypervolume indicator for every independent run is presented:

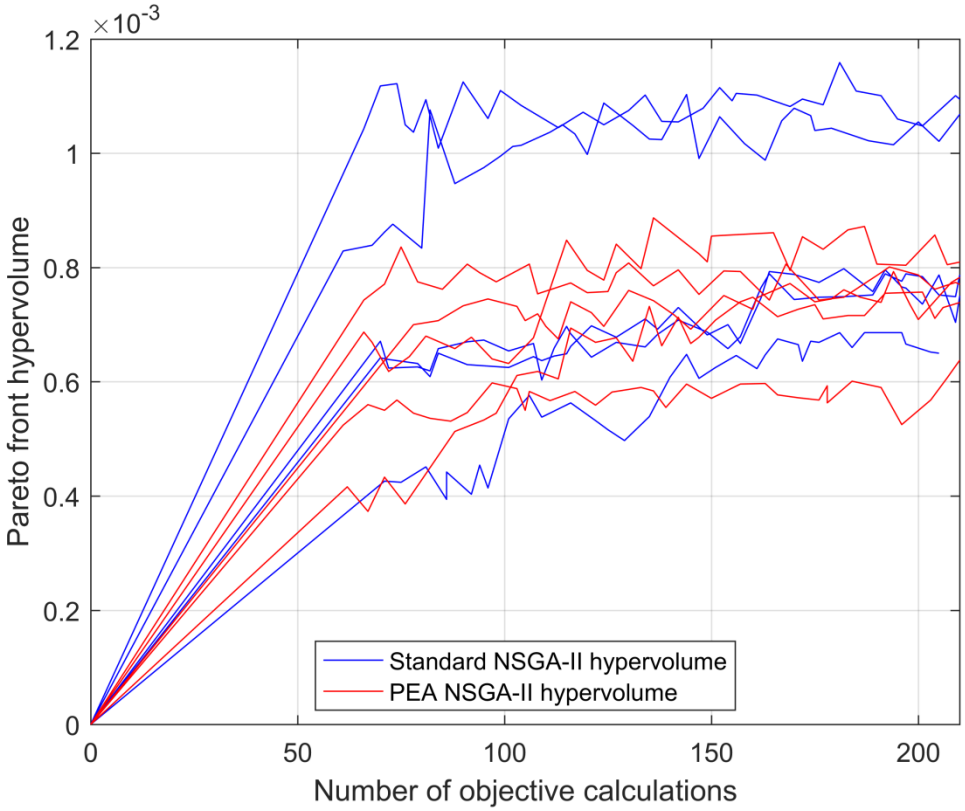

Figure 66: “Hydrophobic bearing: case 2” hypervolume indicator for 10 independent runs

In Figure 67 the super front of standard NSGA-II and PEA NSGA-II is presented:

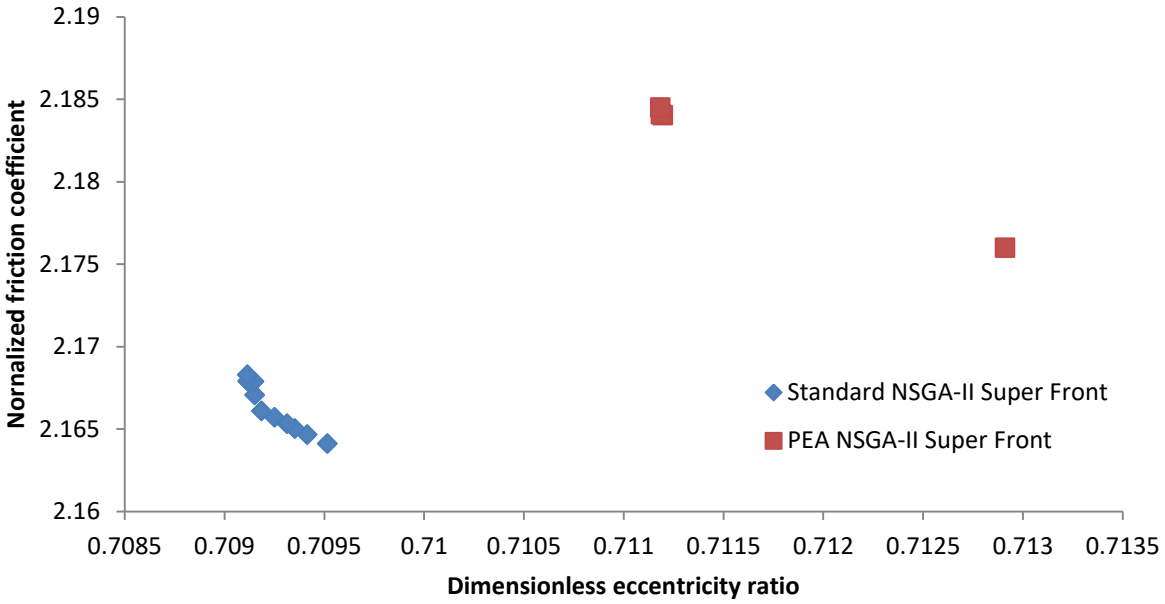

Figure 67: "Hydrophobic Surface 2" super-Pareto fronts.

The super front of "Hydrophobic bearing: case 2" arises from the Pareto fronts shown in Figure 68

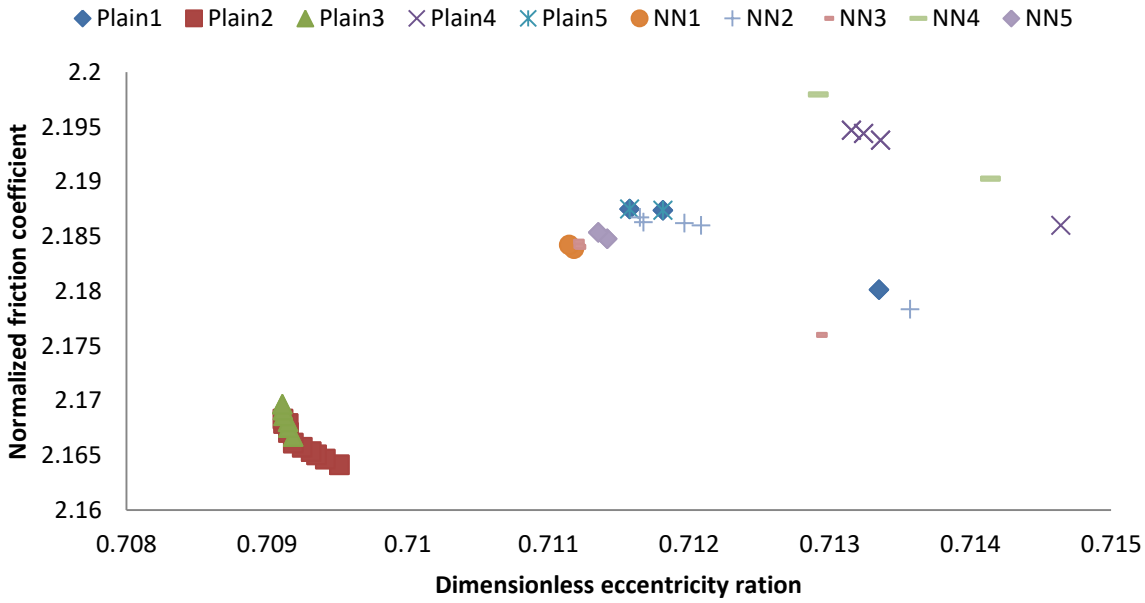

Figure 68: "Hydrophobic bearing: case 2" Pareto fronts from 5 independent runs with PEA NSGA-II and standard NSGA-II

Hydrophobic bearing: Case 3

The first performance metric that will be presented for “Hydrophobic bearing: Case 3” is the hypervolume indicator.

In Figure 69 the hypervolume indicator for every independent run is presented:

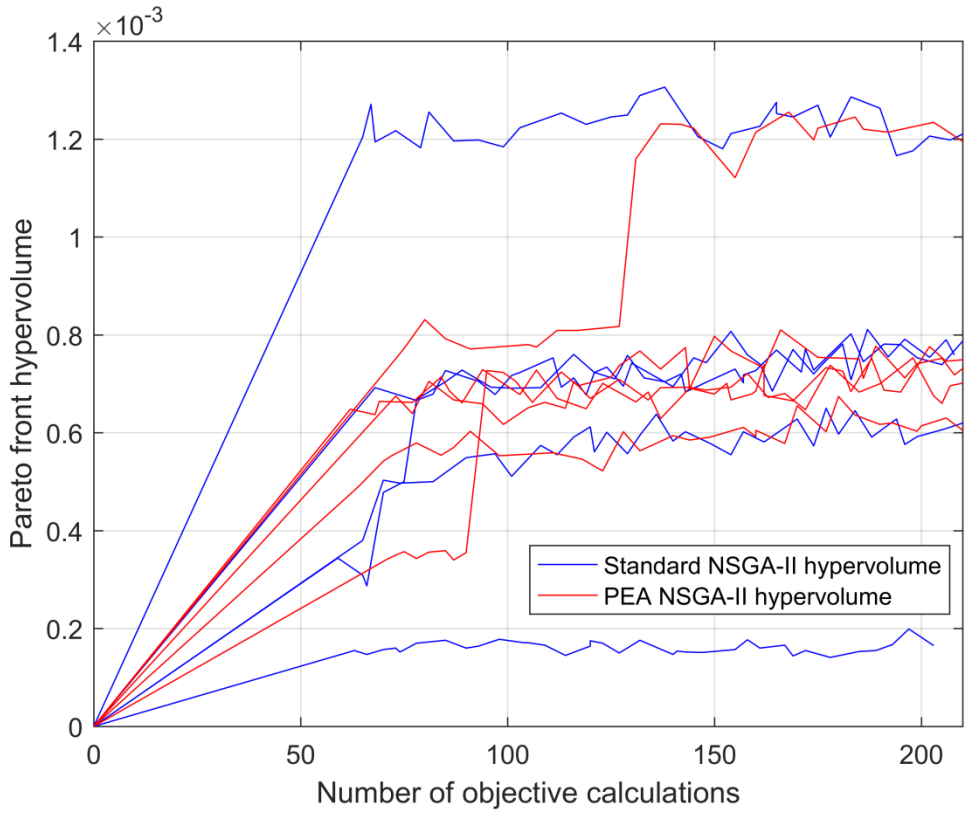

Figure 69: “Hydrophobic bearing: Case 3” hypervolume indicator for 10 independent runs

In Figure 70 the super front of standard NSGA-II and PEA NSGA-II is presented:

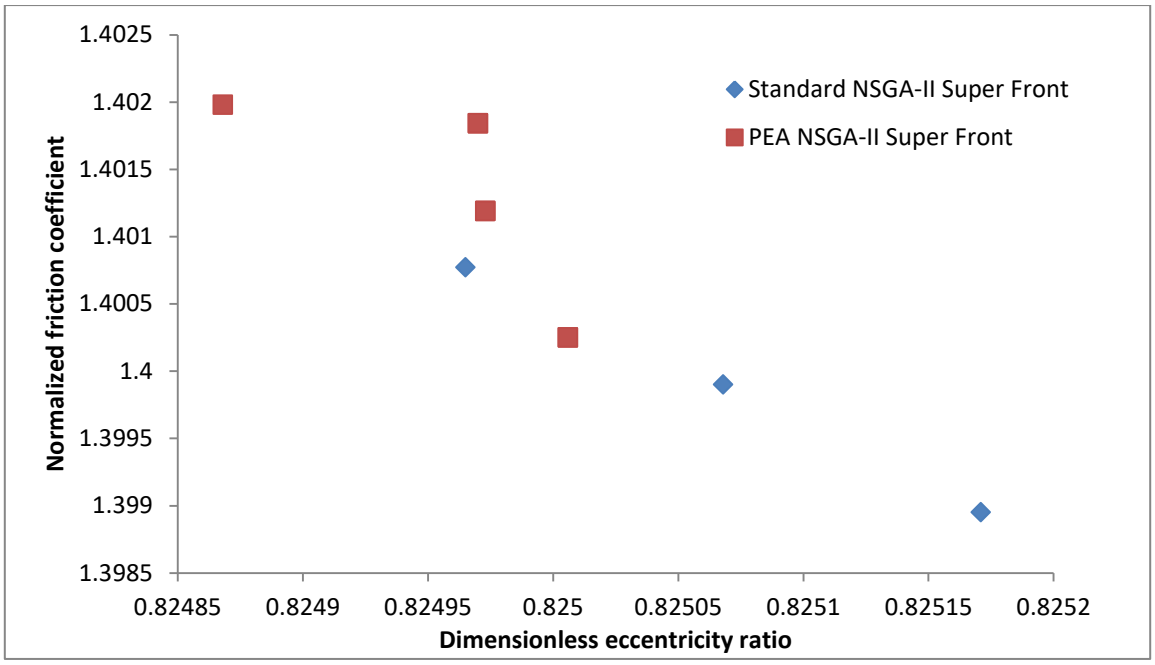

Figure 70: "Hydrophobic bearing: Case 3" super-Pareto fronts

The super front of "Hydrophobic bearing: Case 3" arises from the Pareto fronts shown in Figure 71

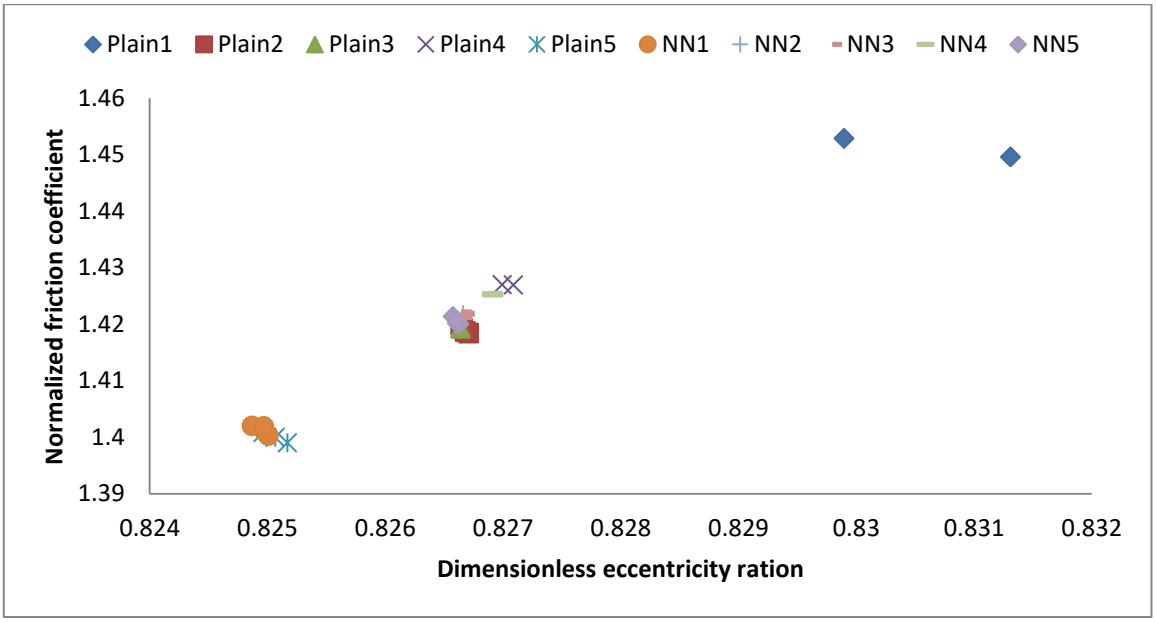

Figure 71: "Hydrophobic bearing: Case 3" Pareto fronts from 5 independent runs with PEA NSGA-II and standard NSGA-II

6.4.3. Conclusions

Overall, optimization results for the hydrophobic bearing shows that both the standard NSGA-II and PEA NSGA-II are performing well in a set of “real world” optimization problems. Furthermore, it can be observed that PEA NSGA-II algorithm outperforms the standard NSGA-II algorithm in every optimization problem.

In problem “Hydrophobic bearing: Case 1” PEA NSGA-II managed to outperform standard NSGA-II in every independent run. In particular, in Figure 64 it can easily be observed that PEA NSGA-II super Pareto front clearly dominates the standard NSGA-II super front. The PEA NSGA-II super Pareto front is both more diverse than the respective NSGA-II front at similar amount of objective calculations. The findings of the super Pareto fronts are also in accordance with the independent Pareto fronts shown in Figure 65, where the worst Pareto front arising from PEA NSGA-II is better than the best Pareto front arising from standard NSGA-II. Additionally, the hypervolume indicator presented in Figure 63 shows that optimization runs that start from similar starting points result in better Pareto front for PEA NSGA-II

Problem “Hydrophobic bearing: Case 2” is the first case where standard NSGA-II and PEA NSGA-II perform equally. In Figure 67 it can be observed that NSGA-II super Pareto front strictly dominates PEA NSGA-II super Pareto front. In this problem the initial population is something that PEA NSGA-II cannot overcome. Without any further knowledge of the topology of the objective space no conclusions can be drawn on why the initial population choice is such a critical factor. Quite possibly, a high number of local Pareto front are located in the objective and given the limited amount of objective calculations NSGA-II is unable to escape them.

Similar findings with those of “Hydrophobic bearing: Case 3” can be observed at second case as well. In this case as it can be observed at Figure 70 the super Pareto fronts corresponding to standard NSGA-II and PEA NSGA-II are incomparable. In this case, as well, the initial population choice is a parameter that NSGA-II cannot overcome even with the assistance of the Pareto extrapolation algorithm. However, in this case as it can be observed in Figure 69 there are some cases where successful mutation operation let NSGA-II from the local Pareto fronts. Those successful mutations are bound to happen in a long enough timeline. However, with the timeframe that was set for this optimization solution NSGA-II was not able to perform them.

6.5. Geometry Optimization of Three-Dimensional Micro-Thrust Bearings

6.5.1. Overview

The last optimization problem that has been solved in the present thesis is the optimization of the geometry of textured three-dimensional micro thrust bearings. The optimization problem has been implemented with the same parameters as it was originally proposed by Papadopoulos et al. (2011), for validation purposes; for more information on the modelling and how the optimization problem was defined, the reader should refer to the aforementioned paper. The optimization consists of minimizing the convergence ratio while maximizing the non-dimensional loading carrying capacity of the bearing. The geometry of a textured micro-thrust bearing is presented in Figure 72.

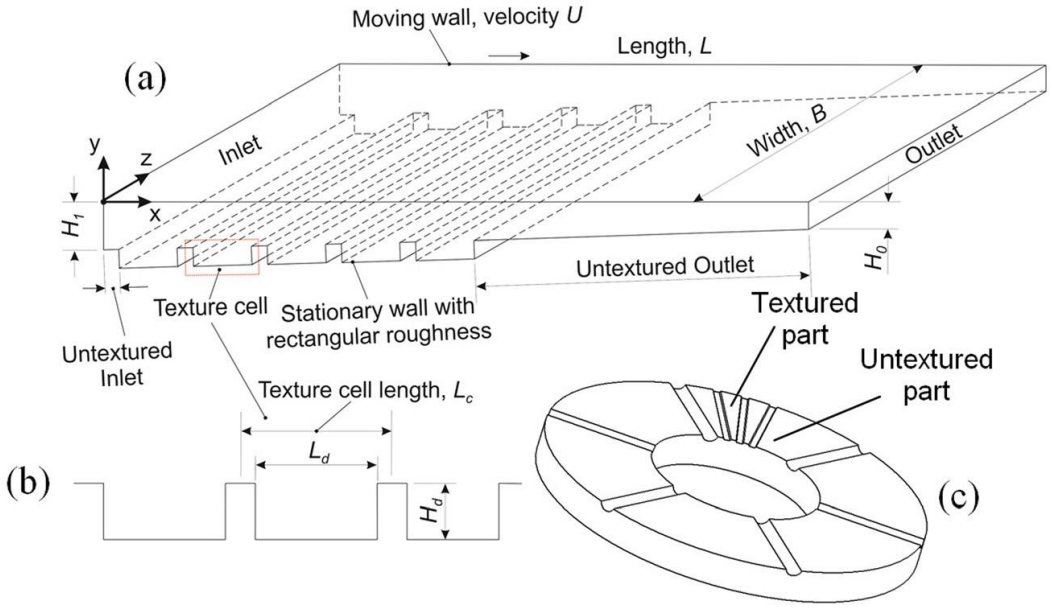

Figure 72: Micro-thrust bearing geometry. (a) Three-dimensional textured converging slider geometry (b) Geometry of dimples. (c) Typical thrust bearing application with partial texturing.

Source: (Papadopoulos et al., 2011)

To solve the optimization problem the following decision variables controlling the geometry are selected:

1. *Convergence ratio,* denoted as *k,* which is responsible for controlling the height of the bearing, and it is defined as $k = (H_1 - H_0)/H_0$.
2. *Non-dimensional un-textured length,* denoted as l_{uo} ,which expresses the portion of the bearing length that is non-textured, and it is defined as $l_{u0} = L_{u0}/L$
3. *Relative dimple length,* denoted as *s,* which is defined as $s = H_d/H_{min}$

The values of the objectives are calculated with the CFD tool ANSYS CFX. The ANSYS

model as well as the CAD files of the geometry files was implemented by the authors of the paper (Papadopoulos et al., 2011).

Decision variables and their respective constrains are presented in Table 16

Table 16: Three dimensional micro-thrust bearing design variables bound

Decision variable	Lower bound	Upper bound
k	-0.4	2
l_{uo}	0.2	0.7
s	0.1	2

The optimization software will select the decision variables and, as stated earlier, ANSYS CFX solver will be used to assign the values of the objectives. In case of an infeasible solution the optimization software assigns the value of 10^{12} to both objectives.

The optimization parameters that will be selected to solve the problem are the following:

Both standard NSGA-II and PEA-NSGA-II will be used separately to solve the optimization problem. The evaluation of the Pareto fronts from both algorithms will result from a representative run of every algorithm. The runs are limited to either 50 generations or 190 objective calculations whichever comes first.

NSGA II and PEA-NSGA II are tuned as follows:

NSGA-II parameters:

- Population size: 30
- Crossover rate 0.8
- Mutation probability 0.1

PEA parameters:

- Ev 5
- Ex 1.1
- Ag 4
- Angle 0.785 ($\pi/4$)

Reference point used for calculating the hypervolume indicator is [10 12].

6.5.2. Optimization Results

In Figure 73, the hypervolume indicator against the number of objectives are presented.

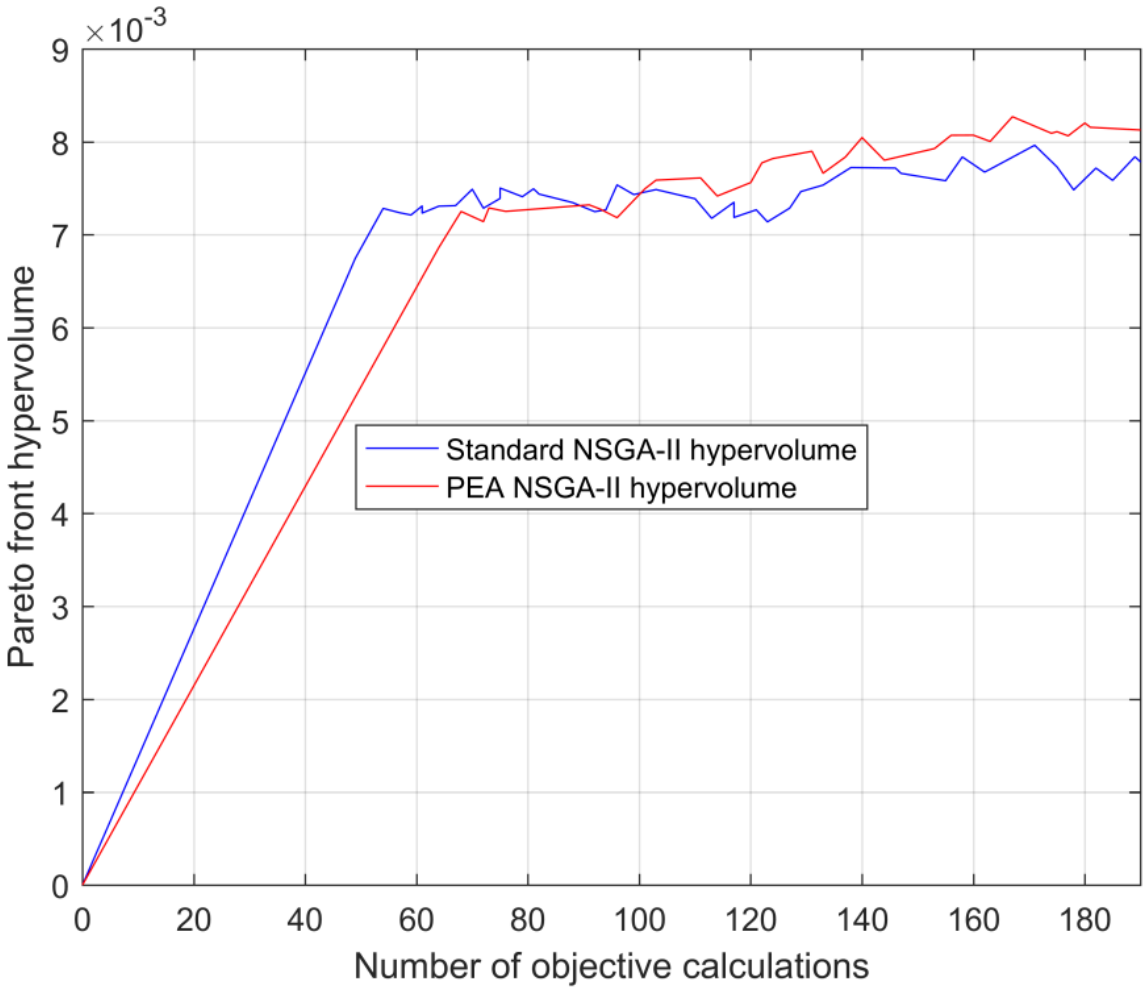

Figure 73: Optimization of three dimensional micro-thrust bearing Hypervolume indicators

The hypervolume indicator after 190 objective calculations corresponds to the Pareto fronts shown in Figure 74.

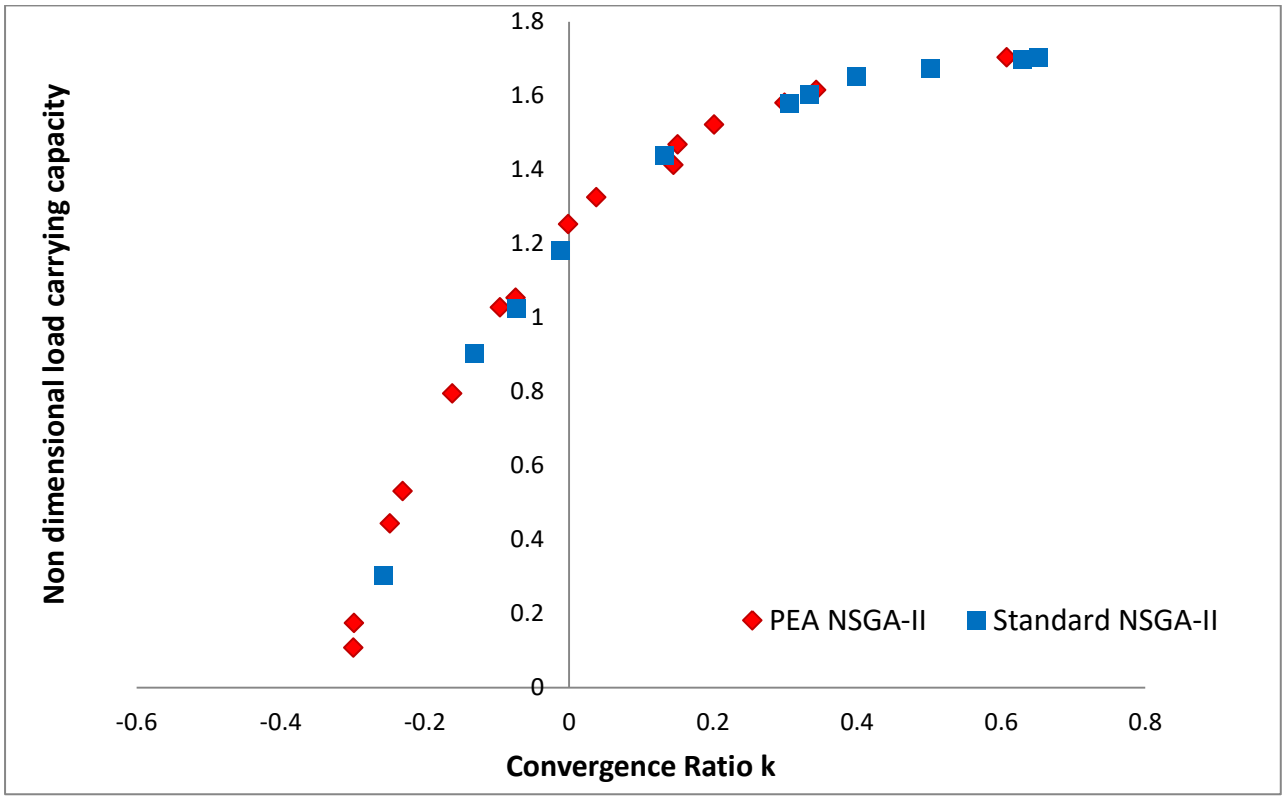

Figure 74: Optimization of three dimensional micro-thrust bearing Pareto fronts

6.5.3. Conclusions

The optimization runs at the previous section demonstrate that both standard NSGA-II and PEA NSGA-II have a robust performance when solving complex CFD problems. The first conclusion that can be drawn from the results is that the integrated ANSYS solver is able to connect the optimization software with ANSYS workbench and optimize ANSYS models. In terms of performance, it can be concluded that the developed optimization software (PEA NSGA-II) in this run although had a weaker initial selection of starting population, it managed to catch up with the standard NSGA-II algorithm and eventually surpass it with respect to convergence. The hypervolume indicator conclusions are easily verified by the corresponding Pareto fronts (Figure 74) where the PEA NSGA-II Pareto front is not only dominating in most of the areas but also is out-diversifying the Pareto front of standard NSGA-II. The Pareto fronts show that after 190 objective calculations NSGA-II almost found the Pareto front of the problem while PEA NSGA-II had already found it and also had spare objective calculations to diversify it. The results of this section cannot be considered as hard proof due to the little amount of independent runs. However, the results show a clear trend and a great potential for Pareto Extrapolation Algorithm.

Chapter 7: Conclusions and Future Work

In this thesis a general purpose optimization software using evolutionary algorithms has been developed. The software provides a user friendly interface in order to assist user unfamiliar users, with the concepts of evolutionary optimization to tune the optimization and optimize their models. Additionally, the interface of the software provides the user with real time graphical representation of the optimization procedure. The user can at runtime monitor the latest Pareto front and control the flow of the optimization. The user is also able to plot multiple Pareto fronts and examine the different Pareto fronts after the optimization process has been completed.

The other major goal of this thesis was to ensure that the optimization software developed is competitive in terms of performance, with other commercial or open source optimization software. This goal although ambitious, was achieved at satisfactory degree, with the introduction of the Pareto Extrapolation Algorithm (PEA) which is capable of identifying new non-dominated solutions by inversing and interpolating the objective functions of the optimization problem.

The evolutionary algorithms implemented for the software were tested in a series of nine challenging optimization problem with the goal of evaluating their performance and their robustness. In particular, in the family of ascending difficulty ZDT problems both standard NSGA-II and Pareto Extrapolation Algorithm (PEA) enhanced NSGA-II (developed in the course of the present thesis) performance was evaluated. Both algorithms managed to achieve satisfactory convergence rate with a low number of objective calculations (approximately 1100). Regarding, their compared performance, PEA NSGA-II managed to enhance the performance of standard NSGA-II in every optimization problem. In particular, PEA NSGA-II managed to speed up the convergence rate of the standard NSGA-II by 1.6 to 4 times in the ZDT problems. . The performance of the PEA shows that the algorithm even in a suboptimal form can improve drastically the optimization convergence rate. An optimization of the Pareto extrapolation algorithm parameters, although not in the scope of this thesis is expected to further improve its performance.

Additionally, the present optimization software was used to tackle the problem of design optimization of a hydrophobic journal bearing under three different operating conditions. In the first operating condition, (200KN vertical load), PEA NSGA-II managed to outperform standard NSGA-II in every of the 5 independent runs. In the other two cases the topology of the objective space limited NSGA-II and consequently PEA NSGA-II ability to converge in the short timeframe of 40 generations. As a result NSGA-II even when assisted with PEA did not manage to improve substantially the original Pareto front that was obtained after the calculations of the original population.

The final optimization problem solved with the present optimization software was the optimization of the geometry of a textured micro-thrust bearing. In order to solve this

optimization problem, the integrated ANSYS solver was utilized. In this optimization problem PEA NSGA-II managed to outperform standard NSGA-II. Although, the amount of independent runs is not sufficient to be considered as strong evidence of performance, it shows a promising trend regarding the Pareto Extrapolation Algorithm performance. PEA NSGA-II managed to outperform standard NSGA-II in this problem even when starting with a worse initial population. In particular PEA NSGA-II was about 40 objective calculations ahead of the standard NSGA-II, and to put this in to perspective, 40 objective calculations consist of the 21% of the total objective calculations.

Although the optimization software managed to satisfy its two fundamental goals (competitive performance and user friendly environment) it would be naive to believe that it does not need further improvements. Some areas that the software can be improved are:

- Optimization of the Pareto extrapolation algorithm parameters
- Development of an algorithm that utilizes surrogate models to improve the convergence of standard evolutionary optimization algorithms
- Utilize parallel computing models such as OpenMP to parallelize the developed optimization software
- Implement evolutionary algorithms for solving single objective optimization problems.
- Development of an auto-tuning algorithm that tunes both PEA NSGA-II and standard NSGA-II algorithm. This algorithm would allow the software to be used at the highest level of performance, without any optimization theory knowledge from the user.

Bibliography

Auger A., J.Bader, D.Brockho,E.Zitzler(2009). Theory of the hypervolume indicator:μ-distributions and the choice of the reference point. Proceedings of the tenth ACM SIGEVO workshop on foundations of genetic algorithms, 87-102

Cahon, S., Melab, N. and Talbi, E. (2004). ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, 10(3), pp.357-380.

Coello Coello, C., Lamont, G. and Van Veldhuisen, D. (2007). *Evolutionary algorithms for solving multi-objective problems*. New York: Springer.

Deb, K., Pratap, A., Agarwal, S. and Meyarivan, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, 6(2), pp.182-197.

Fonseca, L., Barbosa, H. and Lemonge, A. (2009). A similarity-based surrogate model for enhanced performance in genetic algorithms. *OPSEARCH*, 46(1), pp.89-107.

Genetic algorithms in search, optimization, and machine learning. (1989). *Choice Reviews Online*, 27(02), pp.27-0936-27-0936.

Goel, T., Vaidyanathan, R., Haftka, R., Shyy, W., Queipo, N. and Tucker, K. (2007). Response surface approximation of Pareto optimal front in multi-objective optimization. *Computer Methods in Applied Mechanics and Engineering*, 196(4-6), pp.879-893.

Gonzalez, R. (2008). *Neural Networks for Variational Problems in Engineering*. PhD. Technical University of Catanolia.

Herrera, F., Lozano, M. and Moraga, C. (1999). Hierarchical distributed genetic algorithms. *International Journal of Intelligent Systems*, 14(11), pp.1099-1121.

Horn, J., Nafpliotis, N. and Goldeberg, D. (1994). A Niched Pareto Genetic Algorithm for Multiobjective Optimization. In: *First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*. New Jersey: IEEE Service Center.

Jin, Y. (2003). A comprehensive survey of fitness approximation in evolutionary computation. *Soft Computing*, 9(1), pp.3-12.

Karakasis, M. and Giannakoglou, K. (2006). On the use of metamodel-assisted, multi-objective evolutionary algorithms. *Engineering Optimization*, 38(8), pp.941-957.

Mitchell, M. (1996). *An introduction to genetic algorithms*. Cambridge, Mass.: MIT Press.

Nissen, S. (2003). *Implementation of a Fast Artificial Neural Network Library (fann)*. Graduate report. University of Copenhagen Department of Computer Science.

Ocagne, M. (1885). *Coordonnées parallèles & axiales*. Paris: Gauthier-Villars.

Oxforddictionaries.com. (2016). *optimization - definition of optimization in English from the Oxford dictionary*. [online] Available at: http://www.oxforddictionaries.com/definition/english/optimization [Accessed 11 Jul. 2016].

Papadopoulos, C., Efstathiou, E., Nikolakopoulos, P. and Kaiktsis, L. (2011). Geometry Optimization of Textured Three-Dimensional Micro- Thrust Bearings. *Journal of Tribology*, 133(4), p.041702.

Poloni, C., Giurgevich, A., Onesti, L. and Pediroda, V. (2000). Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics. *Computer Methods in Applied Mechanics and Engineering*, 186(2-4), pp.403-420.

Practical mathematical optimization: an introduction to basic optimization theory and classical and new gradient-based algorithms. (2007). *Choice Reviews Online*, 44(05), pp.44-2754-44-2754.4

Riedmiller, M. and Heinrich, B. (1992). Rprop - A Fast Adaptive Learning Algorithm. *Proceedings of the International Symposium on Computer and Information Science*, VII.

Schaffer J. David.(1984) *Multiple Objective Optimization with Vector Evaluated Genetic Algorithms*. PhD thesis, Vanderbilt University.

Sefrioui M. and J. Periaux. Nash Genetic Algorithms: examples and applications. In 2000 Congress on Evolutionary Computation, volume 1, pages 509–516, San Diego, California, July 2000. IEEE Service Center.

Srinivas, N. and Deb, K. (1994). Muiltiobjective Optimization Using Nondominated Sorting in Genetic Algorithms. *Evolutionary Computation*, 2(3), pp.221-248.

Tenne, Y., Tenne, Y. and Goh, C. (2010). *Computational intelligence in expensive optimization problems*. Berlin: Springer.

Thie, P. and Keough, G. (2008). *An introduction to linear programming and game theory*. Hoboken, N.J.: Wiley.

Zitzler E., D. Brockhoff, L. Thiele (2007). The hypervolume indicator revisited: On the design of pareto–compliant indicators via weighted integration. Evolutionary Multi-Criterion Optimization, Vol. 44013, pp.862-876

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