JOHANNES KEPLER
UNIVERSITÄT LINZ | JKU

TNF

Technisch-Naturwissenschaftliche
Fakultät

Enhanced Evolutionary Algorithms for Solving Computationally-Intensive Multi-Objective Optimization Problems

DISSERTATION

zur Erlangung des akademischen Grades

Doktor

im Doktoratsstudium der

Technischen Wissenschaften

Eingereicht von:
Alexandru-Ciprian Zăvoianu, MSc

Angefertigt am:
Institut für Wissenbasierte Matematische Systeme

Beurteilung:
Prof. Dr. Erich Peter Klement (Betreuung)
Prof. Dr. Dana Petcu

Linz, Januar 2015

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Dissertation selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Die vorliegende Dissertation ist mit dem elektronisch übermittelten Textdokument identisch.

Linz, Januar 2015 Alexandru-Ciprian Zăvoianu, MSc

To Diana

Acknowledgments

This thesis would not have been possible without the support and guidance of a number of people.

First and foremost I want to thank my adviser Prof. Erich Peter Klement for accepting me as his doctoral student and for his continuous support over the last few years. I am also very thankful to my second adviser, Prof. Dana Petcu, for her very valuable suggestions related to the present work and for generally encouraging me to grow and better myself, both as a student and as a young researcher. I would like to express my gratitude to Prof. Armin Biere for all his kind help and for agreeing to serve as head of the dissertation committee.

All the work that resulted in this thesis was performed in close cooperation with the Linz Center of Mechatronics (LCM) and the Institute of Electrical Drives and Power Electronics (EAL) of the Johannes Kepler University Linz. I would like to express my gratitude to Prof. Wolfgang Amrhein (the head of EAL) for giving me the opportunity to closely work with amazing professionals like Dr. Sigfried Silber, DI Günther Weidenholzer, Werner Koppelstätter and (especially) Dr. Gerd Bramerdorfer without whom many of the valuable electrical drive design optimizations and software implementations that accompany the presented research would not have been possible. Furthermore, many of the ideas and concepts that steered our research over the past 3 and half years are the direct results of very fruitful discussions with the EAL team.

My work in Austria for the past years would not have been as enjoyable as it was without my wonderful colleagues (and friends) at the Department of Knowledge-Based Mathematical Systems. Special thanks to Dr. Edwin Lughofer for all the help in preparing the publications that disseminate some of the key parts of this thesis, to Prof. Susanne Saminger-Platz for all her help and support (especially in the crucial few months before the submission of the thesis) and to Sabine Lumpi for helping me with countless administrative issues.

I would also like to thank Prof. Michael Affenzeller and Prof. Daniela Zaharie, for introducing me to the field of nature-inspired computing and for the invaluable help and useful suggestions they have given me over the years.

Last but by no means least, I have to thank my parents and my family for their unconditional support and constant encouragements. I also thank all my friends for encouraging me to finish this thesis. This work is dedicated to my love, Diana as a special thank you for her love, constant help and support and unbounded belief in me.

Abstract

Over the past decade, specialized evolutionary algorithms have emerged as one of the best methods for solving multi-objective optimization problems (MOOPs). These approaches have been used to tackle complicated real-life optimization scenarios from various fields. One of the characteristics of evolutionary algorithms is that a large number of evaluations of the objective function need to be performed during their execution. This inherent requirement is extremely problematic in the case of industrial optimization contexts where evaluating objective performance and constraint satisfaction is very computationally intensive (i.e., requiring several minutes or even hours).

We are particularly concerned with (and largely motivated by) the need to improve optimization performance on computationally-intensive multi-objective optimization problems (CIMOOPs) from the field of electrical drive design. Such design problems are fairly complicated as they arise from the need to simultaneously increase the efficiency, reduce the costs and improve the fault tolerance and operating characteristics of new electrical machines. Furthermore, (several) computationally-intensive finite element (FE) simulations are usually required in order to evaluate the performance of a single design.

In this thesis we present the results of research that was aimed at improving the performance of multi-objective evolutionary algorithms (MOEAs) when applied on CIMOOPs. Initial focus falls on designing and applying on-the-fly surrogate modeling in order to reduce the dependency of the MOEAs on FE simulations. By surrogate modeling we understand the creation of fast-to-evaluate linear and non-linear regression models that can accurately approximate FE results. Next, we investigate the best way of distributing MOEA computations over a high-throughput computing environment, when considering a master-slave architecture. Finally, we propose two new MOEAs (based on coevolution) that are both highly competitive when compared to state-of-the-art approaches and quite robust with regard to their own parameterization settings. We also present a newly synthesized performance assessment methodology that can generally aid when wishing to test or fine tune a particular MOEA over a large set of benchmark problems.

Several statistical analyses over benchmark MOOP results and empirical observations of average performance on electrical drive design CIMOOPs confirm that the enhanced evolutionary computation methods we propose are generally able (individually and especially combined) to improve both the speed and the accuracy of multi-objective optimizations.

Contents

List of Figures

List of Tables

List of Algorithms

List of Terms

ANN

Artificial Neural Network

CIMOOP

Computationally-Intensive Multi-Objective Optimization Problem

DE

Differential Evolution

DECMO

Differential Evolution-based Coevolutionary Multi-Objective Optimization algorithm

DECMO2

Improved version of DECMO

DEMO

Differential Evolution for Multiobjective Optimization algorithm

DM

Decision Maker

EA

Evolutionary Algorithm

ES

Evolutionary Strategy

FE simulation

Finite Element simulation

GA

Genetic Algorithm

GDE3

Generalized Differential Evolution 3 algorithm

GEN-MSPS

Generational Master-Slave Parallelization Scheme

GP

Genetic Programming

Ind_H

Hypervolume indicator

HRPC

Hypervolume-Ranked Performance Curves

$\mathbf{p^{Id}}$

Ideal objective vector

$\mathbf{x^{Id}}$

Ideal variable vector / individual

MSP

Master-Slave Parallelization

MCDM

Multi-Criteria Decision Making

MOEA

Multi-Objective Evolutionary Algorithm

MOO

Multi-Objective Optimization

MOOA

Multi-Objective Optimization Algorithm

MOOP

Multi-Objective Optimization Problem

MLP

Multi-Layer Perceptron

MOEA/D

Multiobjective Evolutionary Algorithm based on Decomposition

MOEA/D-DE

MOEA/D version that uses differential evolution strategies

$\mathbf{p}^{\mathbf{nad}}$

Nadir objective vector

NSGA-II

Nondominated Sorting Genetic Algorithm II

nfe

Number of fitness evaluations

PF

Pareto Front

PF_{true}

True Pareto Front

PM

Polynomial Mutation

PN

Pareto non-dominated Set

PS

Pareto-optimal Set

RBFN

Radial Basis Function Network

SBX

Simulated Binary Crossover

SSA-MSPS

Steady-State Asynchronous Master-Slave Parallelization Scheme

SPEA2

Strength Pareto Evolutionary Algorithm 2

SVR

Support Vector Regression

Chapter 1

Introduction

1.1 Background

In the most general sense, an **optimization** (problem) **is** concerned with **finding the best option** (with regard to given criteria) **from a set of available alternatives**. The generality of this plain language definition is intended to emphasize the fact that an optimization problem can be identified in a multitude of contexts and, at close inspection, in nearly all life-related activities.

As over the centuries human societies gradually developed and became more complex, so did the type of optimization problems they were confronted with. In order to find solutions to these problems, ever more ingenious and successful *optimization strategies* were developed. The first of these strategies were purely empirical (i.e., adapting "what seems to work" in one context to a new one), but they evolved in time into (complex) analytical frameworks that emerged from the works of ancient philosophers / mathematicians and are presently found in a large range of scientific (sub)fields from mathematics and computer science.

Whenever the goal of the optimization is to find a solution that must "best" satisfy not one but several (often conflicting) criteria, one is operating inside a *multi-objective optimization* (**MOO**) context. Because of the rapidly increasing complexity of human society, the occurrence (or better said identifiability) rate of *multi-objective optimization problems* (**MOOP**s) and the importance of being able to solve them has known an ever increasing trend in the last 70 years. In present times, people are confronted with MOOPs (sometimes on a daily basis) in various activity fields ranging from the design of mobile communication infrastructures to financial portfolio management and the development of new pharmaceuticals.

Because of the increased complexity associated with MOOPs, over time, a lot of strategies for solving them have been proposed. In the past 20 years, the *evolutionary algorithm* (**EA**) paradigm has emerged as one of the most successful methods for tackling complex MOOPs where the practitioners want to obtain global views of the (estimated) solution space. In general, EAs are non-deterministic strategies that solve search and optimization problems by applying concepts inspired from natural evolution. One of the characteristics of EAs in general, and *multi-objective evolutionary algorithms* (**MOEAs**)

in particular, is that a large number of evaluations of the objective function need to be performed during their execution. This inherent requirement is extremely problematic for a growing class of *computationally intensive multi-objective optimization problems* (**CIMOOP**)s for which the process of evaluating objective performance and constraint satisfaction is very time-intensive (i.e., requiring several minutes or even hours). Concretely, in the present thesis, we are particularly concerned with CIMOOPs originating from the field of electrical drive design. These optimal design problems are quite complicated as they usually arise from the need to simultaneously reduce the costs, increase the efficiency and improve various operating characteristic of new electrical machines. Furthermore, computationally-intensive *finite element simulations* (**FE simulations**) must be used in order to evaluate the performance of a given design.

1.2 Goal and Approach

The main goal of our presently reported research is to generally improve the convergence speed and the final solution quality of evolutionary algorithms applied for solving computationally-intensive multi-objective optimization problems.

In order to achieve this goal, we investigated and obtained very promising results (i.e., MOEA-intended / suitable enhancements) along **three distinct but complementary research lines**:

- Initial focus was directed towards the very intuitive idea of reducing the dependency of the MOEAs on the computationally-intensive objective function evaluation process by incorporating on-the-fly **surrogate modeling** techniques.
- Secondly, we tried to determine the best way of **distributing MOEA computations** over a high-throughput computing environment, when considering a (basic) **master-slave paradigm** and the particularities of the CIMOOP one wishes to solve.
- Thirdly, we experimented with **coevolutionary-based enhancements** in order to obtain good MOEAs that are very robust with regard to their parameterizations (and thus highly suitable as black-box solvers).

The choice of dividing our efforts in what many would consider non-overlapping directions was largely motivated by practical considerations: throughout our collaboration with the LCM we have encountered several electrical-drive design CIMOOPs that proved extremely challenging when considering a single type of MOEA-intended enhancement. Therefore, there was a strong desire to develop several methods that would enable us to tackle such problematic cases from different angles.

1.3 Outline of the Thesis

The rest of this manuscript is organized as follows:

- Chapter 2 starts by introducing the formal general description of a MOOP and presents key concepts related to the structure of a MOOP solution. Focus then shifts to the presentation of classical MOO methods – some of which are critical for understanding more advanced solvers. The last part of this chapter is dedicated to evolutionary MOO and to the presentation of some of its best-known exponents.
- Chapter 3 is divided in three main sections. The first one describes the artificial and industrial MOOPs we refer to throughout this work. The second section is dedicated to standard performance measures used for assessing solution quality in MOO contexts. The third section is dedicated to a newly proposed MOEA performance evaluation methodology.
- Chapter 4 describes a systematic strategy for efficiently hybridizing a standard MOEA with on-the-fly surrogate models. This is the largest chapter of the thesis as it also presents concepts from the field of machine learning / statistical learning algorithms that are required in order to describe the (proposed) surrogate-based enhancements.
- Chapter 5 is dedicated to a combined quantitative and qualitative analysis aimed at providing practitioners of MOEAs with a simple but effective method of deciding which master-slave parallelization option is better given the particularities of the concrete optimization process.
- Chapter 6 introduces two coevolutionary-based MOEAs that are empirically proven to successfully integrate (in a single optimization run) the individual strategies of different standard MOEAs.
- Chapter 7 presents a very brief but extremely important overview of how the concepts form the previous three chapters can be easily combined in order to obtain powerful hybrid MOEAs than can successfully tackle an extremely difficult electrical drive design CIMOOP.
- Chapter 8 contains general conclusions, a summary of the obtained results and an overview on future research lines.

1.4 Original Contribution

Apologizing in advance for referring to concepts not yet introduced, for the general interest of the reader, we now summarize the novel parts of our research, most of which have already been disseminated in scientific journals, conference proceedings and edited volumes:

- in Chapter 3, the racing-based methodology for easily assessing comparative MOEA performance over large problem sets has been introduced in [1];

- in Chapter 4, the systematic strategy for designing MLP-based global surrogate models and for successful integrating them in a hybrid MOEA-based solver has been initially proposed in [2] and described at length in [3];
- the ensemble-based approaches aimed at improving the overall time-wise efficiency of the surrogate construction process (also described in Chapter 4) have been introduced in [4];
- although seriously extended and improved, the main ideas pertinent to the comparison between the generational and steady-state master-slave parallelization schemata from Chapter 5 have been described in [5];
- the DECMO algorithm presented in Chapter 6 has been originally proposed in [6];
- the DECMO2 algorithm also described in Chapter 6 has been introduced in [1];
- the ideas and results from Chapter 7 and from Section 4.4.3 are completely new;

"A human being should be able to change a diaper, plan an invasion, butcher a hog, conn a ship, design a building, write a sonnet, balance accounts, build a wall, set a bone, comfort the dying, take orders, give orders, cooperate, act alone, solve equations, analyze a new problem, pitch manure, program a computer, cook a tasty meal, fight efficiently, die gallantly. Specialization is for insects."

Robert A. Heinlein

Chapter 2

Multi-Objective Optimization

2.1 Multi-Objective Optimization Problems

2.1.1 General Definition and Structure of Solutions

A general unconstrained MOOP can be formulated[1] as:

$$\begin{aligned} &\text{minimize} \quad O(\mathbf{x}) = [o_1(\mathbf{x}), \ldots, o_m(\mathbf{x})] \\ &\text{subject to} \quad \mathbf{x} \in D^n, \quad m \geq 2 \end{aligned} \tag{2.1}$$

where:

- all the m single-objective functions contained in O, $o_i : D^n \to \mathbb{R}$, $i \in \{1, \ldots, m\}$, need to be minimized simultaneously;
- the n-dimensional domain D^n is called the *variable* (or *decision*) *space* of the MOOP;
- the m-dimensional codomain of O is $Z^m \subseteq \mathbb{R}^m$ and is called the *objective* (or *result*) *space* of the MOOP.

Furthermore, in nearly all cases, the feasible decision space of the MOOP is itself multidimensional (i.e., $n > 1$) meaning that $\mathbf{x} \in D^n$ is a (variable) *vector* (*multidimensional point/n-tuple*) and it has the form $\mathbf{x} = (x_1, x_2, \ldots, x_n)$.

Throughout the remainder of this work we shall mark with n the dimension of the feasible decision space of the MOOP and with m the dimension of the objective space. Also we shall use a bold face and superscript notation for vectors (e.g., $\mathbf{x}^*, \mathbf{y}^\mathbf{a}$), subscripts for their components (e.g., x_1^*, x_n^*, y_2^a) and all sets shall be marked with capitals[2] (e.g., D).

Since the single objectives that need to be optimized [i.e., $o_1(\mathbf{x}) \ldots o_m(\mathbf{x})$ from (2.1)] are generally conflicting (e.g., cost vs. quality, risk vs. return on investment), usually, there

[1] Like any optimization problem, a general MOOP can be formulated either as a minimization or a maximization problem. In the present work we prefer the former approach.

[2] In order to maintain consistency with literature, in special cases we also use capitals for numeric parameters.

is no single vector in D able to simultaneously minimize all of them. If such an *ideal variable vector* (not: $\mathbf{x^{Id}}$) does exist, the MOOP is reduced to solving any of the contained single-objective functions and no special multi-objective solving strategies are needed.

When a single vector cannot minimize all the single-objective functions, it is quite intuitive that the solution of the MOOP comes in the form of a set of vectors (that would contain at least the solutions of all the contained single-objective functions). In order to formally define this solution set, we must first introduce the notions of **Pareto dominance** and **Pareto optimality**.

When considering two variable vectors $\mathbf{x}, \mathbf{y} \in D^n$, vector $\mathbf{x}$ is said to *Pareto-dominate* vector $\mathbf{y}$ (not: $\mathbf{x} \preceq \mathbf{y}$) if and only if $o_i(\mathbf{x}) \leq o_i(\mathbf{y})$ for every $i \in \{1,\ldots,m\}$ and $o_j(\mathbf{x}) < o_j(\mathbf{y})$ for at least one $j \in \{1,\ldots,m\}$. This means that $\mathbf{x}$ is better than $\mathbf{y}$ with regard to at least one objective and isn't worse than $\mathbf{y}$ with regard to any objective. A vector $\mathbf{x}^* \in D^n$ with the property that there exists no $\mathbf{y} \in D^n$ such that $\mathbf{y} \preceq \mathbf{x}^*$ is said to be *Pareto-optimal* with regard to (2.1).

The set that reunites all the Pareto-optimal vectors is called the *Pareto-optimal Set* (not: **PS**) and this set is the full /general solution of a MOOP. For many MOOPs, the *PS* is unknown and/or infinite. Therefore, in most application domains, decision makers use the *Pareto non-dominated Set* (not: **PN**) which contains an arbitrarily fixed number of vectors that are able to provide a good approximation of the *PS*. Thus, finding high-quality *PN*s is the goal of most modern methods aimed at solving MOOPs. For the rest of this work, for any given MOOP, we shall use the term *PN* in order to refer to a potential solution set and the term *PS* to refer to the *true solution set*.

The projection of an arbitrary *PN* on the objective space is called the *Pareto Front* (not: **PF**). Like many other authors, we shall refer to the projection of the *PS* on the objective space as the *true Pareto Front* (not: $\mathbf{PF_{true}}$) of the MOOP (please see Figure 2.1 for an example). The components of any Pareto-front are themselves m-dimensional vectors (points) and in order to distinguish these objective vectors from the n-dimensional variable vectors they are connected to, we shall mark Pareto front members exclusively with $\mathbf{p}$ and some associated superscripts. For example, $O(\mathbf{x^a}) = \mathbf{p^a}$, $O(\mathbf{y}^*) = \mathbf{p}^*$, etc. N.B. The Pareto dominance and Pareto optimality relations translate directly from variable vectors to their associated objective vectors and vice versa: $\mathbf{x^a} \preceq \mathbf{x^b} \iff \mathbf{p^a} \preceq \mathbf{p^b}$, given $O(\mathbf{x^a}) = \mathbf{p^a}, O(\mathbf{x^b}) = \mathbf{p^b}$.

When considering the PF_{true} of a problem and all its members, the imaginary point in objective space that would contain the worst (highest in our case) existing values for each of the m objective functions is called the *nadir objective vector* (not: $\mathbf{p^{nad}}$). As shown in Figure 2.1, this anti-optimal point can be considered the opposite of the objective space projection of the ideal variable vector (not: $\mathbf{p^{Id}} = O(\mathbf{x^{Id}})$). Both these extreme – and usually artificially constructed – points are used in the definition of several solving strategies and performance metrics.

It is very important to note that in most real-life applications (like the ones that motivate our research), solving a MOOP can be divided into two stages:

1. **Search** - the goal is to find variable vectors that are able to optimize (minimize) the contained single-objective functions.

2. **Decision making / Articulation of preferences** - the goal is to determine those

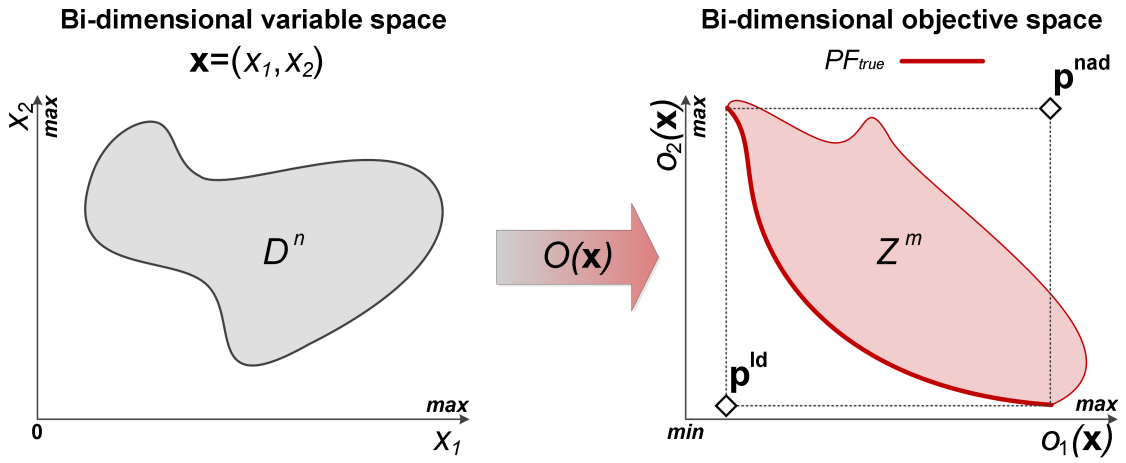


Figure 2.1: A diagram of the most important multi-objective optimization concepts outlined so far. N.B. The presented example is quite trivial as both variable and objective spaces are only bi-dimensional.

variable vectors that incorporate the best trade-offs in the *decision maker* (**DM**)'s opinion.

The focus of this thesis is on the analysis and design of methods (algorithms) that are generally able to efficiently solve the *search* stage by finding high-quality *PN*s. The motivation for this is that once a clear and broad picture of all the existing trade-offs in the MOOP is available (i.e., a good *PF* has been discovered), the DM can better tune the more subjective *multi-criteria decision making* (**MCDM**) part in order to select a very limited number of Pareto non-dominated solutions (sometimes just one) that will be constructed / implemented / applied in the real-life process modeled by the MOOP (s)he wishes to solve. This view regarding the interplay between the search and MCDM stages is by no mean general within the MOO community and Section 2.2.2 contains a more detailed discussion on the subject.

2.1.2 Constraints

When considering the general MOOP definition from (2.1), there are at least three types of constraints that can be enforced:

- result-related constraints;
- variable-related constraints;
- informal constraints.

For most MOOPs, constraints must be imposed in the result space of (2.1) in order to make the problems meaningful: e.g., we are only interested in the cost of the product if the associated quality is above an agreed threshold. Usually, these result-related constraints

pertain to the given individual objectives of the MOOP (i.e., they are internal constraints). Without loss of generality, such an internal constraint can be formulated as:

$$\begin{aligned} &c_{int} : D^n \ \rightarrow \mathbb{R} \\ &c_{int}(\mathbf{x}) \leq 0, \quad \mathbf{x} \in D^n, \end{aligned} \tag{2.2}$$

with the assumption that c_{int} is a composition of at least one of the single-objective functions from (2.1).

In many real-life MOOPs, constraints are also defined around independently specified auxiliary functions (results) that we wish to monitor but not include in the optimization: e.g., the carbon footprint of constructing the product should be smaller than an agreed threshold, although the carbon footprint is not an explicit goal of the optimization. The formulation of such an (external) constraint is identical to that of internal constraints, minus the composition requirement:

$$\begin{aligned} &c_{ext} : D^n \ \rightarrow \mathbb{R} \\ &c_{ext}(\mathbf{x}) \leq 0, \quad \mathbf{x} \in D^n, \end{aligned} \tag{2.3}$$

Therefore, most authors do not differentiate between the two types of result-related constraints.

Obviously, when result-related constraints are provided, a given variable vector $\mathbf{x^c} \in D^n$ will be included in the PN if and only if it satisfies all the constraints. Otherwise, the vector will be labeled as an **invalid vector**.

As the name suggests, the variable-related constraints that can be imposed on (2.1) concern the way the feasible decision space of the MOOP (i.e., D^n) is defined. For example, Miettinen [7] refers to D^n as the *"feasible region"* that is a subset obtained by imposing certain constraints on the *"decision variable space"* of the MOOP that is always $\mathbb{R}^n$. This point of view is in accordance with most real-life and artificial MOOP definitions where the domain D^n of the multi-objective function is defined as the Cartesian product of sets that contain the possible values of the individual decision variables. Formally, for a MOOP that has n decision variables, the (variable) vector encoding will be of the form:

$$\begin{aligned} &\mathbf{x} = (x_1, x_2, \ldots, x_n) \quad \text{with} \\ &x_1 \in D_1, \quad D_1 \subseteq \mathbb{R} \\ &\vdots \\ &x_n \in D_n, \quad D_n \subseteq \mathbb{R} \quad \text{and} \\ &D^n = D_1 \times \ldots \times D_n, \quad D^n \subseteq \mathbb{R}^n. \end{aligned} \tag{2.4}$$

It is important to note that each individual subset $D_1 \ldots D_n$ can have its own specific structure: real interval, infinite / finite integer set, union of real intervals, etc.

Whenever constraints are applied either in variable or result space, usually, the complexity level of the MOOP is also increased. Figure 2.2 provides a small example into this matter and shows how imposing variable (i.e., feasibility) and result-related constraints can seriously alter the solution space of a MOOP.

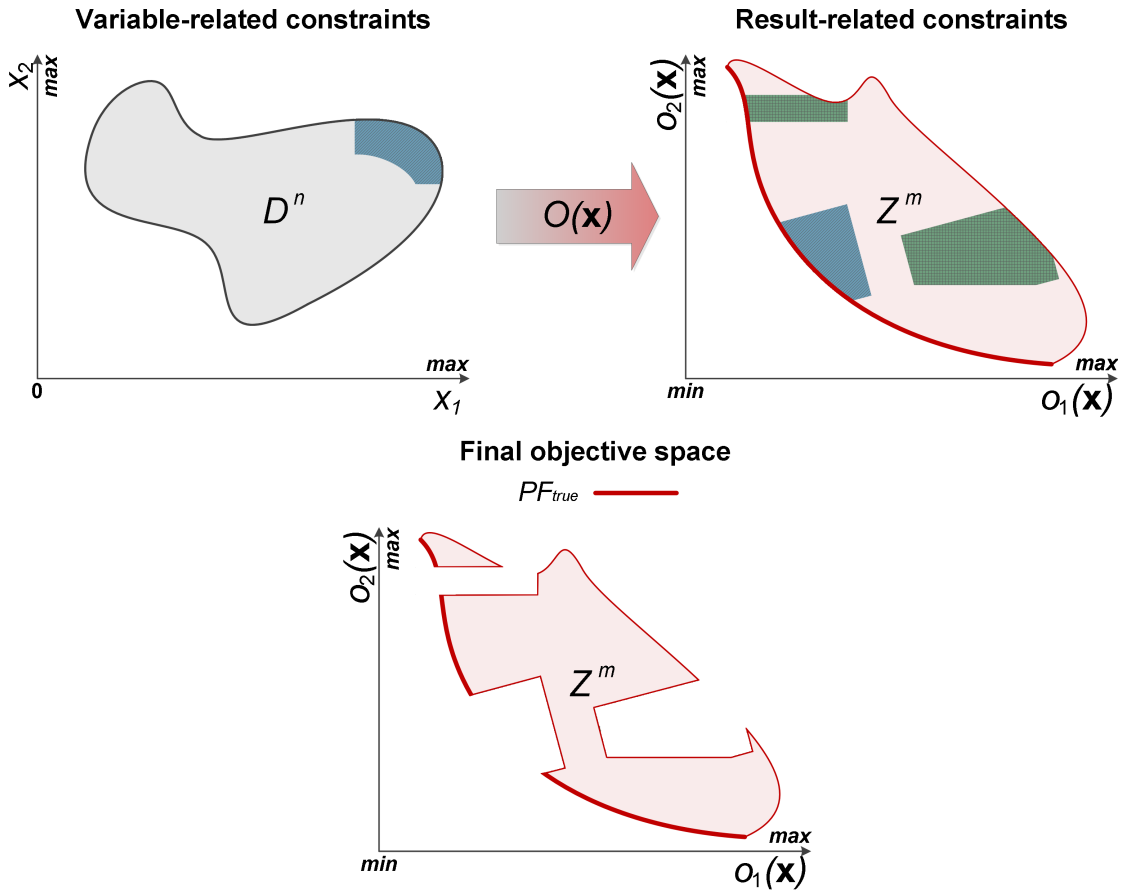


Figure 2.2: The possible effects on Z^m (the objective space of the MOOP) and on PF_{true} when imposing variable and result-related constraints.

We strongly feel that apart from the previously mentioned result and variable-related constraints that are generally mentioned in literature, more informal (domain-specific) limitations can also play a very important role. Two very important types of informal constraints that appear especially in real-life MOOPs are:

- time constraints – allowing a low time limit for solving the MOOP can be extremely challenging, especially when dealing with problems where evaluating one or more of the single-objective functions is computationally intensive;
- formalization constraints – not being able to correctly define D^n (i.e., the feasible decision space of the MOOP) can hinder the ability of most solvers to efficiently tackle the problem. Such cases are common when practitioners (are forced to) define a "search space" (not: $D' \supseteq D^n$) using the very convenient Cartesian product procedure presented in (2.4) without being able to account for the fact that for at least one vector from D' the multi-objective optimization function cannot be defined – i.e., $\exists \mathbf{x}^{\mathbf{err}} \in D' : O(\mathbf{x}^{\mathbf{err}})$ is undefined[3]. In these cases, where at least one such **error**

[3]In essence, $\mathbf{x}^{\mathbf{err}}$ is an infeasible point in the "search space" of the MOOP.

vector exists, it is obvious that $D^n \subset D'$ and , usually, the actual feasible variable space of the MOOP (i.e., D^n) is either not easy to define or even unidentifiable.

It is important to note that while most artificial MOOPs reported in literature, are defined as combinations of the general problem from (2.1) and several internal constraints (2.2) with a clearly defined feasible decision space [as shown in (2.4)] , the real-life electrical drive design optimization problems we aim to solve also involve external constraints (2.3) and, more importantly, time and formalization-related constraints.

2.2 Classical Multi-Objective Optimization Methods

2.2.1 General Considerations

It is widely accepted that the concept of MOO has emerged in the specialized contexts of economic equilibria and welfare theories at the end of the 19^{th} / beginning of the 20^{th} century with the pioneering works of Vilfredo Pareto [8] and Francis Edgeworth [9]. In [10] Stadler states that since MOO plays a central part in economic equilibrium theories, it can even be traced back to 1776 and Adam Simith's magnum opus *"An Inquiry Into the Nature and Causes of the Wealth of Nations"* [11].

The 1951 paper of Khun and Tucker [12] is credited with establishing multi-objective optimization as a mathematical discipline in its own right by introducing the *"vector maximum problem"*. In [13], the authors describe how in the '50s and '60s, a multitude of scientists have started to observe and address various MOOPs arising in production theory [14], public investment businesses [15] and engineering [16]. By the end of the '70s, the usage of MOO methods became quite popular in several fields [17] [18].

For a good overview regarding both early and more modern mathematical methods for solving MOOPs one should consult sources like [7], [19], and [13]. In the next sections, we shall try to classify and briefly describe some of the most well-known and widely used methods for solving MOOPs.

2.2.2 A Taxonomy of Available Approaches

Several ways of categorizing the numerous MOO techniques have been proposed over the years. The one that seems to have gained most ground in the community is attributed to Cohon and Marks [20] and has been used (with slight adaptations) by several researchers, such as [21], [22], [23], and [13]. Proclivity for this taxonomy can be explained by the fact that it is centered on the interaction between the two inherent stages of all MOOPs that are described at the end of Section 2.1.1: (I) the search stage and (II) the MCDM (or preference articulation) stage. Thus, according to Cohon and Mark's classification, we can distinguish between MOO methods that require:

No articulation of preferences - the DM does not state any preferences with regard to trade-offs between the single-objective functions of the MOOP;

A priori articulation of preferences - the DM states the trade-off preferences before the start of the search process;

A progressive articulation of preferences - the DM adapts trade-off preferences during the search process

A posteriori articulation of preferences - the DM considers trade-off preferences after the end of the search process;

As a general remark, regardless of the used articulation of preferences, most of the classical MOO methods we shall now proceed to describe are based on the central idea of *reducing / restating the original MOOP to / as a single-objective function*. Considering the general definition of a MOOP from (2.1), we mark its single-objective restatement by:

$$O_{sng}(\mathbf{x}) : D^n \to \mathbb{R} \tag{2.5}$$

Obviously, the main advantage of restating the problem is that the **reduced function** can be **minimized** using standard single-objective optimization methods. It must be emphasized, though, that the constraints of the original MOOP must also be taken into consideration during the optimization of (2.5) .

2.2.3 MOO with No Articulation of Preferences

The **Tschebyscheff min-max** and **global-criterion** methods are the most well known exponents of MOO techniques that require no articulation of preferences on part of the DM. When employing such methods, at the end of the optimization, the DM is presented with a single solution that (s)he may either accept or reject. Depending on the success of the run, this single solution may or may not be part of a good *PN* approximation of the PF_{true} associated with the original MOOP.

Considering the notations from (2.1), in its most basic form, the reduced function defined by the Tschebyscheff min-max approach is:

$$O_{sng}(\mathbf{x}) = \max_{i=1}^{m} \left[o_i(\mathbf{x}) - p_i^{Id} \right]. \tag{2.6}$$

The solution that minimizes (2.6) is the variable vector $\mathbf{x}^*$ with the property that $O(\mathbf{x}^*)$ is the point in objective space "closest" to a usually unknown $\mathbf{p^{Id}}$ that must be roughly estimated beforehand. Although direct applications of (2.6) exist (e.g., [24]]), since the function is unstable when the single-objectives of the MOOP have different magnitudes, most min-max formulations rely on relative deviations:

$$O_{sng}(\mathbf{x}) = \max_{i=1}^{m} \left[\frac{|o_i(\mathbf{x}) - p_i^{Id}|}{|p_i^{Id}|} \right]. \tag{2.7}$$

Obviously, when using (2.7) and other methods employing similarly-defined relative deviations, one must take special care in (re)formulating the MOOP such that $p_i^{Id} \neq 0$ for all $i \in \{1, \ldots, m\}$.

In the case of the global criterion approach, the reduced function is also aimed at measuring how close the objective vector is to $\mathbf{p^{Id}}$. Depending on how this "closeness" is defined, there can be different formulations, but the most common one is of the form:

$$O_{sng}(\mathbf{x}) = \left\{ \sum_{i=1}^{m} \left[\frac{|o_i(\mathbf{x}) - p_i^{Id}|}{|p_i^{Id}|} \right]^t \right\}^{1/t}. \tag{2.8}$$

In (2.8) we have preferred relative deviations. In literature (e.g., [7] and [23]), absolute ones (i.e., $|o_i(\mathbf{x}) - p_i^{Id}|^t$) are usually reported as standard. In [7], Miettinen also mentions that for $1 \leq t < \infty$, the exponent $1/t$ can be dropped. The usual choices for t are 1,2 or ∞. Results are obviously influenced by this choice as when $t = 1$ all deviations are taken into account in direct proportion to their magnitudes and for $2 \leq t < \infty$ larger deviations carry greater weight. If $t = \infty$, only the largest deviation is taken into consideration and the resulting L_∞-metric is equivalent to the Tschebyscheff min-max formulation from (2.7).

There are also several other methods that do not require an articulation of preferences like the **Nash arbitration**, **Rao's method**, and the **MPB method**. A comprehensive review of these can be found in [7] and [23].

2.2.4 MOO with A Priori Articulation of Preferences

In [22], Andersson states that a priori articulation of the DM's preferences is the most common way of conducting MOO. This should explain the numerous approaches reported in literature that fall in this category. In order to allow the DM to express his preferences / opinions / hopes on what the outcome of the optimization should be, most of these a priori approaches rely on $O_{sng}(\mathbf{x})$ formulations that require *parameters* like weights, constraints, exponents, etc. Nevertheless, Miettinen argues that a priori approaches involve inherent difficulty in the sense that the DM does not necessarily *"know beforehand what is possible to attain in the problem and how realistic her or his expectations are"* [7]. Hence, bad parameter settings (i.e., unrealistic expectations) can seriously influence the outcome of the optimization.

As mentioned, several a priori MOO approaches rely on assigning weights to the different single-objectives contained by the MOOP. Among them, the **weighted sum** method is the simplest and most common:

$$O_{sng}(\mathbf{x}) = \sum_{i=1}^{m} w_i o_i(\mathbf{x}). \tag{2.9}$$

Unsurprisingly, numerous weighting strategies for (2.9) have been reported over the years (please see [25] for a review). The simplest and most common of them include ranking methods, where the different objectives are weighted proportionally to their perceived importance. Another option are categorization methods, where objectives are assigned weights corresponding to a broad category (e.g., "highly important", "moderately important") that they have previously been assigned to by the DM. More advanced weighting schemata are based on pair-wise comparisons between objectives and eigenvalues [26].

By introducing weights, both methods presented in the previous subsection can also be parameterized such as to articulate the DM's preferences. As such, by modifying (2.6) we obtain the **weighted Tschebyscheff** (or weighted min-max) method:

$$O_{sng}(\mathbf{x}) = \max_{i=1}^{m} \left\{ w_i \left[o_i(\mathbf{x}) - p_i^{Id} \right] \right\} \tag{2.10}$$

and by modifying (2.8) we attain the **weighted global criterion** formulation:

$$O_{sng}(\mathbf{x}) = \left\{ \sum_{i=1}^{m} w_i^p \left[\frac{|o_i(\mathbf{x}) - p_i^{Id}|}{|p_i^{id}|} \right]^t \right\}^{1/t}. \tag{2.11}$$

Another popular approach for a priori articulating preferences is **lexicographic ordering**. The first step requires the DM to order the m single objectives of the MOOP in decreasing order of their absolute importance: $o_{L1}(\mathbf{x}), \ldots, o_{Lm}(\mathbf{x})$. After this reordering the first (i.e., most important) single-objective function is minimized subject to the original constraints [see (2.2) and (2.3)] of the MOOP. We reunite all the solutions of this problem into the set LS^1 and then proceed to minimize the second most important objective: $o_{L2}(\mathbf{x})$. This time the optimization is subject to both the original MOOP constraints as well as to the extra constraint that if $\mathbf{x}^{*\mathbf{L2}} \in D$ is to be accepted as a solution of $o_{L2}(\mathbf{x})$ then $o_{L1}(\mathbf{x}^{*\mathbf{L2}}) \leq o_{L1}(\mathbf{x}^{*\mathbf{1}}), \forall \mathbf{x}^{*\mathbf{1}} \in LS^1$. In other words, as the optimization progresses, at the i^{th} iteration ($1 < i \leq m$), we wish to preserve the best found values of the previous, more important, $i-1$ objectives. Considering that at step i we aim to minimize $O_{sng}(\mathbf{x}) = o_{Li}(\mathbf{x})$ and that LS^i contains the solutions of this minimization, the iterative optimization process stops either after the m^{th} iteration or when optimizing $o_{Li}(\mathbf{x})$ does not yield a solution. In case of the former, LS^m will hopefully contain some of the solutions of the original MOOP and, in case of the latter, these best-so-far solutions will be aggregated in LS^{i-1}.

Among lexicographic methods, those approaches where authors recommend applying more relaxed constraints (so called **hierarchical methods**) deserve special mention [27] [28]. Early MOO methods based on lexicographic orderings are examined in depth in [29].

In [7], Miettinen states that the **goal programming** method introduced by Charnes et al. in [30] and [31] is one of the first techniques *"expressly created"* for MOO. In it's simplest form it requires the DM to specify (optimistic) attainment levels (i.e., goals) for each of the simple-objective functions that are part of the MOOP. All these values are aggregated into a vector of goals (not: $\mathbf{g}$) such that g_i is the optimization goal of $o_i(\mathbf{x})$ from (2.1). The idea is to minimize the total deviation from these goals, i.e. to optimize $\sum_{i=1}^{m} |\delta_i|$, where δ_i is the deviation of $o_i(\mathbf{x})$ from g_i. Taking into account that goal programming traces its roots in the field of linear programming, a quite convenient way of handling the absolute values is to split them into positive and negative deviations. As such, given that $\delta_i = g_i - o_i(\mathbf{x})$, δ_i can be either positive or negative depending on how the goals have been defined. Regardless of sign, we can model them as $\delta_i = \delta_i^- - \delta_i^+$, with $\delta_i^- \geq 0$, $\delta_i^+ \geq 0$ and $\delta_i^- \delta_i^+ = 0$ for all $i \in \{1, \ldots, m\}$. We mark with δ_i^- a negative deviation or *underachievement* [i.e., the case when $g_i \geq o_i(\mathbf{x})$] and with δ_i^+ a positive deviation or overachievement [i.e., the case when $g_i \leq o_i(\mathbf{x})$]. The fact that $\delta_i^- \delta_i^+ = 0$ generally holds implicitly (as there can not be non-negative underachievements and overachievements at the same time) and this restriction

is usually not included in the formulation of the method in order to avoid nonlinearity. Summarizing, the goal programming method redefines the original MOOP as:

$$\begin{aligned} & O_{sng}(\mathbf{x}) = \sum_{i=1}^{m} |\delta_i^- - \delta_i^+| \\ \text{subject to } & o_i(\mathbf{x}) + \delta_i^- - \delta_i^+ = g_i \\ & \delta_i^-, \delta_i^+ \geq 0 \text{ for all } i \in \{1, \ldots, m\}. \end{aligned} \tag{2.12}$$

The basic formulation from (2.12) can be improved such as to allow the DM to express preferences for certain (types of) deviations through weighting: w_i^- and w_i^+. More flexibility can also be provided by enabling the DM to specify preference (not: p_i) for certain goals using priority factors. These priority factors are conceptually different than regular weights as their role is to construct a *"combined approach"* [7] that also features a "lexicographic" ("hierarchical") order among the objectives. This, more general, goal programming method can be formulated as:

$$\begin{aligned} & O_{sng}(\mathbf{x}) = \sum_{i=1}^{m} p_i |w_i^- \delta_i^- - w_i^+ \delta_i^+| \\ \text{subject to } & o_i(\mathbf{x}) + \delta_i^- - \delta_i^+ = g_i \\ & \delta_i^-, \delta_i^+, w_i^-, w_i^+ \geq 0 \text{ for all } i \in \{1, \ldots, m\}. \end{aligned} \tag{2.13}$$

Preferences can also be formulated using products instead of objective-wise sums. For example, in their 2004 review [23], Marler and Arora mention that a **weighted product** approach could be used in order to ensure that, in the absence of any transformation functions, objectives with different orders of magnitude have similar importance:

$$O_{sng}(\mathbf{x}) = \prod_{i=1}^{m} [o_i(\mathbf{x})]^{w_i}. \tag{2.14}$$

The weights w_i from (2.14) are used to assign the relative significance of the objectives. However, Marler and Arora also mention that this approach is not extensively used and that this might be the result of potential nonlinearities and consequent computational difficulties.

Far more popular product formulations for $O_{sng}(\mathbf{x})$ can be found in approaches related to **multi-atribute utility theory**. In these, the key to solving the MOOP is to construct a good utility function that is able to correctly capture the DM's preference structure. It is widely mentioned (e.g., [13] [22]) that designing good utility functions generally requires great effort on behalf of the DM. However, there are also utility-inspired approaches that are rather simple. One of the them is based on *acceptability functions* and was proposed by Wallace et al. in [32]. Using the notations from Section 2.1, this method requires the DM to assign an acceptability function to every single-objective of the original MOOP (not: $a_i : \mathbb{R} \to [0,1]$, $i \in \{1, \ldots, m\}$). For a fixed $u_i \in R$ such that $u_i = o_i(\mathbf{x}^*)$, $a_i(u_i)$ will equal 1 if the DM considers that u_i is completely acceptable (i.e., close to/equal to the true optimum of o_i) and, complementary, $a_i(u_i)$ will equal to 0 if the DM considers that u_i is completely unacceptable. For values of u_i that are between completely unacceptable and completely acceptable, $a_i(u_i)$ will take values inside $(0,1)$. Under the assumption of a deterministic

assignment of performance[4], $a_i[o_i(\mathbf{x}^*)]$ can thus be interpreted as the probability of accepting $\mathbf{x}^*$ as the solution of the original MOOP based solely on the performance achieved by this parameter vector with regard to the i^{th} objective. The global probability of accepting $\mathbf{x}^*$ is equal to $\prod_{i=1}^{m} a_i(\mathbf{x}^*)$. As such, $O_{sng}(\mathbf{x})$ can be defined as:

$$O_{sng}(\mathbf{x}) = -\prod_{i=1}^{m} a_i[o_i(\mathbf{x})]. \tag{2.15}$$

In their 1996 paper, Wallace at al. also give hints on how to design good acceptability functions that are able to *"drive design"*, i.e., ensure a good formulation of (2.15) by being sufficiently discriminant.

2.2.5 MOO with A Progressive Articulation of Preferences

MOO methods that allow for a progressive articulation of preferences are also known as interactive methods. They are motivated by the assumption that the DM is unable to correctly articulate (global) preferences before the start of optimization due to the complexity of the MOOP. At the same time, because of the same complexity, the DM has good reasons to believe that the more simple MOO approaches that require no articulation of preferences would not be able to deliver very good results.

Nevertheless, a good compromise can be achieved by allowing the DM to interact with the optimization process. The idea is that as the search advances, the DM learns more about the problem at hand and, based on this new insight, (s)he is able to steer the optimization process. Usually, this steering is based on (local) preferences that are developed/adapted based on the interplay between previous expectations and results achieved so far. Several sources (e.g., [7],[22], and [13]) argue that one of the major advantages of using interactive methods is that, because of the very active involvement in the resulting search-MCDM cycle, the DM is more likely to accept the final solution returned at the end of the MOO procedure.

In a different light, the requirement for a constant interaction between the DM and the optimization process can also be a major disadvantage, especially for MOOPs that are computationally intensive (see Section 3.1.2). This happens because the search-MCDM cycle is paused while waiting for the DM to express new preferences and this adds considerable strain either on the DM's work schedule, or on the already lengthy (expected) total duration of the optimization. Another disadvantage is that most interactive MOO approaches[5] also explicitly or implicitly demand a general consistency with regard to the iteratively specified preferences. As such, even though an adaptation of previously expressed preferences is expected, a total shift (that might be caused by changing the DM during a lengthy search process or by very important new insight) would normally require a restart of the optimization process.

Concretely, interactive MOO approaches are usually constructed around modifying an a prori technique in order to allow the DM to change the initial weights or to progressively reduce the search space by enforcing new objective-derived constraints.

[4] Evaluating $o_i(\mathbf{x})$ yields an exact result, not a value sampled from a predefined/known probability.

[5] Variations of the **reference point** method [33] form a notable exception.

The **interactive weighted Tschebycheff** approach proposed in [34] and refined and adapted in [7] works by progressively adapting the weights from (2.10). With regard to the used weighting vectors, viz. $\mathbf{w} = (w_1, \ldots, w_m)$, $\mathbf{w} \in \mathbb{R}^m$, this method requires that $0 < w_i < 1$ and $\sum_{i=1}^{m} w_i = 1$. At every step, (2.10) needs to be minimized p times, each time using a different weighting vector. After this, the p potential solutions are presented to the DM. (S)he can either decide to select one of them as the final solution of the MOOP or to focus (and proceed to the next step of) the search by automatically generating p more weighting vectors that are well-dispersed around some of the presented potential solutions. Methods for generating the initial set of p weighting vectors and for generating the subsequent DM-focused weighting vectors are discussed in [7]. Compared to all the other methods presented so far, this approach is much more computationally demanding as it requires performing $s \cdot p$ optimizations of $O_{sng}(\mathbf{x})$ [as defined by (2.10)] , where s is the total number of steps required by the DM.

The **STEM algorithm** (also known as STEP in some sources) is a classical example of progressive articulation of preferences via search space reduction. STEM also uses the weighted Tschebyscheff approach to find a potential solution. After this, some objectives (that are deemed as having been acceptably optimized) are progressively relaxed in order to allow for improvements to objectives that remain fixed (i.e., these are those objectives that are deemed as having been unacceptably optimized). The relaxation of a certain objective $o_i(\mathbf{x})$ translates into modifying the Tschebyscheff formulation by setting the weight corresponding to $o_i(\mathbf{x})$ to 0 (i.e., removing $o_i(\mathbf{x})$ as an objective) an by adding $o_i(\mathbf{x})$ as an additional hard constraint. The upper bounds of the new $o_i(\mathbf{x})$-derived constraint (i.e., the acceptable upper-bound of the relaxation) as well as the reason for which $o_i(\mathbf{x})$ was selected for relaxation are subject to the DM's preferences. In a successful STEM run, after at most $m-1$ relaxation stages, the method returns a solution that is fully acceptable for the DM. If this is not the case, STEM fails and the DM might consider restarting it after adjusting her or his expectations.

2.2.6 MOO with A Posteriori Articulation of Preferences

Methods that enable the a posteriori articulation of preferences are explicitly aimed at presenting the DM with a PF that is as close as possible to the PF_{true} of the MOOP. Hence, these approaches are the only ones that are truly aimed at solving the MOOP as a whole and not just at finding a few interesting solutions in (the vicinity) of PF_{true}. Obviously, the major advantage of these approaches is that, on the condition that the obtained solutions are *well-spread* and *sufficiently accurate*, the DM is presented with a global picture of the specific trade-offs between the objectives of the current problem.

An apparently simple way of obtaining a posteriori methods is to **run several independent optimization runs of a weighted a priori approach** and to use a different weighting vector for each run. The general restriction imposed on the weighting vectors is that all the weights must be non-negative. Most authors recommend imposing stronger (convex) restrictions on the weights. At the end of the process, by collecting all the obtained solutions, in theory, the DM should obtain an accurate discretization of the PN. At least two prerequisites are necessary for obtaining good results:

- the specific way in which the objectives are combined inside the weighted approach must be suitable for the MOOP at hand;

- the set of weighting vectors must be chosen such as to generate searches that are evenly spread across PF_{true};

With regard to the first prerequisite, it is well known (see [7] for details) that a linear combination of the objectives [like the one described in (2.9) with the extra restrictions: $\sum_{i=1}^{i\leq m} w_i = 1, w_i \geq 0$] cannot produce solutions on non-convex parts of the Pareto front. For MOOPs that have non-convex regions in the PF, the ε-**constraint** method proposed by Haimes et al. in [35] is more suitable. The idea of the ε-constraint approach is to minimize only one objective (the most important one) and to transform all the other objectives into constraints that are bound by individual thresholds:

$$\begin{aligned} O_{sng}(\mathbf{x}) &= o_i(\mathbf{x}),\ i \in \{1,...,m\} \\ \text{subject to } & o_j(\mathbf{x}) \leq \varepsilon_j \text{ for } j \in \{1,\ldots,m\} \text{ and } j \neq i \end{aligned} \tag{2.16}$$

By varying the threshold values [i.e., ε_j from (2.16)], one can potentially obtain multiple Pareto non-dominated solutions. Usually a preliminary analysis is required in order to discover proper values/ranges for the thresholds.

Even though better suited for MOOPs that have concave regions in their PF, like the basic weighted approaches, the ε-constraint method does not ensure finding PNs that are well spread even when systematically varying the weight or threshold vectors. In contrast, more advanced weighting-based strategies like the **normal boundary intersection (NBI)** method proposed by Das and Dennis in [36] and the **normal constraint (NC)** method introduced by Messac et al. in [37] can be used to generate PNs that are well spread on condition that the provided weights are systematically varied.

2.2.7 Remarks Regarding Classical MOO Approaches

It should be by now self-evident that the classification of MOO approaches from Section 2.2.2 can also be used to rank the methods according to their computational complexity. As such, while methods that require no articulation of preferences and a priori methods (excepting the lexicographic ordering) are reductions of the original MOOP to one single-objective optimization problem, the interactive articulation of preferences involves solving at least a few separate single-objective problems and the a posteriori methods essentially decompose the original MOOP in many single-objective optimization problems, each of which targeted at discovering a single solution in a specific section of the PF.

Nevertheless, the major disadvantage of all the MOO methods mentioned in the previous subsections and of a posteriori methods in particular is that, besides their internal strategy, the success of these methods in providing acceptable solutions is fully based on the ability to accurately solve the single-objective optimization problem(s) obtained from the original MOOP. In fact, some of the obtained single-objective problems have proven so complicated (multi-modal, false global optima, etc.) that practitioners of MOO and DMs were forced to

employ unorthodox and less mathematically-grounded optimization methods (e.g., evolutionary algorithms and other population-based metaheuristics) in order to solve them. The gap from using standard evolutionary algorithms to solve the individual single-objective problems to designing special evolutionary algorithms able to solve the original MOOP directly was quickly bridged and multi-objective evolutionary algorithm research began to flourish as these approaches rapidly became state-of-the-art (a posteriori) methods in the field of MOO. For more information on this topic, please see Section 2.3.2.

Because of technical limitations, a posteriori methods have been somewhat neglected in the initial years of MOO research. Two major disadvantages were generally associated with this type of approaches. Unsurprisingly, the first one regarded the large computational burden incurred by a posteriori methods. The second one was concerned with the fact that when using them, the DM might be overwhelmed with too many potential solutions to choose from.

With regard to the computational burden, although quite upsetting in several early a posteriori approaches, more modern strategies, like the previously mentioned evolutionary-based ones, and ever more powerful computers have managed to mitigate this setback for most types of MOOPs. Nevertheless, CIMOOP, which are the focus of the present research, remain a noteworthy counter-example and still impose very long optimization run times on a posteriori approaches.

In our opinion, obtaining a very large number of solutions in the *PN* of a given MOOP is a false problem nowadays as the alleged disadvantage can be easily offset by the wide range of MCDM strategies that can be used to *"navigate"* (order/filter) the solutions in the *PN*. In fact, in several real-life applications, like the electrical drive design problems we solve in our day-to-day activities, having a clear global picture of the trade-offs between the objective functions of the MOOP is of the utmost importance for both the DM (i.e., the electrical drive engineer) and the final client (i.e., the commercial/company partner). Thus, there is a very strong push to use methods that enable the a posteriori articulation of preferences even for the CIMOOPs that are common in mechatronics (please see Section 3.1.2 for details). All these aspects, in turn, largely motivate the present work that is aimed at presenting improvements of existing as well as new a posteriori approaches that can solve CIMOOPs more efficiently.

2.3 Multi-Objective Evolutionary Algorithms

2.3.1 Evolutionary Algorithms

Evolutionary algorithms (EAs) reunite a large family of nature-inspired problem solving techniques constructed around a theoretical framework that can be traced back to the pioneering works of the renowned English naturalist Charles Darwin who laid the foundation of the modern theory of evolution.

In his most influential work, the book *"On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life"* from 1859 [38], Darwin theorizes that evolution surely occurs in nature, that it is a gradual process that takes

place over thousands or millions of years and that the main driving force behind evolution is the process of *natural selection*, which, in turn, is based on two principles:

1. *Heredity* - in sexually reproducing species no two individuals are perfectly identical and this variation (i.e., specific traits) can be passed on to respective offspring.

2. *Survival of the fittest* - in populations with a relatively stable size over time, the individuals that display the "best" (most desirable) characteristics are more likely to survive, while individuals that exhibit undesirable characteristics are not so likely to survive.

In its simplest form, (i.e., a direct combination of the above two principles) the theory of natural selection postulates that genetically transferable "desirable" characteristics are very likely to survive as they are passed on to the offspring that will form the future generation. By contrast, "undesirable" characteristics are not so likely to survive since the individuals that display them do not have the chance to procreate sufficiently. As evolution progresses, generation after generation, desirable characteristics are very likely to become dominant within the entire population.

In natural environments, evolution and natural selection are tools of the utmost importance that enable a given species to survive (the primary goal/instinct of any living organism) by developing and retaining in the general population characteristics that enable the species to overcome various changes in the environment. As such, on a broad level, evolution can be seen as a natural search mechanism aimed at finding the genetic makeup (inheritable genotype) most likely to ensure the survival of the species.

The fact that **evolution** is a process that is proven to work (generally and eventually) even without knowing an exact "recipe" of how to reach its goal (which steps to take and in what order), makes it a **very powerful and universal search strategy**[6]. In fact, in his 1950 seminal paper on artificial intelligence titled *"Computing Machinery and Intelligence"* [40], Alan Turing speculates that a machine capable of passing the *"The Imitation Game"*[7] could be systematically designed and improved (i.e., taught) by means of an *"education process"* that displays *"an obvious connection"* to evolution. Furthermore, Turing even proceeds to explicitly suggest the following associations:

$$\begin{aligned} &\textit{"Structure of the child machine} = \textit{Hereditary material} \\ &\textit{Changes of the child machine} = \textit{Mutations} \\ &\textit{Natural selection} = \textit{Judgment of the experimenter"} \end{aligned}$$

The previously mentioned observations have led other pioneers like John Holland [41], Ingo Rechenberg [42], Hans-Paul Schwefel [43], John Koza [44] and many others to lay the ground for problem solving techniques like the *genetic algorithm* (**GA**), *evolutionary strategies* (**ES**) and *genetic programming* (**GP**) that are based on applying the principles

[6]For an interesting perspective on this matter from the field of evolutionary biology, please see *"The Blind Watchmaker"* by Richard Dawkins [39]

[7]Nowadays this procedure is known as the *"Turing test"* and constitutes one of the best known challenges from the field of artificial intelligence.

of evolutionary biology to computer science. In time, the term evolutionary algorithms (EAs) has become a commonly accepted general reference for evolutionary-inspired search / optimization methods and these methods have become quite popular in the past two decades as, in the words of John Holland [45], *"computer programs that 'evolve' in ways that resemble natural selection can solve complex problems even their creators do not fully understand"*.

Generally, an EA operates with a population of individuals that are actually representations of solution candidates for the optimization problem to be solved. By simulating various operations inspired from natural evolution (e.g., selection, recombination, mutation, survival), the algorithm creates new offspring individuals / solution candidates that (can) replace their parents inside the population set. Usually, the genetic operations are steered by considering the relative performance with regard to the given optimization goals (i.e. their fitness). As this process generates new individuals, at the same time, it explores the search space and, after several such iterations, the hope is that it finds individuals that encode good solutions of the problem to be solved.

More specifically, although characterized by a generally simple structure (as shown in Algorithm 1), EAs have proven extremely successful in tackling very complicated (real-life) non-linear optimization problems by **generally** being **able to discover acceptable solutions** in **reasonable time** [46] [47] [48][49]. The words "generally", "acceptable", and "reasonable" must be emphasized because, like natural evolution itself, an EA is a stochastic process that can not provide any guarantees with regard to global solution optimality, success ratio and time required for convergence. Nevertheless, in most cases, even after basic parameter tuning, the stochastic behavior of an EA can be controlled to a certain extent (range-bound).

It is important to note that the description of the EA process from Algorithm 1 is by no means a standard within the field. We propose it because it provides a truly general template that, when implemented, allows for a rapid specialization towards different techniques / paradigms (e.g., GAs, ESs, GP), different goals (e.g., single objective optimization, multi-objective optimization, tracking of population behavior) and different flavors of simulating evolution (e.g., generation-wise, steady-state, variable degrees of elitism).

There are six auxiliary methods used across Algorithm 1 that we shall now describe in order to provide a more in depth understanding of the search strategy proposed by EAs:

- **INITIALIZEPOPULATION**(*popSize*, *problem*) - this function returns a set containing a number of *popSize* **individuals** (**= solution candidates**) that have been created in a manner suitable for the *problem* at hand. The two things that must be considered are (I) choosing the correct **codification** (**= solution representation**) for the individuals and (II) making a choice between a random initialization and a strategy/heuristic-based one. In the case of GAs and ESs, the standard codification options are *fixed or variable-sized vectors* (arrays) consisting of binary, integer or real numbers. In the case of GPs, *syntax trees* are the preferred solution representation option. As a general rule, regardless of the problem one wishes to solve, the codification must be simple enough to allow for the design of useful genetic operators, but it must be comprehensive enough to also allow the storage of all the data required for evaluating

Algorithm 1 The possible structure of a general Evolutionary Algorithm

```
1: function EA(problem, popSize, genSize, stopCrit)
2:     P ← INITIALIZEPOPULATION(popSize, problem,)
3:     O ← Φ
4:     BestSolSet ← Φ
5:     EVALUATEFITNESS(P, problem)
6:     BestSolSet ←EXTRACTBESTSOLUTION(P, BestSolSet, problem)
7:     while stopCrit ≠ true do
8:         i ← 0
9:         while i ≤ genSize do
10:            S ← SELECTPARENTS(P)
11:            O' ← CREATEOFFSPRING(S)
12:            O ← O ∪ O'
13:            i ← i + |O'|
14:        end while
15:        EVALUATEFITNESS(O, problem)
16:        BestSolSet ←EXTRACTBESTSOLUTION(O, BestSolSet, problem)
17:        P ← SELECTFORSURVIVAL(P, O)
18:    end while
19:    return BestSolSet
20: end function
```

the quality of the individual. Furthermore, in order to accommodate simplicity and generality inside Algorithm1, we presume that the solution representation of our individuals also allows for the storage of any fitness-related information and any secondary meta-information required by the genetic operators (e.g., selection flags, generational tracking data, etc.).

- **EVALUATEFITNESS**(*Set*, *problem*) - this method computes the *problem*-specific **fitness function (= quality estimation)** for every individual from *Set* and stores the associated fitness value(s) within the corresponding individual. In fact, the only major requirement imposed on a given search / optimization problem in order to be solvable via EAs is that one should be able to suitably encode candidate solutions (as individuals) and to evaluate the quality of any individual that can be generated during the evolutionary process. The role of the fitness function is to perform the latter requirement by measuring how close a certain individual is to the true goal(s)/solution(s) of the problem. Since for many problems, the (exact) true solution is unknown, fitness functions are usually designed such as to quantify how "desirable" are the characteristics of a given individual by measuring: (I) how far it is from (an artificial) anti-ideal solution or (II) how good it is when comparing to previously generated individuals.

- **EXTRACTBESTSOLUTION**(*SolSet*, *CurentBestSolSet*, *problem*) - the purpose of this function is to find and return a set containing the elite individual(s) that is/are the

best at solving the current *problem*. The search for these elite individuals is conducted in the union set of *SolSet* (the set that generally contains newly evaluated individuals) and *CurrentBestSolSet* - the set that contains the best solution(s) found so far during the evolutionary run.

- **SELECTPARENTS**(*Set*) - this function selects and returns a limited number of individuals from the current population stored in *Set*. These individuals will be used as a base (i.e., parents) for creating new individuals. The actual number of parent individuals that will be returned depends on the specific type of EA. In the case of GP, two parents are usually required. In the case of ESs, one parent is enough. For GAs, generally two parents are required, but some paradigms can require up to six or more parents. There are numerous strategies for deciding how the parents should be selected. The most simple one is to select them without considering any fitness information (i.e., random selection). Another popular option is to select the parents with a probability that is directly proportional to their fitness (i.e., roulette selection). A combination of the previous two approaches (i.e., tournament selection) is also widely used. In the latter, several individuals are initially selected randomly and the fittest one is finally selected as a parent individual. For more details regarding various parent selection techniques and the reasons for choosing one over the other, please see the highly valuable introductory overviews by Goldberg [50] and Mitchell [51].

- **CREATEOFFSPRING**(*ParentSet*) - is the function responsible for creating new (offspring) solutions based on the (genetic) information encoded in the individuals from a given *ParentSet*. The actual creation process usually involves two (genetic) operations: (I) crossover/recombination and (II) mutation. These genetic operators can be applied separately or they can be chained: e.g., using crossover to obtain offspring and then applying mutation on these offspring is a very popular approach. Specific paradigms, like ESs, generally rely on a mutation-driven evolution (where the crossover operation is marginal or even not used), while most EA approaches recommend a careful balance between the **crossover operator** (**= the main genetic operator responsible for ensuring a good global search / diversification**) and the **mutation operator**. The latter is generally seen as a **secondary** but nevertheless **very important operator** that is responsible for both local search / intensification (on an individual level) as well as for diversification - introduction of new genetic material in the global genotype (on a population level). Usually the previously mentioned balance between the two operators is achieved by assigning rates for their usage. For example, a crossover rate of 0.9 means that the crossover operator is used with a 90% probability to obtain offspring. A mutation rate of 0.05 means that there is a 5% probability that an individual (usually a crossover obtained offspring) will undergo a mutation operation. Applying / designing crossover and mutation operators that are suitable for the problem at hand is one of the main requirements for a successful EA. Literature provides many examples of very successful genetic operators - some of them are highly specialized (problem specific) while others are quite generic.

- **SELECTFORSURVIVAL**(*ParentSet*, $Offspring Set$) - this function selects and re-

turns the set of individuals that will form the next generation of the EA. These individuals are selected from those of the current generation (i.e., the *ParentSet*) and from those that have been recently created using genetic operators (i.e., the $OffspringSet$). The selection strategy can range from something very simple like (I) accept all the offspring and disregard the parent individuals (i.e., generational replacement) or (II) rank all the individuals and select the best according to fitness (i.e., elitism) to (III) complex approaches designed to also maintain a good spread over the search space and genetic diversity. With regard to the size of the returned set of individuals, most EAs are based on the assumption that the size of each generation is constant (i.e. $= |ParentSet|$), but variants based on variable population sizes have also been proposed [52] [53].

The design of the offspring computation cycle (i.e., lines 9-14 from Algorithm 1) also deserves some extra clarifications. The primary idea behind it is to allow for two main paradigms:

1. *generational evolution* - each population of individuals is (largely) replaced in a single step by a new population containing their respective offspring. Using the notations from Algorithm 1, generational evolution happens when $genSize = popSize$.

2. *steady-state evolution* - only one new offspring is generated at a given time and, depending on the selection for survival criterion, it can be added to the current population – where it will replace an existing individual. Using the notations from Algorithm 1, steady-state evolution happens when $genSize = 1$.

Although minor at a first glance, the difference between the generational and steady-state evolutionary models can have a substantial impact on several classes of algorithms and problems (as we shall present in detail in Chapter 5).

The secondary reason for the presented offspring computation cycle is that it also allows for the implementation of the previously mentioned evolutionary strategies that rely on variable population sizes.

We end this short introduction into the field of evolutionary computation with the mention that for a specific flavor of EAs (i.e., fixed length binary-encoded generational GA with an infinite population, proportional selection, single-point crossover and bit flip mutation), in [41], Holland proved the now famous **Schema Theorem** (also known as the **Fundamental Theorem of Genetic Algorithms**) – a mile stone result (formal model) that explains the general effectiveness of the GA/EA search process. A schema can be defined as a binary-coded matching pattern that contains well-specified loci (i.e., positions in the binary string) and wildcard loci . The well-specified loci are marked by “1”s or “0”s and the wildcard loci are marked with “*”. Inside the population of the GA a given schema is said to *”represent"* all the individuals with whom it shares identical values of the well-specified loci. The fitness of a schema is computed as the average fitness of all the individuals in the population that it represents. The order associated with a schema is the total number of well-specified loci and the length of a schema is the difference between the positions of the

last and the first well specified loci. For example, under the assumption that the length of the binary-encoded individuals in the population is 10, some possible schemata might be:

$$\begin{aligned}
\mathbf{s_1} &= (1,0,*,*,1,*,*,*,*,1) \quad \text{with } order = 4 \quad \text{and } size = 10-1 = 9 \\
\mathbf{s_2} &= (*,*,1,*,0,1,1,*,*,0) \quad \text{with } order = 5 \quad \text{and } size = 10-3 = 7, \\
\mathbf{s_3} &= (*,*,*,0,0,0,*,*,*,*) \quad \text{with } order = 3 \quad \text{and } size = 6-4 = 2
\end{aligned}$$

Holland's theorem states that small, low-order schemata of above-average fitness increase exponentially in successive generations. In plain language, the Schema Theorem gives mathematical proof that the design of the (analyzed) evolutionary process ensures that "desirable characteristics" present at a certain moment in the population are highly likely to be transferred (and multiplied) in future generations.

2.3.2 Extensions to Multi-Objective Optimization Problems

The first applications of EAs in the field of multi-objective optimization were aimed to use the good global optimization performance displayed by these heuristic methods in order to solve MOOPs that were reduced in advance to single objective optimization problems using the objective-aggregation techniques presented in Section 2.2.7. One of the first examples of such usage can be found in Rosenberg's PhD thesis from 1967 [54].

David Schaffer is widely regarded as the first to have proposed an EA that incorporates some type of multi-objective optimization logic inside the evolutionary cycle. His approache is called the *Vector Evaluated Genetic Algorithm* (VEGA) [55] and is considered the very first multi-objective evolutionary algorithm (MOEA). The specific MOO logic is found inside the parent selection procedure (line 10 in Algorithm 1) that, at each generation, starts by creating m equal sub-populations – where m is the number of objectives to be optimized, as defined in (2.1). Each of these sub-populations is obtained by performing proportional selection (on the current population of the GA) according to a specific objective function [i.e., $o_1 \dots o_m$ from (2.1)]. In a final step, the niched sub-populations are shuffled together to obtain the new parent population that will produce offspring via crossover and mutation. These offspring, in turn, will form the population of the next generation. The main problem of VEGA is that its niching mechanism is too coarse and solutions that have an acceptable performance but are not the best w.r.t. any of the m objectives cannot survive (for long) under the proposed selection scheme. This means that very promising trade-off solutions (i.e. non-dominated solutions) tend to be disregarded and thus, in some cases, like MOOPs with concave PFs, the overall VEGA population is prone to *speciation*: division in sub-populations that are particularly strong in only one objective.

In [50] Goldberg suggests that genetic algorithms aimed at solving MOOPs should employ a mechanism based on *non-dominated ranking* inside the parent selection procedure. Goldberg proposes a ranking method that would assign the highest rank to the individuals that are not Pareto-dominated by any other individuals in the population. The second highest rank would be assigned to the individuals that become non-dominated when the previously ranked individuals would not be considered. The ranking would continue in a

similar fashion until the entire population is processed. In order to prevent the algorithm to converge to a single solution, Goldberg also suggests to combine the ranking with a *niching* technique, like his previously proposed fitness sharing [56] scheme. Although he does not provide an implementation for his method, Goldberg postulates that the resulting evolutionary paradigm would gradually move the population of the GA towards the *PF* and would be able to main solutions all along the non-dominated frontier.

The ideas put forward by David Goldberg quickly gained ground and, when considering all the MOEA developments from the last 25 years, one could also easily argue that they became canonical within the field. Some of the most successful early proposals that followed and validated Goldberg's recipe were (in decreasing order of perceived performance according to [57]):

- the *Multi-Objective Genetic Algorithm* (MOGA) [58];
- the *Niched-Pareto Genetic Algorithm* (NPGA) [59];
- the *Nondominated Sorting Genetic Algorithm* (NSGA) [60];

The last, but by no means least, early contribution to MOEA development is credited to Zitzler and Thiele with the introduction of *elitism* as a key part of the evolutionary process in their 1999 proposal of the *Strength Pareto Evolutionary Algorithm* (SPEA) [61]. More specifically, Zitzler and Thiele proposed to store the non-dominated individuals found during the entire evolutionary process inside an secondary / external population (or archive). The reasons for such an approach are quite intuitive as an individual that is non-dominated inside the current population of the EA is not necessarily non-dominated with respect to all the individuals that have been generated during the evolutionary cycle. Nevertheless, at the end of the run, it is highly desirable to provide the user solutions that are non-dominated with regard to any other solution found during the run. In the basic definition, the external archive can be seen as an updatable array where:

- a new solution is inserted into the archive only if it is not Pareto dominated by any individual currently in the archive;
- if a solution that is newly inserted in the archive Pareto dominates existing solutions, than the latter are removed form the archive.

After the publication of SPEA, many most researchers started to integrate elitism into their MOEAs and the importance of the concept became even more evident in 2000 when Rudolph and Agapie showed that elitism is a theoretical requirement for guaranteeing the convergence of a MOEA [62]. Apart from SPEA, the *Pareto Archived Evolutionary Strategy* (PAES) [63] is another well known early employer of the elite external archive mechanism.

Nevertheless, outfitting the MOEA with an external archive is not the only way to implement elitism. The other popular contender is the $(\mu + \lambda)$ selection (for survival) strategy . This general EA concept demands that, at every generation, all μ parent individuals of the current population must compete with all their λ offspring to form the population (of size μ) of the next generation. In the context of single-objective problems, this competition

can be settled by a simple fitness-based ranking. In the case of MOOPs, the winners are selected using a Goldberg-inspired schema based on a non-domination criterion (i.e. Pareto-based elitism) that is usually coupled with a distribution of solutions metric (i.e. a niching criteria). From a structural point of view, implementing elitism via the $(\mu + \lambda)$ strategy is far more straightforward than managing an external archive as it only requires the concrete implementation of the **SELECTFORSURVIVAL** function from Algorithm 1. Apart from this, implementing elitism via the selection for survival operation is, in many cases (some of which we shall present in detail in the next sections), computationally efficient.

With the hope that the previous paragraphs were able to at least capture the key stages of MOEA inception, we would like to kindly direct the reader interested in more details to an excellent historical overview of the field (till 2005) by Carlos A. Coello Coello [64].

Before proceeding with the detailed description of some of the most well-known and widely used MOEAs, we mention that the implementation of such an algorithm on the EA template from Algorithm 1 is usually quite simple as, apart from specialized strategy related aspects, it would only generally require:

- a basic (natural) codification of individuals either as binary arrays (in very specific cases) or, usually, as n-dimensional real-valued arrays [i.e., $\mathbf{x} = (x_1, x_2, \ldots, x_n), x_i \in D_i, i \in \{1, \ldots, n\}$] provided that the MOOP is defined as described in (2.1);
- no special actions during the generation of the initial population (i.e., the **INITIALIZEPOPULATION** method) as this can be done randomly and independently for each component of the real-valued arrays that encode the candidate solutions;
- a straightforward implementation of the **EVALUATEFITNESS** method that, when applied on a given individual $\mathbf{x}$, computes (and stores) the individual objective functions, viz. $o_1(\mathbf{x}), \ldots, o_m(\mathbf{x})$, associated with the MOOP to be solved;
- that at every generation $t > 0$, the **EXTRACTBESTSOLUTION** function returns the *PN* of the union between the current best solution [i.e., the *PN* extracted only from $P(t)$] and the newly generated offspring [i.e., $O(t)$].

2.3.3 NSGA-II and SPEA2

In 2002 the research group of Kalyanmoy Deb proposed the **Nondominated Sorting Genetic Algorithm II** (**NSGA-II**) [65] as an improved version of the original NSGA approach. The new method was much more computationally efficient and displayed such a good performance that it quickly became a landmark in the field and even present MOEAs (i.e., 2011 - 2014) are still being compared to it across various benchmark MOOPs.

Since NSGA-II is a reference in the field and has also inspired the development of several other well-known MOEAs, we shall now proceed to describe it in more detail using the general EA structure from Algorithm 1 as a basis. We mention that in Deb's original (generational) proposal, $|P| = |O|$ and therefore, when considering the notations from Algorithm 1, $genSize = popSize$. However, this equality restriction is by no means mandatory.

Preserving previous notations, at each generation t, NSGA-II operates with a parent population $P(t)$ and an offspring population $O(t)$. The selection for survival mechanism constructs population $P(t+1)$ by selecting the best $popSize = |P(t)|$ individuals from the combined population $C(t) = P(t) \cup O(t)$. In order to discriminate between different solutions, NSGA-II proposes a Goldberg-inspired fitness assessment mechanism that uses two metrics. The first one is based on the classification of the individuals from the combined population $C(t)$ into non-dominated fronts. The highest-level front $F_1(t)$ contains all the individuals that form the *PN* of $C(t)$. The next (lower-level) fronts $F_j(t), j > 1$ are obtained by iteratively removing higher level Pareto fronts from $C(t)$ and extracting the Pareto optimal set from the remaining individuals. For example, $F_j(t), j > 1$ contains the *PN* from $C(t) \setminus \bigcup_{k=1}^{j-1} F_k(t)$. Individuals from a higher-level front are assigned a higher fitness than those in a lower-level front and are preferred during the selection for survival procedure. In order to differentiate between individuals that are part of the same front, NSGA-II a secondary fitness metric, namely the *crowding distance*. For each individual still in contention, this secondary metric is used in order to estimate the local solution density by computing the average distance in objective space between the current solution and the two closest neighbors that surround it on each objective axis. Higher fitness values are assigned to individuals from less densely populated areas (i.e., the crowding distance is applied as a niching strategy). A graphic overview of the NSGA-II process through which population $P(t+1)$ is evolved from population $P(t)$ is presented in Figure 2.3.

In the year 2002, Zitzler et al. also proposed a much improved version of their 1999 MOEA. The most important novelty of the **Strength Pareto Evolutionary Algorithm 2** (**SPEA2**) [66] is also an elitist Goldberg-inspired selection for survival strategy - the *environmental selection* operator. Since we shall reference this particular strategy extensively in other parts of this work (predominantly in Chapter 6), we mark the environmental selection operation with $E_{sel}(Set, count)$, with the understanding that we refer to the procedure described in [66] through which one can select a subset of maximum *count* individuals from an original *Set*. In the context of Algorithm 1, at a given generation t, by applying the environmental selection strategy as $P(t+1) = E_{sel}(P(t) \cup O(t), |P(t)|)$, we obtain a direct implementation of line 17. We mention that, unlike in NSGA-II, the creators of SPEA2 generally recommend that $genSize < popSize$ which directly translates into $|O| < |P|$.

Environmental selection works by assigning a general rank to every individual $\mathbf{x} \in Set$ and then by selecting those individuals that display the lowest values. This general rank is a sum of two metrics, the raw rank $r(\mathbf{x})$ (2.17) – a non-domination measure – and the density $d(\mathbf{x})$ (2.18) – a solution density measure that acts as a niching strategy. The raw rank is computed by initially assigning a strength value $s(\mathbf{x})$ to every individual $\mathbf{x} \in Set$. This value quantifies the number of individuals from *Set* that are Pareto-dominated by $\mathbf{x}$ and is inspired by a similar mechanism proposed earlier in MOGA [58]. The raw rank assigned to $\mathbf{x}$ is obtained by summing up the strengths of all the individuals in the population that Pareto-dominate individual $\mathbf{x}$, i.e.,

$$r(\mathbf{x}) = \sum_{\mathbf{y} \in Set \,:\, \mathbf{y} \preceq \mathbf{x}} s(\mathbf{y}). \tag{2.17}$$

The density $d(\mathbf{x})$ associated with individual $\mathbf{x}$ is computed as the inverse of the distance to

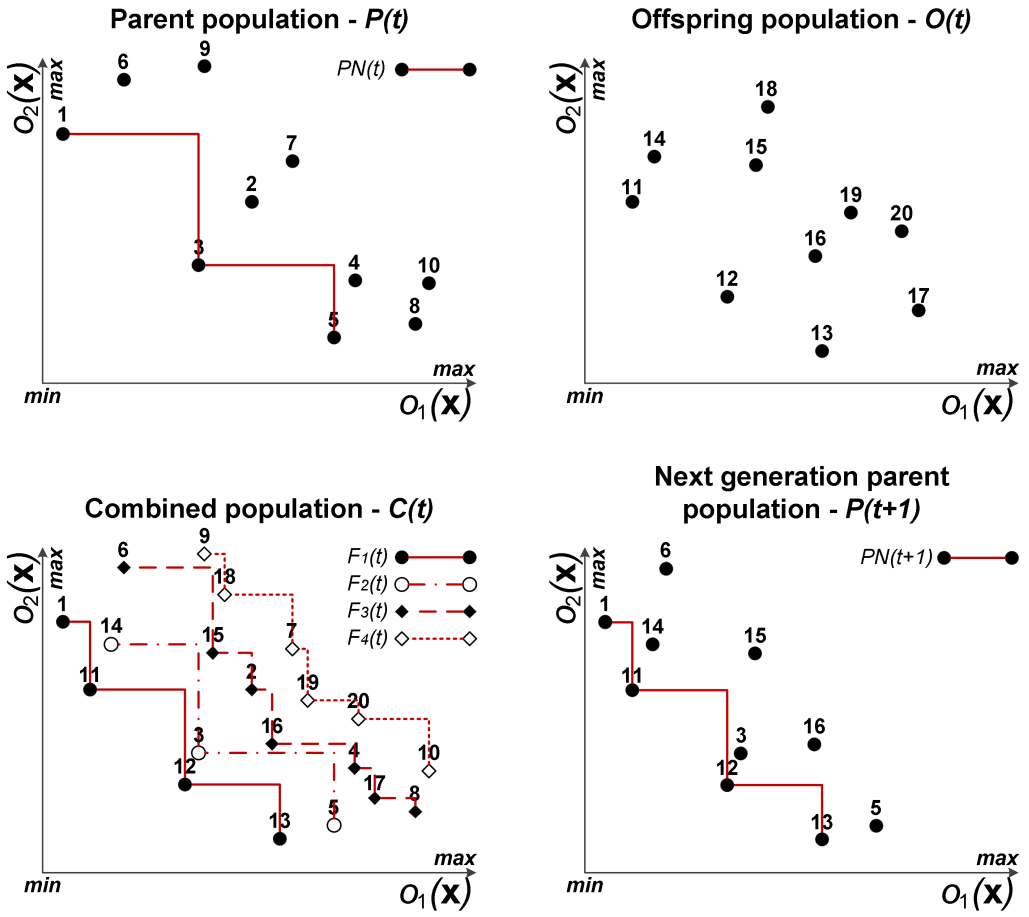


Figure 2.3: Example of the selection for survival strategy of NSGA-II for a toy example consisting of a MOOP with two objectives and a population of size 10

the k-th nearest neighbor, i.e.,

$$d(\mathbf{x}) = \frac{1}{dist_E(\mathbf{x},k)+2} \tag{2.18}$$

where $dist_E(\mathbf{x},k)$ is the Euclidean distance in objective space between individual $\mathbf{x}$ and its k-th nearest neighbor, where $k = \sqrt{|Set|}$.

When wishing to adapt the original computation cycles to the general EA template from Algorithm 1, both NSGA-II and SPEA2 generally require:

- a random selection of the parents inside the **SELECTPARENTS** function since the highly elitist nature of both algorithms does not demand an increased selection pressure;
- the usage of two specialized genetic operators (proposed by Deb), *simulated binary crossover* (**SBX**) [67] and *Polynomial Mutation polynomial mutation* (**PM**) [68], inside the **CREATEOFFSPRING** function.

- a **SELECTFORSURVIVAL** function that implements the appropriate highly elitist Goldberg-inspired selection strategy.

After more than a decade since their emergence, both NSGA-II and SPEA2 have fully proven their effectiveness and are still widely used in various application domains. In our opinion, these two algorithms are, by far, the most representative approaches of successful early-day / classical MOEAs that are exclusively centered around the implementation of elitism via the $(\mu + \lambda)$ selection for survival strategy. Moreover, both approaches propose similar, two-tier, selection for survival strategies that combine Pareto ranking (as the primary quality measure) and solution crowding indices (as equality discriminants). These Pareto-based selection for survival strategies / operators, viz. **non-dominated sorting** (for NSGA-II) and **environmental selection** (for SPEA2), have themselves become undeniable landmarks within the MOEA field as they have been incorporated in several other successful algorithms.

Some special attention is also owed to the SBX and PM operators since they also became extensively used by MOEAs and single-objective EAs alike.

The SBX operator must be applied on two parent individuals (not: $\mathbf{x^a}$ and $\mathbf{x^b}$) in order to obtain two offspring (not: $\mathbf{y^a}$ and $\mathbf{y^b}$). The stated design goal of SBX is to replicate on real-valued arrays a key characteristic displayed by the single-point crossover operator when applied on binary encoded individuals: *"the common interval schemata that exist between the parents are preserved in the offspring"* [69], i.e., the spread (in objective space) that exists between $\mathbf{x^a}$ and $\mathbf{x^b}$ is likely to be preserved between $\mathbf{y^a}$ and $\mathbf{y^b}$. SBX is based on a continuous probability distribution $P(\alpha, \beta)$ given by:

$$P(\alpha,\beta) = \begin{cases} \frac{\alpha+1}{2}\beta^{\alpha} & \text{, if } \beta \in [0,1] \\ \frac{\alpha+1}{2}\frac{1}{\beta^{\alpha+2}} & \text{, if } \beta > 1 \end{cases} \tag{2.19}$$

where the non-negative parameter α is the *crossover index* that controls the shape of the probability distribution function. A large value of α (e.g., > 5) gives a higher probability for creating near-parent offspring, a small value of α (e.g., <2) allows for the frequent creation of offspring that are farther away from their parents and values of the crossover index between 2 and 5 help SBX replicate the search behavior (i.e., probability distribution) of the single-point crossover operator. The SBX crossover procedure contains three steps:

1. a number $u \in [0,1]$ is randomly generated

2. the parameter δ is determined such that:

$$\int_0^{\delta} P(\alpha,\beta)\,\mathrm{d}\beta = u \tag{2.20}$$

3. the individual variable values of the offspring encoding are determined using the formulae:

$$\begin{aligned} y_i^a &= \frac{1}{2}(x_i^a + x_i^b - \delta|x_i^a - x_i^b|) \\ y_i^b &= \frac{1}{2}(x_i^a + x_i^b + \delta|x_i^a - x_i^b|) \\ &\text{where } i \in \{1,\ldots,n\} \end{aligned} \tag{2.21}$$

In time, applying SBX with a crossover index of 20 has become the standard in both NSGA-II and SPEA2. This is because creating near-parent solutions is advantageous in the context of a highly elitist selection for survival mechanism both in the initial part of the algorithm, (when the parents are spread, the generated offspring will be too $\rightarrow$ exploration), as well as towards the end of the optimization run (when the parents are close, the generated offspring will have the overall effect of intensifying the search).

The PM operator is applied on a given individual $\mathbf{x}$ in order to obtain a changed (mutated) encoding $\mathbf{x^{mut}}$. The mutation rule for every real-valued component (variable) $i \in \{1,\ldots,n\}$ in the encoding is:

$$x_i^{mut} = \begin{cases} x_i + \psi_i(u_{D_i} - l_{D_i}) & \text{, with probability } p_{mut} \\ x_i & \text{, with probability } 1 - p_{mut} \end{cases} \tag{2.22}$$

where

$$\psi_i = \begin{cases} (2r)^{\frac{1}{\gamma+1}} - 1 & \text{, if } r < 0.5 \\ 1 - (2-2r)^{\frac{1}{\gamma+1}} & \text{, otherwise} \end{cases} \tag{2.23}$$

and r is a random number uniformly distributed in [0,1], l_{D_i} and u_{D_i} are the lower and upper bounds of D_i – the domain of variable x_i, γ is the mutation distribution index and p_{mut} is the mutation rate. The literature recommended values for γ and p_{mut} in NSGA-II and SPEA2 are 20 and $1/n$.

2.3.4 DEMO and GDE3

differential evolution (**DE**) was introduced by Storn and Price in [70] as a very efficient, population-based, global stochastic optimization method. By design, DE is especially suitable for optimization problems that have real-valued objective functions.

When comparing with most other evolutionary paradigms, DE is motivated more by practical aspects than by biological ones. As such, DE proposes the replacement of the standard evolutionary model (parent selection $\rightarrow$ recombination $\rightarrow$ mutation) with a more streamlined approach able to deliver both faster convergence and robustness. At each generation, mutation and crossover are applied in order to obtain one offspring (*trial vector*) for each member of the current population (i.e., no parent selection is required). Generally, in order to construct a trial vector for a given parent individual x, three or more individuals are chosen from the current population. The only constraint is that they must be different than the parent x. Then, a special DE mutation operator is applied on the selected individuals in order to create a mutant individual. Finally, a standard crossover operator (binary or polynomial) is applied on the resulting mutant and on the original parent x in order to

produce the trial vector (i.e., the offspring). If the fitness of the offspring is better than that of its parent, the former will replace the latter in the population of the next generation.

Given the *"outstanding"* results displayed by DE on single-objective optimization problems [71], several researchers attempted to exploit the very good search performance exhibited by some DE operators and introduce MOEAs that incorporate DE strategies. Among them, the most noteworthy early attempts were **Differential Evolution for Multiobjective Optimization** (**DEMO**) ([72]) and the third version of the **Generalized Differential Evolution Optimization** (**GDE3**) method ([73]). The approaches are very similar as they both propose the replacement of the SBX and PM operators with various DE variants, while maintaining the elitist Pareto-based selection for survival mechanisms introduced by NSGA-II and SPEA2. Some convergence benchmark tests, like [72] and [74], have show that even this rather straightforward application of DE in MOEAs helps to explore the decision space far more efficiently for several classes of MOOPs.

Given the mentioned particularities of the DE evolutionary model, the implementation of the DEMO and GDE3 computational models using the general EA template from Algorititm 1 would require:

- a generational strategy ensured by the setting $popSize = genSize$;
- a **SELECTPARENTS** function that at every call (during a given generation) will return a different member of the parent population and the individuals required for construncting the mutant and the trial vectors;
- a **CREATEOFFSPRING** function that uses DE mutation and crossover operators to create an offspring trial vector and that will return (I) either this offspring individual if it Pareto-dominates its parent, (II) either the parent individual if it Pareto-dominates the offspring, or (III) both offspring and parent individuals if they are in a Pareto non-domination relationship;
- a **SELECTFORSURVIVAL** function that implements either the non-dominated sorting procedure of NSGA-II or the environmental selection mechanism proposed by SPEA2.

2.3.5 MOEA/D

As it may have become obvious by now, the emergence of usable MOEAs in the early 90's has also started a veritable rift in the MOO community. Some chose to exclusively focus on these new stochastic approaches as they generally proved able to deliver quality *PN*s after single runs, while others continued to work and develop the more classical solving methods described throughout Section 2.2.

Although some attempts to bridge the two communities have been made over the years (e.g., the MOGLS approach from 2002 of Jaszkiewicz [75]), the first very successful approach that combined methods from both sides was the **Multiobjective Evolutionary Algorithm based on Decomposition** (**MOEA/D**) proposed by Zhang and Li in 2007 [76]. The algorithm explores the traditional a posteriori MOO strategy of decomposing the original MOOP into several single-objective sub-problems that are obtained by applying objective

aggregation methods that use different weighting vectors. However, MOEA/D applies this strategy inside an evolutionary computation framework that tries to simultaneously solve all the single-objective sub-problems during the run. The idea is that each individual in the population is interpreted as the solution to one or more of these sub-problems and after the end of the optimization, provided that the weighting vectors were well chosen, the union of all these individuals (i.e., the current population) should offer a good approximation of the *PS*. One key feature of the evolutionary cycle of MOEA/D is that it imposes neighborhood restrictions on potential parent individuals meaning that a given sub-problem SP_i is optimized by only considering those individuals who represent solutions to sub-problems defined using weighting vectors that are in the vicinity of the weighting vector that defines SP_i. The vicinity of each weighting vector is determined based on the Euclidean distance.

It is noteworthy that MOEA/D proposes a totally different paradigm to multi-objective optimization than most of the previous evolutionary approaches as its computation cycle is not directly regulated by a Goldberg-inspired elitist selection for survival scheme. Instead, MOEA/D relies on:

1. suitable objective aggregation functions like weighted sum (2.9), weighted Tschebycheff (2.10) or the boundary intersection method proposed in [36] for defining meaningful sub-problems that lead the search towards the *PS* of the MOOP;

2. well chosen weighting vectors and neighborhood sizes for niching the search and discovering well-spread *PN*s.

When it comes to the genetic operators used during the search, the first MOEA/D proposal mentions SBX and PM, while in [77], the authors also describe a DE based variant, namely MOEA/D-DE. The general optimization framework proposed by MOEA/D has proven quite successful, especially when dealing with problems with complicated Pareto-optimal sets, and a version of MOEA/D-DE [78] won the CEC2009 Competition dedicated to multi-objective optimization. As such, this MOO optimization method is considered state-of-the-art by many researchers in the field.

Apart from the DE related aspects, a concrete (but by no means optimal) implementation of MOEA/D-DE given the general EA template from Algorithm 1 demands:

- a codification of individuals that stores both the typical n-dimensional binary / real-valued variable array as well as the weighting vector used for aggregating objectives and thus for defining the sub-problem the individual is meant to solve;

- an **INITIALIZEPOPULATION** method that initializes both the random initial population (i.e., $P(0)$) as well as all the (evenly spread) weighting vectors required during the search process;

- a DE-based **SELECTPARENTS** function that is neighborhood-bound;

- a DE-based **CREATEOFFSPRING** function that assigns to each offspring trial vector the weighting vector of its parent;

- an **EVALUATEFITNESS** method that, based on the corresponding weighting vector, computes the fitness value associated with an individual via the preselected objective aggregation function;
- a **SELECTFORSURVIVAL** function that admits offspring trial vectors in the population of the next generation, provided they are better at solving the defined subproblems than current members of the population (N.B. After the selection process, each original weighting vector must be represented in the population of the next generation).

2.3.6 Remarks Regarding MOEAs

The introduction in 2002 of what Coello Coello [64] labeled as *"second generation"* MOEAs, viz. NSGA-II, SPEA2 and PAES, also marked the start of an age where these MOO solvers became ever more poplar in a wide range of application domains [47]. Since any application of a general MOEA on a given real-life scenario usually highlights the need for improvements, modifications, hybridization with other search techniques, the wide spread usages of MOEAs triggered a veritable publishing boom in the field. Therefore, by December 2014, Google Scholar reports over 12.900 citations for NSGA-II, over 4200 citations for SPEA2, over 550 citations for GDE3 and DEMO combined and over 1000 citations for MOEA/D. Furthermore, in the time period taken to conduct the research reported in this thesis (i.e., 2011-2014), Google Scholar reports over 450 new publications that contain either *"multi-objective evolutionary algorithm"*[8] or *"MOEA"* in their title.

Obviously, when confronted with such a scholarly deluge, one could not hope to cover in the few pages of the previous sections all the trends and flavors in the field. However, we have tried to present the three major directions and milestones of MOEA research (i.e., Pareto-based elitism, DE-based search, decomposition-based EAs) by describing their most well-known and successful exponents via a unitary EA template. The primary aim of this approach was to familiarize the reader with the current state-of-the-art in MOEA design and to provide a rather simple but nevertheless accurate placement of these methods within the much larger family of evolutionary-inspired optimization techniques.

The secondary aim of our MOEA presentation layout was to try to separate the general underlying (Darwinian-inspired) search/optimization strategy from the particularities imposed by the MOO application domain. This is because the domain-specific parts (i.e., Goldberg-inspired selection for survival strategies, external archives, simultaneously solving linked decomposition problems, etc) can also be fitted on top of other well-known population based (bio-inspired) global optimization paradigms like ant colony optimization [79] particle swarm optimization (PSO) [80] and artificial immune systems (AIS) [81]. A posteriori multi-objective optimization solvers have also been developed on top of trajectory-based stochastic methods like simulated annealing (SA) [82] and some successful MOO strategies even use burst (i.e., random search) algorithms [83] as solution generators.

While two of the three MOO enhancements we propose are specifically tailored for the much more popular MOEA paradigm, the on-the-fly surrogate-modeling improvements

[8] We also considered the alternative spelling "multiobjective".

we discuss in Chapter 4 are inherently suitable for virtually any MOO solving strategy (including the classical methods presented in Section 2.2). The new solution assessment strategy we describe in Section 3.3 can also be used to evaluate numerous (a posteriori) MOO techniques. In order to mark the difference between MOEA-specific parts and more general ideas, whenever referring to an a posteriori general MOO solver, we shall use the term *multi-objective optimization algorithm* (**MOOA**).

Chapter 3

Test Problems and Performance Evaluation

3.1 Test Problems

3.1.1 Artificial Test Problems

Over the years, several artificial MOOPs sets have been proposed by various practitioners/scientists from the field of multi-objective optimization with the goal of providing general benchmarks for evaluating the (comparative) performance of MOOP solvers. In time, some of these artificial problems have become a more or less standard way of evaluating the general performance of any new MOOA and its behavior with regard to specific domain challenges like concave PF_{true}s, disconnected PF_{true}s, biased search spaces, etc.

The main advantage of using an artificial problem is that the PF_{true} (or an equivalent near-perfect finite approximation) is known and that the level of difficulty can be well estimated, if not even fully understood. As such, several artificial problems can be easily aggregated in order to construct "meaningful" problem sets for benchmarking MOOAs. The importance of having "meaningful" test scenarios is quite intuitive as several authors (e.g., [84] and [13]) warn that results obtained on "pedagogical"/"academic"/"toy" problems offer very little insight into real-world performance. A good benchmark MOOP problem set should contain problems with various *PF* characteristics and with wide-ranging degrees of difficulty.

For the purpose of evaluating the multi-objective optimization techniques used throughout this work, we rely on MOOPs from a set of 25 artificial problems that we have selected from five different well-known[1] sources. These problems are:

- **KSW10** - a classic (but not trivial) MOOP based on Kursawe's adaptation [85] of two functions provided by Ackley and Schwefel;
- **ZDT3** and **ZDT6** from the problem set described in [86];

[1] In December 2014, the articles describing these MOOP problem sets had over 4500 citations (in total) according to Google Scholar.

Table 3.1: Characteristics of the first 16 artificial MOOPs we compiled in the benchmarking set

MOOP	Variable #	Objective #	Separability	Multimodality	Geometry	Challenges
DTLZ1	7	3	✓*	✓	linear surface	many-to-one
DTLZ2	12	3	✓*	×	concave surface	many-to-one
DTLZ3	12	3	✓*	✓	concave surface	many-to-one
DTLZ4	12	3	✓*	×	concave surface	biased, many-to-one
DTLZ6	12	3	✓	×	concave 3D curve	biased, many-to-one
DTLZ7	22	3	✓	✓	disconnected surfaces	-
KSW	10	2	✓	✓	concave, disconnected	-
WFG1	6	2	✓	×	convex, mixed	biased
WFG2	6	2	×	✓	convex, disconnected	-
WFG3	6	2	×	×	line	-
WFG4	6	2	✓	✓	concave	-
WFG7	6	2	✓	×	concave	biased
WFG8	6	2	×	×	concave	biased
WFG9	6	2	×	✓	concave	biased
ZDT3	10	2	✓	✓	disconnected 2D curve	-
ZDT6	10	2	✓	✓	concave curve	biased, many-to-one

- **DTLZ1**, **DTLZ2**, **DTLZ4**, **DTLZ6**, and **DTLZ7** from the problem set proposed in [87];
- **WFG1**, **WFG2**, **WFG3**, **WFG4**,**WFG7**, **WFG8**, and **WFG9** from the problem set proposed in [88];
- all nine problems (**LZ09-F1**, **LZ09-F2**, ..., **LZ09-F9**) from the problem set described in [77];

and they are meant to provide a rather high average degree of difficulty although neither problem contains any result-related constraints – i.e., constraints of the type (2.2) or (2.3). The constrained problems we use in our tests are the five real-life industrial MOOPs described in the next section. We have chosen this segregated approach because we mainly employ the artificial problems to test the general/default search behavior of the studied MOOAs without considering the impact of various constraint handling strategies. This is because many of these strategies (like the very widely used penalty functions [89]) are algorithm-independent [90] and can be retrofitted with ease to nearly all MOOP solvers.

In Table 3.1, we compiled from [88] six general characteristics to provide insight into the complexity of each of the first thirteen artificial problems we integrated in our problem set. The characteristics we selected are:

1. *Variable #* - the length of the variable vector (i.e., n - the size of the decision space).

2. *Objective #* - the number of objectives featured by the MOOP [i.e., m from (2.1)].

3. *Separability* - as defined by Huband et al.[2] in [88], is the property of the MOOP to contain only single-objective functions that can be individually minimized by independently (one at a time) optimizing each variable $x_i \in \mathbf{x}$, $i \in \{1,\ldots,n\}$, $\mathbf{x} \in D^n$. In [88], the authors state that *"finding at least some points on the Pareto optimal front for a separable problem tends to be simpler than for an otherwise equivalent nonseparable problem."*

4. *Multimodality* - is the property of the MOOP to contain at least one single-objective optimization function that has several local optima (i.e., that is multimodal). Multimodal MOOPs are generally considered more difficult to solve than problems that only feature unimodal objective functions.

5. *Geometry* - concerns the shape of the PF_{true}. This can be: linear, convex, concave, disconnected, 2D/3D curve, surface. Some MOOAs are known to have difficulties exploring concave and disconnected PFs so it is very important to integrate MOOPs with these types of fronts when constructing challenging problem sets.

6. *Challenges* - are specific design features that make finding good PNs harder. The most common challenges are biased search spaces (vectors are more densely distributed around certain parts of the PS or further away from the PS) and many-to-one mappings ($O(\mathbf{x^a}) = O(\mathbf{x^b})$ with $\mathbf{x^a}, \mathbf{x^b} \in D^n$ and $\mathbf{x^a} \neq \mathbf{x^b}$).

In [88] it is stated that although DTLZ1 – DTLZ4 have non-separable objectives, the objectives of these problems are classified as separable because optimizing them one variable at a time will identify at least one of the multiple global optima from PF_{true} that are induced by the many-to-one characteristic.

In literature, very little information is provided about the shape (spatial distribution) of the PSs corresponding to the MOOPs from Table 3.1 although for some (e.g., the ZDT set from [86]) the PSs are known to have very simple shapes.

Okabe et. al. [93], were the first to argue the need to construct MOOPs with specifically designed PSs. Li and Zhang were some of the few who built on this idea and in [77] they proposed nine MOOPs with arbitrarily prescribed complicated PS shapes: the LZ09 problem suite. N.B. The fact that problems from this test suite generally display complicated PSs does not necessarily mean that they also display a complicated geometry of their PF_{true} shapes.

The problem LZ09-F6 has three objectives and all the other problems have two. The problems LZ09-F7 and LZ09-F8 have 10 variables, while the rest have 30. The PF_{true} shape of LZ09-F1 – LZ09-F5, LZ09-F7, and LZ09-F8 is a convex 2D curve. The PF_{true} shape of LZ09-F9 is a concave 2D curve in $[0,1]^2$, the PF_{true} shape of LZ09-F6 is a concave surface

[2] In EA literature (e.g.,[91], [92]) separability is usually defined in a less restrictive form by demanding that the objective functions should not contain any non-linear interactions between variables.

in $[0,1]^3$, and the PF_{true} shapes of the other 7 problems are convex 2D curves in $[0,1]^2$. Although separable (according to the definition in [88]) and easy to solve analytically, the LZ09 problems have proven extremely difficult for iterative approximation methods and they are quite challenging for most state-of-the-art MOOAs. Because of this, we have added the entire LZ09 suite to our problem set.

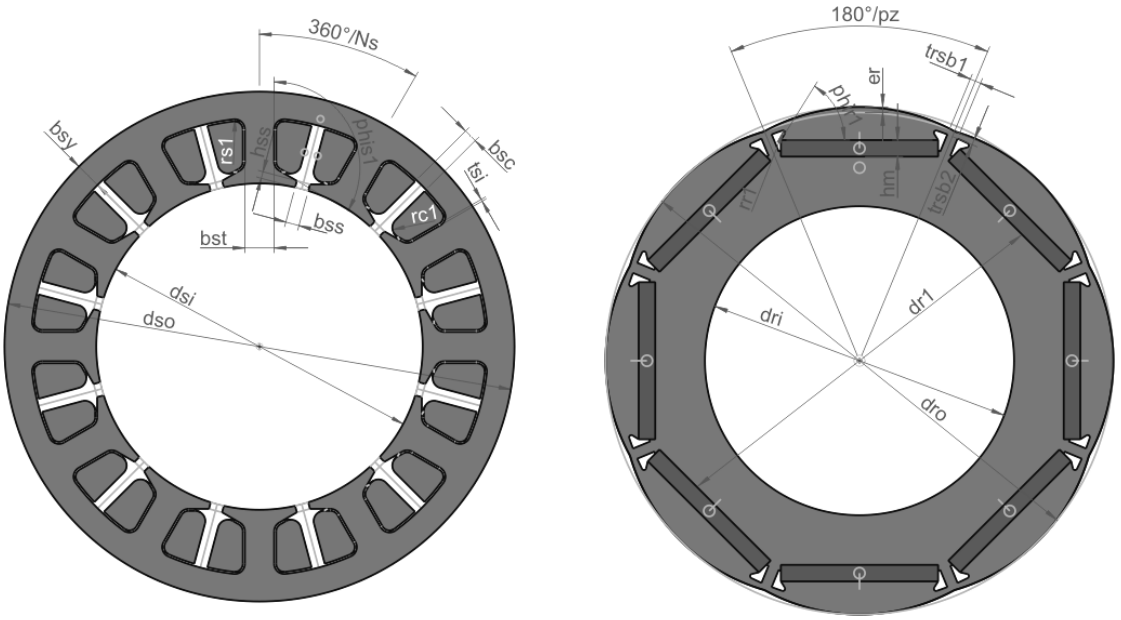
3.1.2 Industrial Test Problems

Because our research efforts are primarily motivated by practical applications of multi-objective optimization, analyzing MOOA performance on artificial problems is mainly intended to help with the algorithm design/prototyping and parameter tuning stages. The final goal is to obtain robust methods that are able to successfully tackle real-life MOOPs.

As mentioned in Chapter 1, the main motivation of this work lies in trying to improve the design process of electrical machines. This process generally contains at least an optimization of the geometric dimensions for a pre-selected topology. For example, Figure 3.1 presents the cross-section of an electrical drive featuring a slotted stator with concentrated coils and an interior rotor with buried permanent magnets. Some of the geometric parameters of this assembly (e.g., d_{si}, b_{st}, b_{ss}, etc.) are also presented. Depending on the actual design goal, some of these parameters need to be varied in order to achieve a cost-efficient motor that displays good operational behavior. In other cases, because of the fast moving international raw materials market, motor manufacturers want to investigate the behavior of target parameters with regard to different types of materials. Sometimes, a comparative analysis of different motor topologies is also required during the design stage. As a result, we are dealing with industrial multi-objective optimization scenarios where, apart from several conflicting objectives, there is also an increasing number of input parameters (variables) an electrical engineer needs to consider when optimizing a motor.

Industrial MOOPs from the field of electrical drive design are particularly challenging because of the way the performance of a design is assessed. Thus, even though the physics behind the process is known for centuries, the behavior of the materials used to construct the electrical drive cannot be modeled linearly and the evaluation of an elicited design is done via one or more computationally-intensive FE simulation. These simulations are solving non-linear differential equations in order to obtain the values of the **target parameters** associated with the design parameter vector (i.e., $\mathbf{x} \in D^n$). These target parameters are the objectives functions [as defined in (2.1)] and the constraints [as defined in (2.2) and / or (2.3)] of the optimization. Before performing the FE simulations, an intermediate modeling stage is required in order to construct, starting from the design parameter vector, a 2D or 3D model and the corresponding mesh. This mesh of the electrical drive will act as input for the no-load FE simulation and several FE simulations where various current-load configurations are considered. Some problem-dependent post-processing is also required in order to obtain the values of the target parameters. The full evaluation process is presented Figure 3.2 and can take from 2 minutes to 20 minutes even when computing all the required FE simulations in parallel.

In all our industrial MOOPs tests, we used the software package FEMAG™[94] to perform the FE simulations (i.e., calculation of 2D problems on electromagnetics) – all the



(a) Stator cross-section for IndMOOP1

(b) Rotor cross-section for IndMOOP1

Figure 3.1: Cross-sections with the geometric dimensions of the stator and the rotor for a motor with an interior rotor topology with embedded magnets

tasks with light red backgrounds from Figure 3.2.

For the present thesis, we consider five electrical drive design MOOPs. While the first one describes, a more or less, standard problem in the field and we can provide more detailed information regarding it, the other four regard current "work in progress" at the LCM which can lead to patent applications or commercial products. Therefore, we can only give limited information concerning the latter industrial problems.

IndMOOP1 features a motor with an interior rotor topology with embedded magnets. The stator and rotor cross-sections are presented in Figure 3.1. The corresponding design parameter vector has a size of 6 and is given by:

$$\mathbf{x} = \left(\; h_m, \quad \alpha_m, \quad e_r, \quad d_{si}, \quad b_{st}, \quad b_{ss} \; \right) \tag{3.1}$$

where all parameters are illustrated in Figure 3.1 except for α_m, which denotes the ratio between the actual magnet size and the maximum possible magnet size as a result of all other geometric parameters of the rotor. The goal is to simultaneously optimize four unconstrained objectives:

$o_1(\mathbf{x}) = -\eta_{ed}(\mathbf{x})$ – where $\eta_{ed}(\mathbf{x})$ is the efficiency of the electrical drive. Since we have formulated (2.1) as a minimization problem, the simplest option is to minimize negated efficiency values.

$o_2(\mathbf{x}) = T_{cogPP}(\mathbf{x})$ – the peak-to-peak value of the motor torque for no current excitation. This parameter shows the behavior of the electrical drive at no-load operation

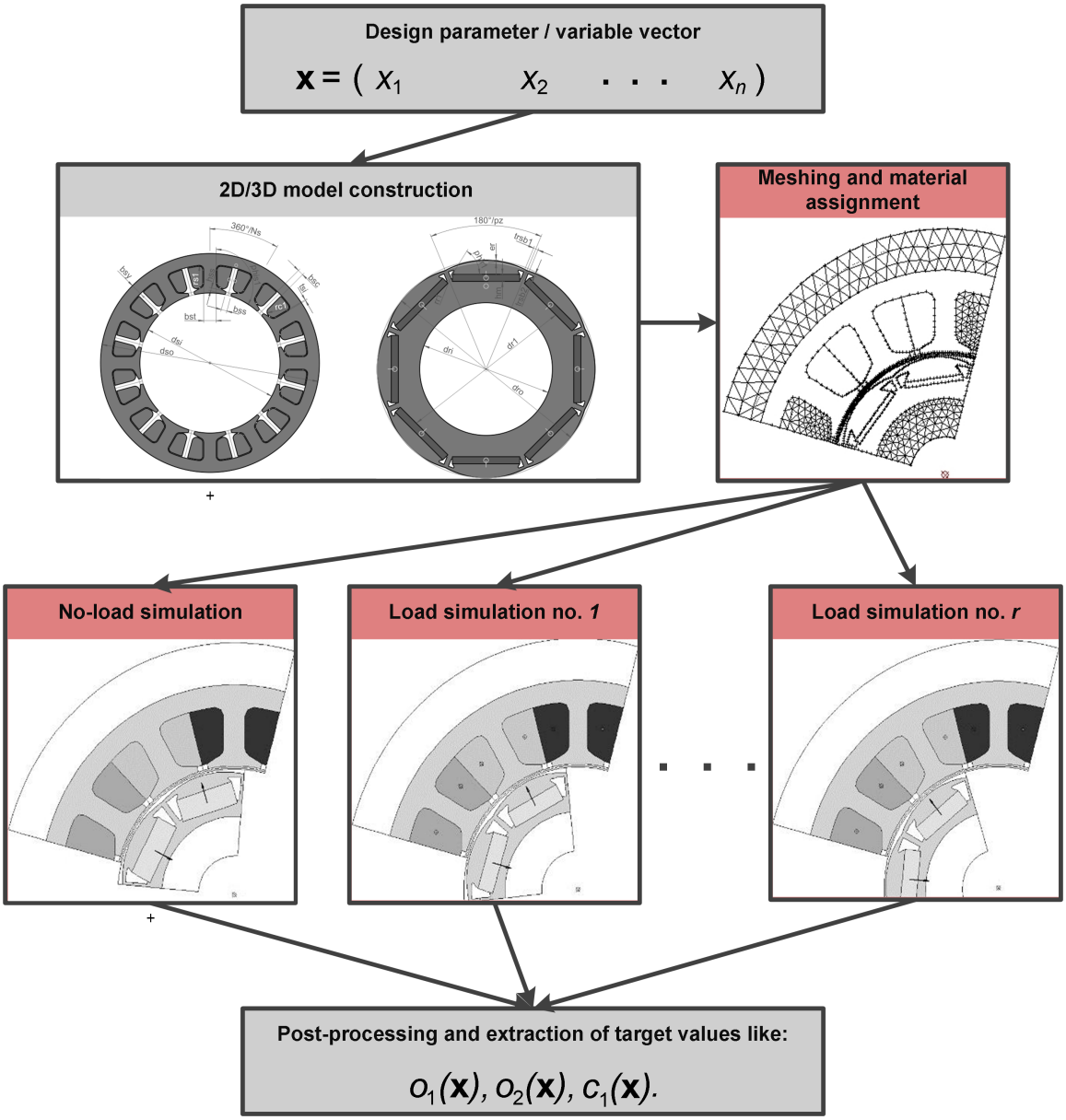


Figure 3.2: The general stages of the electrical drive performance evaluation process in the case of the considered industrial MOOPs.

and should be as small as possible in order to minimize noise and vibrations that are the result of torque fluctuations.

$o_3(\mathbf{x}) = TC(\mathbf{x})$ – the overall cost of the materials necessary for building the motor. Minimizing this objective is a common task in most electrical drive MOOPs we deal with.

$o_4(\mathbf{x}) = T_{rippPP}(\mathbf{x})$ – the equivalent of $T_{cogPP}(\mathbf{x})$ at load operation. The values of this objective should also be as small as possible.

The evaluation of one motor design in the case of IndMOOP1 requires 6 FE simulations, each with a different current-load.

IndMOOP2 concerns a motor with an exterior rotor. The design parameter vector contains 7 geometric dimensions. The aims of the optimization is to minimize the total losses of the system at load operation [$o_1(\mathbf{x})$] and the total mass of the assembly [$o_2(\mathbf{x})$]. Simultaneously, other characteristics like cogging torque and manufacturing costs must be maintained below a certain threshold. In the case of this MOOP, the first objective is constrained [i.e., internal result-related constraint as described in (2.2)] and the second is unconstrained. The problem also contains an external result-related constraint, $c_1(\mathbf{x})$ imposed on the overall manufacturing costs of a given design [as described in (2.3)].

IndMOOP3 is also related to a motor topology that features an exterior rotor. The design parameter vector has a size of 10 and the problem features four constrained objectives: the total axial length of the assembly [$o_1(\mathbf{x})$], the total mass of the assembly [$o_2(\mathbf{x})$], the ohmic losses in the stator coils [$o_3(\mathbf{x})$] and the total losses as a result of material hysteresis and eddy currents in the ferromagnetic parts of the motor [$o_4(\mathbf{x})$].

IndMOOP4 and **IndMOOP5** are both formulated for the same motor that features an interior rotor topology. Apart from the size of the considered search space D^n, the only major difference between the two MOOPs is that the first one evaluates the performance of the assembly at 1500 rotations per minute (RPM) and the second one evaluates it at 3000 RPM. In both cases the parameter vector contains 22 real-valued variables that must be configured in order to optimize four unconstrained objectives that regard efficiency and production costs. The evaluation of each design for IndMOOP4 and IndMOOP5 requires 18 different current-load FE simulations.

All the five industrial MOOPs suffer (some more than others) from the formalization constraints described in Section 2.1.2. This means that, under various conditions, the result of the performance evaluation process described in Figure 3.2 can also be the labeling of the input variable vector as an *error / infeasible* design.

3.2 Performance Measures for MOOPs

3.2.1 Primary *PN* Quality Indicators

From a mathematical perspective, in the case of single-objective optimization problems, assessing the performance of (iterative) solving techniques largely requires just to analyze

the best value found during the searches – i.e., the minimal value in the case of minimization problems. When using most of the classical MOO optimization methods discussed in Section 2.2.7 – with the notable exception of a posteriori techniques, performance analysis was also as simple as the DM having to decide if the resulting solution presented the acceptable trade-offs.

When considering a posteriori MOO methods, since the expected general solution of the MOOP is more complicated as it comes in the form of a set of trade-off solutions (the *PS*) that must be approximated, assessing performance is also far more challenging. The goodness of a *PN* approximation produced by a MOOA boils down to two characteristics of its objective space representation (i.e., the corresponding *PF*):

1. **Convergence** - shows how close (in Euclidean space, for instance) is the *PF* to the PF_{true} of the problem to solve.

2. **Diversity** - indicates how well-spread across the PF_{true} are the solutions from the given *PF*.

Figure 3.3 gives a summary illustration why simultaneously achieving both good convergence and good diversity is of the utmost importance in the case of MOOAs.

Over the years, several quality indicators aimed at measuring the two criteria have been proposed. In [95], the authors propose a classification showing that some indicators are designed to predominantly quantify convergence or diversity and a few take into account both criteria. In light of this, the Pareto quality indicators that, over the years, have gained the strongest foothold in the MOO community can be labeled as:

- *convergence quantifiers*: the epsilon measure (not: Ind_ε) [96], the generational distance (not: Ind_{GD}) [97];

- *diversity quantifiers*: the spread metric (not: Ind_S) [69] and the generalized spread metric (not: Ind_{GS}) [98]

- *both convergence and diversity quantifiers*: the inverse generational distance (not: Ind_{IGD}) [97] and the Ind_H [99].

Let us consider an arbitrary MOOP, defined as in (2.1), with a known Pareto-optimal set *PS*, whose objective space representation is PF_{true} with $|PF_{true}| = u$ and a candidate Pareto non-dominated set PN_c, whose objective space representation we mark by PF_c with $|PF_c| = v$. The previously listed primary quality indicators determine if PN_c is a good approximation of *PS* in the following manner:

- The epsilon measure aims to quantify the smallest distance $\varepsilon \in \mathbb{R}$ needed to translate every solution from PF_c such that it Pareto dominates at least one solution from PF_{true}. More formally:

$$\begin{aligned} Ind_\varepsilon(PF_c) &= \inf\{\varepsilon \in \mathbb{R} \,|\, \forall \mathbf{p}^* \in PF_{true},\ \exists \mathbf{p^c} \in PF_c : \mathbf{p^c} \preceq_\varepsilon \mathbf{p}^*\} \\ \text{where } \mathbf{p^c} \preceq_\varepsilon \mathbf{p}^* &\iff p_i^c < \varepsilon + p_i^* \ \forall i \in \{1,\ldots,m\} \end{aligned} \tag{3.2}$$

  A value as small as possible of $Ind_\varepsilon(PN_c)$ is obviously preferred as it indicates a very high quality Pareto non-dominated set.

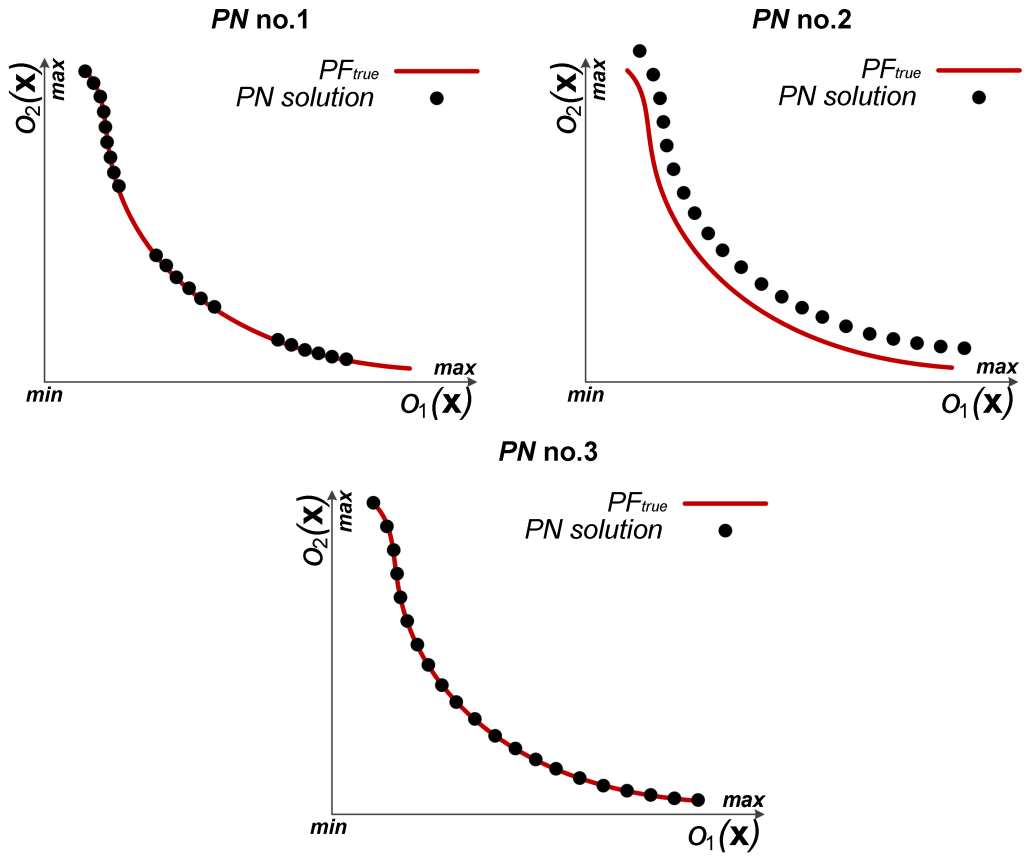


Figure 3.3: Example of *PF*s corresponding to three different *PN*s of 21 individuals and the way they compare to the PF_{true} of the MOOP: the *PF* corresponding to *PN* no. 1 displays good convergence and bad diversity, the *PF* corresponding to *PN* no. 2 displays bad convergence and good diversity, and the *PF* corresponding to *PN* no. 3 displays both good convergence and good diversity.

- The generational distance indicator measures how far from PF_{true} are the individuals that make up PF_c by computing:

$$Ind_{GD}(PF_c) = \frac{\sqrt{\sum_{\mathbf{p^c} \in PF_c} dist_E(\mathbf{p^c}, PF_{true})}}{v} \tag{3.3}$$

  where $dist_E(\mathbf{p^c}, PF_{true})$ is the Euclidean distance in objective space between the vector $\mathbf{p^c}$ and its nearest neighbor from PF_{true}. Smaller values of Ind_{GD} are also preferred and a value of 0 indicates that all the members of the examined *PN* are also members of PF_{true}.

- The Ind_S indicator is generally used to compute the spread of Pareto fronts when considering MOOPs with only two objective functions via the formula:

$$Ind_S(PF_c) = \frac{e_1 + e_2 + \sum_{i=1}^{v-1} |d_i - \bar{d}|}{e_1 + e2 + (v-1)\bar{d}} \tag{3.4}$$

where $d_i, i < v$, is the distance between any two vectors from PF_c that are neighbors, $\bar{d}$ is the average of these values and e_1 and e_2 are the distances between the corresponding best (extreme) objective-wise values of PF_{true} and PF_c . Figure 3.4 contains a graphical representation of all the individual components required for the computation of Ind_S when considering the usual Euclidean distance measure.

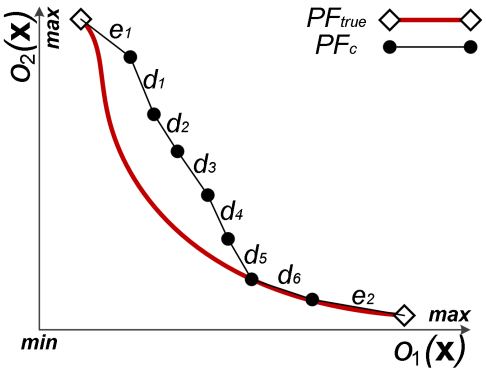


Figure 3.4: An illustration of the various distances required to compute Ind_S.

- The Ind_{GS} indicator is an extension of Ind_S for MOOPs that have more than two objective functions and is computed using the formula:

$$Ind_{GS}(PF_c) = \frac{\sum_{i=1}^{m} d(\mathbf{p}^{\{*,\mathbf{i}\}}, PF_c) + \sum_{\mathbf{p}^* \in PF_{true}} |d(\mathbf{p}^*, PF_c) - \bar{d}|}{\sum_{i=1}^{m} d(\mathbf{p}^{\{*,\mathbf{i}\}}, PF_c) + u * \bar{d}} \tag{3.5}$$

where

$$d(\mathbf{p}^*, PF_c) = \min_{\mathbf{p}^* \in PF_{true}, \mathbf{p}^{\mathbf{c}} \in PF_c, \mathbf{p}^* \neq \mathbf{p}^{\mathbf{c}}} ||\mathbf{p}^* - \mathbf{p}^{\mathbf{c}}||^2 \tag{3.6}$$

and

$$\bar{d} = \frac{1}{u} \sum_{\mathbf{p}^* \in PF_{true}} d(\mathbf{p}^*, PF_c). \tag{3.7}$$

The vector $\mathbf{p}^{\{*,\mathbf{i}\}} \in PF_{true}$ is the one that displays the best value for the single objective $o_i(\mathbf{x}), \mathbf{x} \in D^n$. The distance $d(\mathbf{p}^{\{*,\mathbf{i}\}}, PF_c)$ is computed in a similar fashion to (3.6). As with the original Ind_S, smaller values of Ind_{GS} are preferred.

- The Ind_{IGD} indicator computes the distance between PF_{true} and PF_c as:

$$Ind_{IGD}(PF_c) = \frac{\sqrt{\sum_{\mathbf{p}^* \in PF_{true}} dist_E(\mathbf{p}^*, PF_c)}}{u} \tag{3.8}$$

where $dist_E(\mathbf{p}^*, PF_c)$ is the Euclidean distance between the Pareto-optimal objective vector $\mathbf{p}^*$ and its nearest neighbor from PF_c. Smaller values of this indicator signal a better performance.

- The basic hypervolume indicator (not: Ind_H^b) measures the amount (volume) of objective space that is dominated by PF_c. This is achieved by constructing a union of the individual volumes of objective space that are dominated by each member of PF_c. For an arbitrary $\mathbf{p^i} \in PF_c, i \leq v$, the individual objective space volume it dominates can be defined as the hypercube h_i that has $\mathbf{p^i}$ and $\mathbf{p^{ref}}$ as its diagonal corners. $\mathbf{p^{ref}}$ is called an anti-optimal reference point and it has the property that either $\mathbf{p^{ref}} \not\preceq \mathbf{p^{nad}}$ or $\mathbf{p^{ref}} = \mathbf{p^{nad}}$. Formally:

$$Ind_H^b(PF_c) = \text{volume}\left(\bigcup_{i=1}^{v} h_i\right) \tag{3.9}$$

  When dealing with a MOOP with only two objectives, the visualization of Ind_H^b is easy, as shown in Figure 3.5. In this particular case, the hypercube components are rectangles in objective space and the hypervolume associated with PN_c is the area dominated by this approximation set.

  Since the numeric value of Ind_H^b is heavily dependent on the choice of the anti-optimal reference point $\mathbf{p^{ref}}$, it makes sense to compute a normalized version (not: Ind_H) as suggested by Veldhuizen in [57]:

$$Ind_H(PF_c) = \frac{Ind_H^b(PF_c)}{Ind_H^b(PF_{true})} \tag{3.10}$$

  In the case of Ind_H, a value of 1 denotes the fact that PF_c is a perfect approximation of PF_{true} while a value of 0 indicates very bad convergence (i.e., every element in PF_c is dominated by the chosen reference point $\mathbf{p^{ref}}$). $Ind_H(PF_c) = 0.73$ shows that 73% of the objective space volume that is dominated by PF_{true} is also dominated by PF_c. A 2003 study of Fleischer [100] shows that, besides being able to consider both the convergence and diversity criteria, Ind_H also presents the added advantage of having a **monotonic convergence behavior**. This means that there is a mathematical proof that the higher the Ind_H associated with a PF the closer that front approximates PF_{true}. To our knowledge, at present time, this is the only primary quality indicator for which such a valuable theoretical result exists.

Since all the primary quality indicators rely in some manner on Euclidean distances, it makes sense to apply them on normalized objective values.

It is also important to remark that all of the above indicators are applied on a single candidate PF in order to provide an estimation of the latter's quality (i.e., similarity to PF_{true}). Because of this they became known in MOO as **unary quality indicators**.

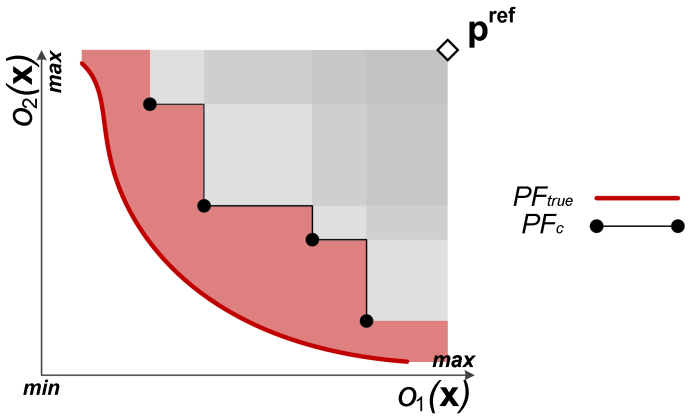


Figure 3.5: Given an arbitrary reference point $\mathbf{p}^{\mathbf{ref}}$, the hypervolume associated with PF_c is shaded in gray – darker shades correspond to regions that are dominated by more points from PF_c. The area that is dominated by PF_{true} and not by PF_c is shaded in red.

3.2.2 Considerations regarding MOO Quality Assessment

Since non-deterministic techniques have proven to be the best a posteriori MOO solvers, using only the previously described primary quality indicators for measuring algorithmic performance does not suffice. This is because multiple executions of the same non-deterministic algorithm over the same problem can produce different results. Therefore, a general single-objective EA standard was also adopted by the MOO community. It advocates that **the base performance of a particular stochastic algorithm over a given problem is given by quality indicator results averaged over several independent runs** (at least 30, usually 100). Furthermore, when comparing between different algorithms, some sort of statistical analysis should usually be applied in order to see if the observed differences between the central tendencies are significant or just the result of inherent randomness.

Unfortunately the direct translation of performance assessment principles from stochastic single-objective optimization to MOO is not as complete / meaningful as one would have hoped. This is because most quality indicators can not be implicitly used to make a clear comparison / choice between different *PN*s. For example, when given two possible *PS* approximations, PF_a and PF_b, a study from 2003 of Zitzler et al. [101] provides compelling arguments that no unary quality indicator is able (from a mathematical perspective) to undisputedly determine a winner. This means that even if $Ind_\varepsilon(PF_a) < Ind_\varepsilon(PF_b)$ and / or $Ind_{IGD}(PF_a) < Ind_{IGD}(PF_b)$ and / or $Ind_H(PF_a) < Ind_H(PF_b)$, we cannot say (based solely on these indicator values) that PN_a *"is better than"* PN_b – i.e., that every element of PN_b is Pareto-dominated by at least one element from PN_a and $PN_a \neq PN_b$. In order to make such a hard statement, a visual inspection (or a specialized point by point comparison) of the two *PN*s must also be carried out.

Nevertheless, the same study by Zitzler et al. [101] also shaded some new light on the usefulness of Ind_H. In accordance with its unique monotonic property, the Ind_H **does provide the highest comparative strength among all unary indicators** in the sense that: given the two sets, PN_a and PN_b, if $Ind_H(PN_a) > Ind_H(PN_b)$, one is assured that PN_a *"is*

not worse than" PN_b, i.e., there are elements in PN_a that are not Pareto-dominated by any element in PN_b. Coupled with the highly practical interpretation of this metric that higher values indicate that larger sections of objective space are being dominated by the analyzed PF, the above summarized findings have made many researchers – including us – to consider the Ind_H as the main / most trustful unary PN quality indicator. In light of this, **most of the comparisons made throughout this thesis are based on the scalar values produced by the Ind_H indicator**.

Confronted with the fact that unary performance indicators didn't seem suitable for ranking competing PFs, many scientists also investigated binary quality indicators. Having been specifically designed for comparing the quality of two different fronts, quantifiers like the binary ε-indicator [101], the binary hypervolume indicator [99] and the coverage indicator [102] are generally considered better metrics for MOO by several researchers [101] [64]. The major downside is that when applying these indicators on a given PN pair, two values must be computed because usually the result obtained when comparing PN_a with PN_b is different than the one obtained when comparing PN_b with PN_a (i.e., binary indicators are sensitive to the input order). This means that the number of indicator values that must be considered during a comparison is not linear but quadratic in the number of PNs one wishes to compare. On top of this, most binary measures are far more costly to compute than their unary counterparts. When also factoring the general demand to perform several independent runs for each experiment (optimization) and to perform statistical significance tests, it is easy to see why the usage of binary quality indicators, although more widely spread, has not become generalized within the community.

Apart from the **quality of the generated solution** (i.e., PN and associated PF), two other common criteria also need to be considered when analyzing the performance of a posteriori MOOAs:

- **convergence speed** - the physical time required by the algorithm to reach a PN of acceptable quality;
- **generality** - the ability of the algorithm to display a good convergence behavior over a wide range of MOOPs.

These three quality criteria can be used to evaluate all MOOAs. For example, the best solution to any MOOP can be obtained by performing a very fine-grained grid search over the entire variable space D^n. Grid searches also score very high on the generality scale as they can be applied in any optimization context. However, the excessively poor convergence speeds they exhibit, make them useless in nearly all cases. The very good MOOAs display a carefully crafted balance across all three quality criteria.

When only considering iterative optimization methods, and especially non-deterministic ones, there is a well established trend to measure convergence speed through the *number of (fitness) evaluations* (not: *nfe*) that must be performed during the optimization in order to reach good solutions. This approach generally makes sense as it can help to abstract from the performance analysis factors like, the quality of the software implementation, the general speed of implementation language, differences between hardware configurations, etc. In the case of CIMOOPs, the huge physical time required for solving the problems is

exclusively due to the very long duration of the fitness evaluation process. Hence, actively trying to reduce the necessary *nfe* is the stated general goal of two of our three proposed enhancements, namely the ones described in Chapter 5 and Chapter 6. When developing, testing and fine tuning the new MOEAs proposed in Chapter 6, we were confronted with the need for a fast and yet comprehensive strategy for evaluating the relative performance of several algorithms over benchmark problem sets that contained several (>15) MOOPs. In the end, because of the scarcity / lack of literature on the particular topic, in [1] we proposed a *nfe*-centered methodology for assessing the general *relative convergence performance* of MOOAs over large problem sets. In the next section we present an updated and slightly improved version of this methodology.

3.3 A New Methodology for Evaluating MOOAs

3.3.1 Motivation

In Chapter 1 we stated that improving the general convergence speed of MOEAs used to solve CIMOOPs is the main goal of this thesis. Now is the time to explain in detail what we mean by this.

In the previous section we saw that measuring the convergence speed of a MOOA is often reduced to the simple task of counting the *nfe* required to reach a *PN* of "acceptable quality". However, we have also seen that quantifying the quality of a *PN* is by no means trivial. Consequently, deciding if a *PN* displays an "acceptable quality" is even more of a challenge since this choice is also highly domain-dependent and subject to the DM's experience, tastes and feelings.

For instance, in the case of most publications from the field of (evolutionary) MOO, acceptable quality means finding a *PN* that provides a near perfect approximation of PF_{true} whenever dealing with artificial test problems and known solutions. The quality of the approximation is evaluated based on several quality indicators (like Ind_{IGD}, Ind_H, Ind_GS) and some type of statistical significance testing. On a different note, when considering real-life MOOPs, a newly found PN is likely to be labeled as "acceptable" if it contains a few solutions that are not dominated by any other known Pareto-approximation that has ever been discovered for the considered problem (i.e., if it expands the region of knowledge / feasibility). This is the case even if the new *PN* is actually a poor approximation of the (yet) unknown PF_{true}.

Given the computationally-intensive nature of the industrial MOOPs we aim to solve, our opinion on *PN* acceptability is motivated by deeply practical considerations: a given PN_t produced by a MOOA after nfe_t fitness evaluations is acceptable if "is not worse than" any other Pareto approximation that might have been obtained by a different algorithm after also performing nfe_t fitness evaluations. Based on this statement and considering nfe_{max} the maximal number of fitness evaluations we are able (willing) to perform, enabling a MOOA to produce acceptable *PN*s over a wide range of MOOPs for every $nfe_t < nfe_{max}$ can be considered as **a more explicit description of our main research goal**.

Although vague and somewhat scientifically naive[3] at a first glance, starting from this application-motivated perspective on *PN* quality assessment, in [1] we developed a general ranking framework for multi-objective optimization algorithms. that can offer MOO scientists and practitioners valuable insight regarding the *relative performance* of different MOOAs and of different parameterization settings for a given MOOA.

3.3.2 Hypervolume-ranked Performance Curves

In order to introduce the new ranking strategy, we consider a toy scenario that contains four MOOAs – Alg-A, Alg-B, Alg-C, and Alg-D – and test set of four artificial MOOPs – P1, P2, P3, and P4. The goal is to evaluate the comparative performance of the four solvers on the mini benchmark set. Focus shall fall on the convergence behavior of the four methods. For each MOOA-MOOP pair we performed 25 independent runs and we allowed for 50000 fitness evaluations to be performed during each run.

The more or less standard approach for evaluating the comparative convergence of the four MOOAs is to compute averages over the scalar results returned by a quality indicator applied on intermediate run-time results. An intermediate result is the best-so-far *PN* discovered by the solver – e.g., the current parent population / archive of an elitist MOEA. Let us assume that the plots from Figure 3.6 display the average run-time Ind_H-measured performance of the four algorithms on each individual MOOP. The measurements were performed after every 100 fitness evaluations. Since the individual plots show that the analyzed solvers perform quite differently across the considered problems, in order to quickly evaluate the general performance of the MOOAs, in Figure 3.7 we also present the average Ind_H-measured performance over the entire problem set. This latter plot is extremely useful because usually it quickly indicates:

- the algorithm that generally tends to converge faster (i.e., Alg-D in this case);
- the algorithm that displays the best average performance at the end of the optimization (e.g., Alg-C);
- which algorithms have a similar convergence behavior during (a certain part of) the optimization run. In our case, Alg-D and Alg-B converge quite fast (average $Ind_H \geq 0.7$ after 15000 fitness evaluations) while Alg-A and Alg-C converge slower but reach slightly better average results at the end of the runs (after 50000 fitness evaluations).

The last remark is particularly important in the case of CIMOOPs. In practice, given a limited *nfe* that can be performed, we would prefer Alg-D or even Alg-B over Alg-C and Alg-A. On the other hand, if the execution of optimization run would not face any time / *nfe* -wise restrictions, we would prefer Alg-C.

Nevertheless, the Ind_H plot from Figure 3.7 is also misleading as, like the result of any averaging operation, it masks very good or very bad performance. For example the fact that

[3] Our redefined research goal seems to ignore the famous No Free Lunch Theorem of Wolpert and Macready [103].

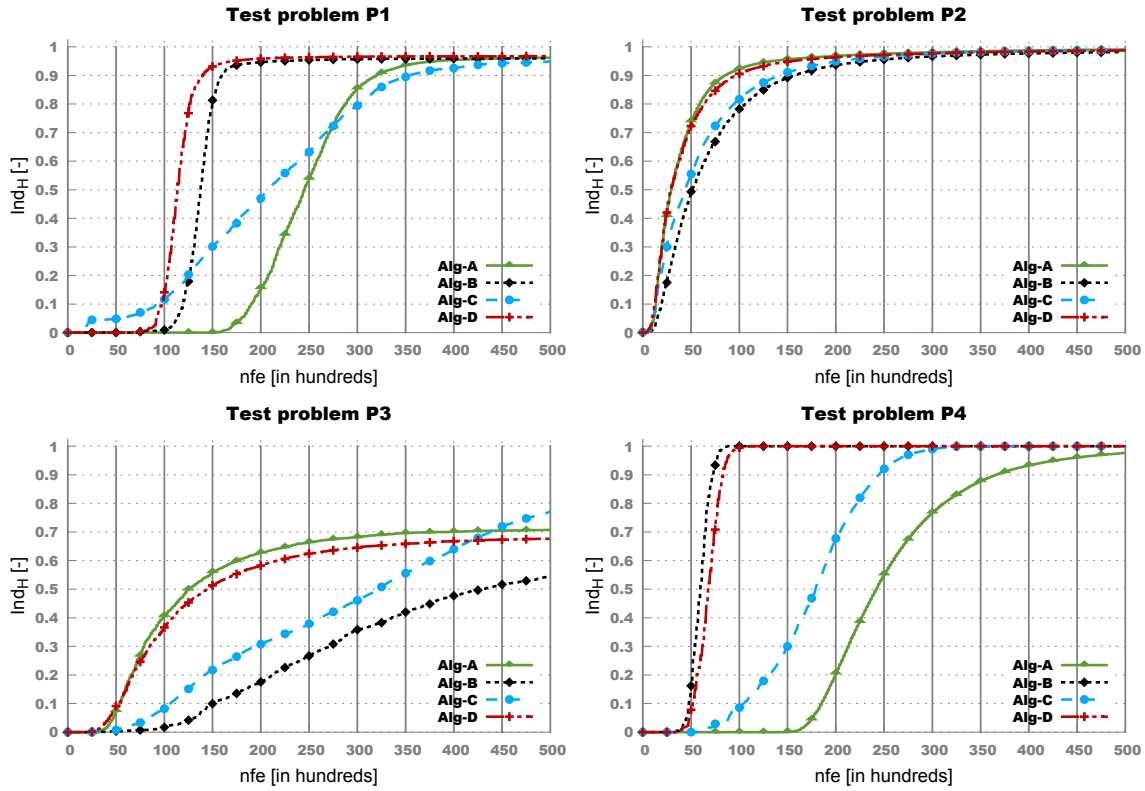


Figure 3.6: Run-time Ind_H-measured performance on the four MOOPs from the toy benchmark set

all four MOOAs are eventually able to "fully converge" (i.e., reach a Ind_H value of ≥ 0.99 for *nfe*=50000) [4] on one MOOP is totally concealed by the averaging plot. The plot also indicates that after 40000 fitness evaluations all four MOOAs reach Ind_H values between 0.85 and 0.91. Hence, we might be led to believe that all methods generally produce largely similar Pareto approximations for *nfe*=40000. In fact, at this stage of the optimization runs:

- the very good performance of Alg-A on P1, P2 and especially P3 masks the poorer performance of this MOOA on problem P4;
- the very good performance of Alg-B on P1, P2 and P4 masks the very poor performance of this MOOA on problem P3;

Of course, on our toy benchmark set, both shortfalls can easily be overcome by performing a visual / numerical inspection of all the individual MOOA-MOOP Ind_H values after 40000 and 50000 fitness evaluations. Nevertheless, when rigorous performance comparison contexts (with tens of MOOPs and several MOEAs) are concerned, this case-by-case comparison quickly becomes very cumbersome, especially when wanting to factor in the

[4] We use this rather lax and arbitrary definition of convergence because small differences between a Pareto approximation and PF_{true}, even though statistically significant, are of very little practical importance, especially when considering CIMOOPs with unknown solutions.

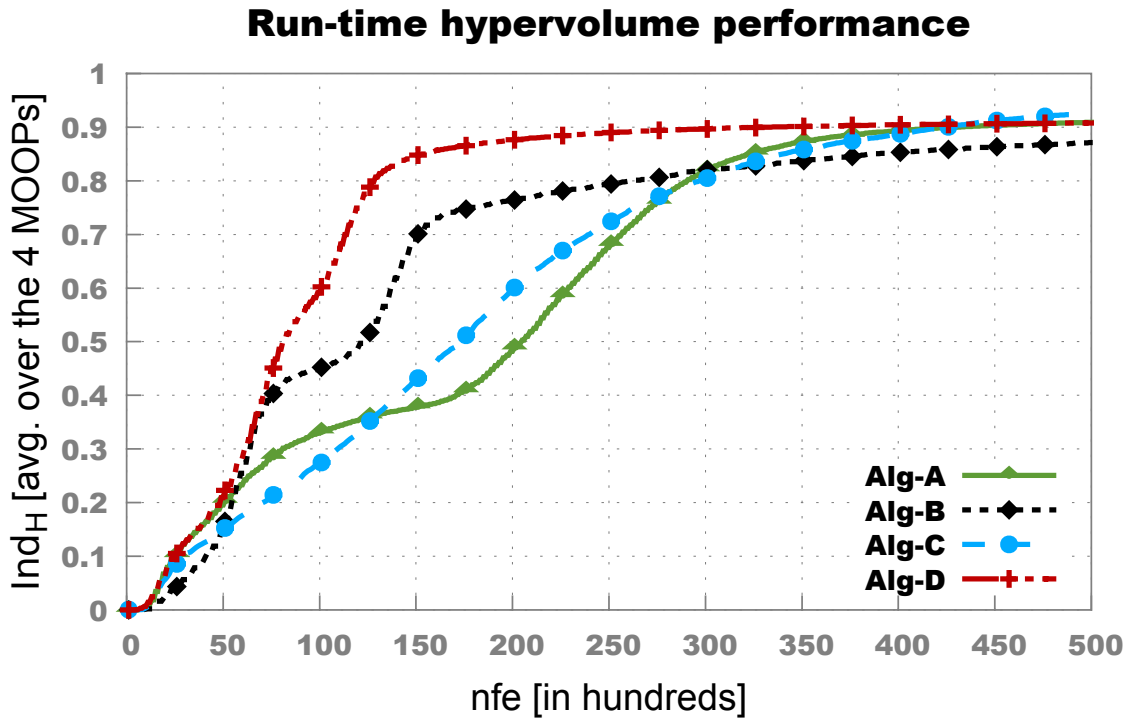


Figure 3.7: Averaged run-time Ind_H-measured performance over the entire toy benchmark set

analysis more than one primary *PN* quality indicator. Furthermore, towards the end of the runs, the case-by-case and simple averaging perspectives are not always very meaningful. This is because , on several MOOPs, by this stage, most good solvers find *PN*s of roughly similar quality and, when relying only on plotted values, it is hard to tell apart when the small differences in PN quality matter (e.g., the performance of the four algorithms on P2 after 35000 fitness evaluations) and when they don't (e.g., Alg-B, Alg-C and Alg-D on P4 after 35000 fitness evaluations). In order to make such distinctions, a second analysis based on statistical significance testing is required leading to a much lengthier and complicated performance analysis process.

Our **application-motivated main idea aimed at simplifying the comparison process** is centered on interpreting the run-time hypervolume performance plots as results of a **multi-stage race between competing MOEAs**. The main goal of this race is to achieve full convergence (e.g., $Ind_H \approx 1$) after the lowest possible *nfe*. The secondary goals are to have the highest Ind_H values at the end of each stage in the race. Consequently, it makes sense to envision a simplistic ranking strategy that, at the end of each stage, assigns ranks to the analyzed algorithms in ascending order of their Ind_H values: firstly, the worst performer is assigned the highest rank, then the second worse performer is assigned the second highest rank, etc. Special bonus or penalty ranks can also be assigned to excellent or very bad performers.

For example, let us consider eleven equidistant comparison stages for our toy scenario.

The first one (numbered with 0) takes place before the actual start of the optimization runs and is intended to evaluate the random initializations (initial populations) of the four MOOAs. The next ten comparison stages are placed 5000 fitness evaluations apart. In Figure 3.6, each of these 11 stages of the race is marked with a vertical solid line. With regard to the ranking schema, we adopt a basic strategy that, at each stage, assigns the rank 4 to the MOOA with the smallest Ind_H value, the rank 1 to the MOOA with the highest value and appropriate in-between values for the other two algorithms. We also adopt a **bonus / penalty system** that:

- assigns the rank 0 to a MOOA if the latter is able to achieve a Ind_H value higher than 0.99 (i.e., "fully converge"). This means we reward with a special rank value the algorithms that are able to discover Pareto approximations that cover more than 99% of the objective space covered by PF_{true};
- assigns the rank 5 (i.e., total number of evaluated algorithms plus one) to the MOOA that, at a certain stage, has not yet produced a relevant Pareto approximation that covers more than 1% of the objective space covered by PF_{true}. This means that rank 5 is assigned only if $Ind_H(PF_c) < 0.01$ where PF_c is the Pareto front of the best-current *PN* of the MOOA.

The ranks we obtained after applying the aforementioned schema are presented in Table 3.2. For each analyzed algorithm, the table also contains four important average ranks:

- μ_P - the average rank achieved by the MOOA on an individual MOOP. The closer the value of this average rank is to 0, the better the MOOA performs on the considered problem. Hence, μ_P can be used to rapidly / automatically identify those problems on which a MOOA performs very well (small μ_P values like those of Alg-B on P4 and Alg-D on P4) or very poorly (μ_P values close to the number of algorithms that are analyzed like those of Alg-A on P4 and Alg-B on P3).
- μ_S - the average rank across the entire problem set at a given stage (i.e., after a fixed *nfe*). These ranks are especially useful as we shall later combine them in order to display the dynamics of the relative MOOA performance over time.
- μ_F - the average rank achieved by an algorithm across the entire problem set in the final stage (i.e., at the end of the experiment). The MOOA that displays the smallest value of μ_F was able to "fully converge" or discover higher quality Pareto approximations on more problems than any of its competitors. In our case Alg-C and Alg-D share the best μ_F values.
- μ_A - the overall average rank achieved by the MOOA during the comparison (i.e., the average of all individual ranks). The value of μ_A can be used to single out the MOOAs that tend to generally outperform their counterparts. In our case, μ_A indicates Alg-D as the best overall performer.

In the left-side plot from Figure 3.8 we used the μ_S values to plot the *hypervolume-Ranked Performance Curves* (**HRPC**s) associated with our (toy) comparison scenario. The

Table 3.2: Ranks corresponding to the run-time Ind_H plots presented in Figure 3.6. For each algorithm, the highlighted values are used to create the left-side plot from Figure 3.8.

	Rank computation stages based on Ind_H values											
Pb.	0	1	2	3	4	5	6	7	8	9	10	μ_P
	Ranks achieved by Alg-A											
P1	5	5	5	5	4	4	3	3	3	2	2	3.73
P2	5	1	1	1	1	1	1	1	1	1	0	1.27
P3	5	2	1	1	1	1	1	1	1	2	2	1.64
P4	5	5	5	5	4	4	4	4	4	4	4	4.36
μ_S	5.00	3.25	3.00	3.00	2.50	2.50	2.25	2.25	2.25	2.25	2.00	
	$\mu_A = 2.75$, $\mu_F = 2.00$											
	Ranks achieved by Alg-B											
P1	5	5	5	2	2	2	2	2	2	3	3	3.00
P2	5	4	4	4	4	4	4	4	4	4	4	4.09
P3	5	5	4	4	4	4	4	4	4	4	4	4.18
P4	5	1	0	0	0	0	0	0	0	0	0	0.55
μ_S	5.00	3.75	3.25	2.50	2.50	2.50	2.50	2.50	2.50	2.75	2.75	
	$\mu_A = 2.95$, $\mu_F = 2.75$											
	Ranks achieved by Alg-C											
P1	5	1	1	3	3	3	4	4	4	4	4	3.27
P2	5	5	3	3	3	3	3	2	2	2	2	2.82
P3	5	5	3	3	3	3	3	3	3	1	1	3.00
P4	5	5	3	3	3	3	3	0	0	0	0	2.27
μ_S	5.00	3.50	2.50	3.00	3.00	3.00	3.25	2.25	2.25	1.75	1.75	
	$\mu_A = 2.84$, $\mu_F = 1.75$											
	Ranks achieved by Alg-D											
P1	5	5	2	1	1	1	1	1	1	1	1	1.82
P2	5	2	2	2	2	2	2	3	3	3	3	2.64
P3	5	1	2	2	2	2	2	2	2	3	3	2.36
P4	5	2	0	0	0	0	0	0	0	0	0	0.64
μ_S	5.00	2.5	1.5	1.25	1.25	1.25	1.25	1.5	1.5	1.75	1.75	
	$\mu_A = 1.86$, $\mu_F = 1.75$											

plots confirm the general result hinted by the μ_A and μ_F ranks that Alg-D is by far the best performer among the four MOOAs for the entire duration of the optimization. However, this observation somehow contradicts our earlier choice (made after consulting Figure 3.7) of preferring Alg-C in the late stages of the run. This is due to the fact that the **basic ranking schema** we have previously introduced favors the MOOA that is able to perform

very well on the highest number of MOOPs and totally ignores the magnitude of the existing differences. As such, if we would have only two algorithms competing over two MOOPs and one algorithm would constantly outmatch its counterpart on the first problem by 10% and would constantly be outperformed on the second problem by 0.5%, the basic schema would rank them as perfect equals with $\mu_A = 1.5$ and $\mu_F = 1.5$ for both (we also assume that no penalty or bonus ranks are assigned). Obviously this strategy is not always preferred, but one of the main advantages of our racing-based comparison is that small adjustments of the ranking schema are able to easily outline desired performance characteristics.

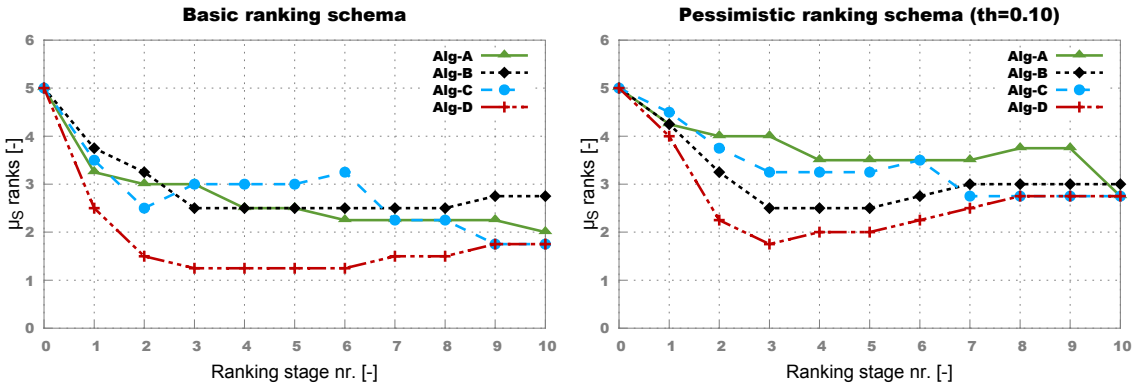


Figure 3.8: HRPCs obtained when applying the racing-based ranking methodology on the toy benchmark set

For example, in our toy scenario, we can focus the MOOA comparison on analyzing if there are large differences in performance between the tested algorithms by imposing a ranking threshold that only allows rank improvements if the difference between consecutive hypervolumes is higher than $th\%$. According to this modification, even when maintaining the aforementioned bonus / penalty system, we obtain a quite **pessimistic ranking schema** that, for $th = 0.1$ would rank five MOOAs that have Ind_H values of 0.64, 0.78, 0.84, 0.985 and 0.995 with 4, 3, 3, 1 and 0. The HRPCs obtained when applying the pessimistic ranking schema with $th = 0.1$ are presented in the right-side plot from Figure 3.8 and they highlight that at the end of the experiment there is at least one MOOP on which Alg-B performs much worse than its competitors (final objective space coverage smaller by at least 10%). In contrast, the differences between Alg-C and Alg-D near the end of the runs don't appear to exceed this threshold value for any of the four test problems. By applying the pessimistic ranking with different values of the threshold th, one is expected to get an even better overview of the comparative performance of the studied MOOAs.

Considering a fixed value as the ranking threshold, is not the only option. Since the ranks are computed on indicator values that are averaged over several independent runs, it would make a lot of sense to also impose a **statistical ranking schema** that only allows rank improvements if the differences between consecutive values are statistically significant – given a one-sided Mann-Whitney-Wilcoxon test [104] with a considered significance level of 0.025, for example. Throughout the thesis, whenever using this statistical ranking schema, we maintained the bonus / penalty ranking criteria.

In is very important to understand that our racing-based evaluation methodology is primarily aimed to facilitate a fast *comparative performance analysis* of multiple algorithms across multiple problems. In these cases, when using HRPCs with well crafted comparison schemata, one can quickly gain insight what algorithm (if any) performs better or worse than the average of its tested counterparts. Nevertheless, because of the design (and intended purpose) of the new ranking methodology, when using it, one should be careful how to interpret the results. Consider for instance the comparison (e.g., μ_F results) of three MOOAs: Alg-X, Alg-Y and Alg-Z over a set of 10 MOOPs. Let us assume that Alg-X has a convergence behavior that is generally very similar but always slightly better [e.g., $0.01 < [Ind_H(\text{Alg-X}) - Ind_H(\text{Alg-Y})] < 0.02$] than that of Alg-B across all test problems. Alg-Z has a different convergence behavior: it performs well on 5 problems where Alg-X and Alg-Y do not and it performs worse than both Alg-X and Alg-Y on the other 5 problems. In all cases $|Ind_H(\text{Alg-Z}) - Ind_H(\text{Alg-X})| = 0.05$. Assuming a basic ranking schema with no bonuses and penalties, Alg-X ($\mu_F = 1.5$) is ranked much better than Alg-Z ($\mu_F = 2.0$) because Alg-X is assigned a rank of 2 (as it is still better than Alg-Y) on the five problems where it performs poorly while Alg-Z is assigned rank of 3 on the five problems where it performs poorly. In other words, the presence of a mediocre algorithm that has a strongly biased convergence behavior (i.e., Alg-Y) impacts the entire comparison. The removal of Alg-Y from the comparison, would result in Alg-X and Alg-Z being ranked as equals (which is in fact the truth). In light of these considerations, throughout this thesis:

- we shall compare more than two MOOAs via HRPC plots only in order to get a rough idea of the average comparative performance of the algorithms (across large problem sets);
- we shall use HRPC plots of only two algorithms in order to display a more focused and accurate image of the comparative performance of the analyzed MOOAs across large problem sets.

All in all, we strongly believe that the new numerical (i.e., μ_P, μ_F, μ_A) and graphical (i.e., HRPCs based on different ranking schemata) tools / methods presented in this section can be extremely useful for MOO scientists and practitioners as they enable them to rapidly evaluate the general comparative performance of their MOOAs (especially on benchmark sets that contain many MOOPs).

Naturally, the general racing-based evaluation methodology we envisioned can be modified / improved to work with different base indicators like (3.8) or (3.5), different ranking schemata and different bonus / penalty criteria. The concrete version we have chosen to describe is adapted to the particularities of our considered CIMOOPs and our stated research goal.

3.3.3 Using HRPCs to tune MOEA/D

Whenever performing comparative tests between NSGA-II, SPEA2, GDE3, DEMO and other algorithms that implement a $(\mu + \lambda)$ selection for survival mechanism centered on Pareto-based elitism, using the same population (and or archive) size across al MOEAs and

all MOOPs (one wishes to evaluate them on) is considered quite fair. For example, it is quite common to report results obtained with the above described MOEAs when using a fixed population size of 50, 100 or 200 individuals despite the fact that the population size is one of the key settings in most EAs and carefully tuning this parameter can yield significant result improvements.

In the case of decomposition-based approaches like MOEA/D, choosing a good population size [that would also result in a fair comparison with $(\mu + \lambda)$ MOEAs] is a bit more complicated. This is because the population size determines the number of single-objective sub-problems the original MOOP is decomposed into and thus critically influences the search behavior of MOEA/D. For example, in [77], the authors recommend running MOEA/D with a population size of 300 for MOOPs with 2 objectives and with a population of 595 for MOOPs with 3 objectives. However, when adjusting the population size based on the definition of the problem, one is creating an unfair advantage when comparing to algorithms that run with a fixed population on all MOOPs since the latter approaches should also be (at least roughly) tuned in order to best match population sizes and MOOP specifications.

Since MOEA/D is considered by many in the field as one of the best a posteriori MOO methods, we wanted to **tune MOEA/D to deliver a very good general performance when using a fixed population size for all the 25 artificial (benchmark) MOOPs** mentioned in Section 3.1.1. More precisely, we considered the MOEA/D-DE version and, while setting most of the parameters as suggested in [78], we wanted to check what fixed population size (i.e., number of weighting vectors) would deliver the best convergence behavior when setting a limit of 50000 fitness evaluations. The goal was to obtain a black-box MOEA (MOEA/D-DE version) able to deliver a solid performance on the entire benchmark set without any parameterization. The reasons for wanting to create such a "robust" version of MOEA/D-DE (or of any other MOEA for that matter) are discussed at length in Chapter 6.

In order to determine the best population choice, we used the newly proposed race-based MOEA evaluation methodology to estimate the relative performance achieved by MOEA/D-DE (on the 25-MOOP benchmark set) when using different population sizes: 100, $200, \ldots, 900$. The HRPCs obtained when applying the basic ranking schema (with ranking stages after every 1000 fitness evaluations) are found in the top left plot of Figure 3.9. The HRPCs obtained when applying a $th = 0.05$ pessimistic ranking schema[5] are shown in the top right plot of the same figure. These two sets of HRPCs indicate that the results obtained with populations of 100, 200 and 300 individuals are sensibly worse than those obtained with larger population sizes for $nfe \geq 5000$. Also, when using population sizes ≥ 600, the convergence speed in the initial phase of the runs (i.e., until ranking stage 10) is negatively impacted.

Considering our application domain, where (at present) MOEAs are usually allowed to perform at most 10000-15000 fitness evaluations per run, we would prefer a MOEA/D-DE configuration that does not clearly underperform in the initial (and key-interest) parts of the run. Therefore, we chose to inspect the performance achieved when using populations

[5]That requires a minimum Ind_H-measured *PN* improvement of at least 5% in order to apply a rank improvement.

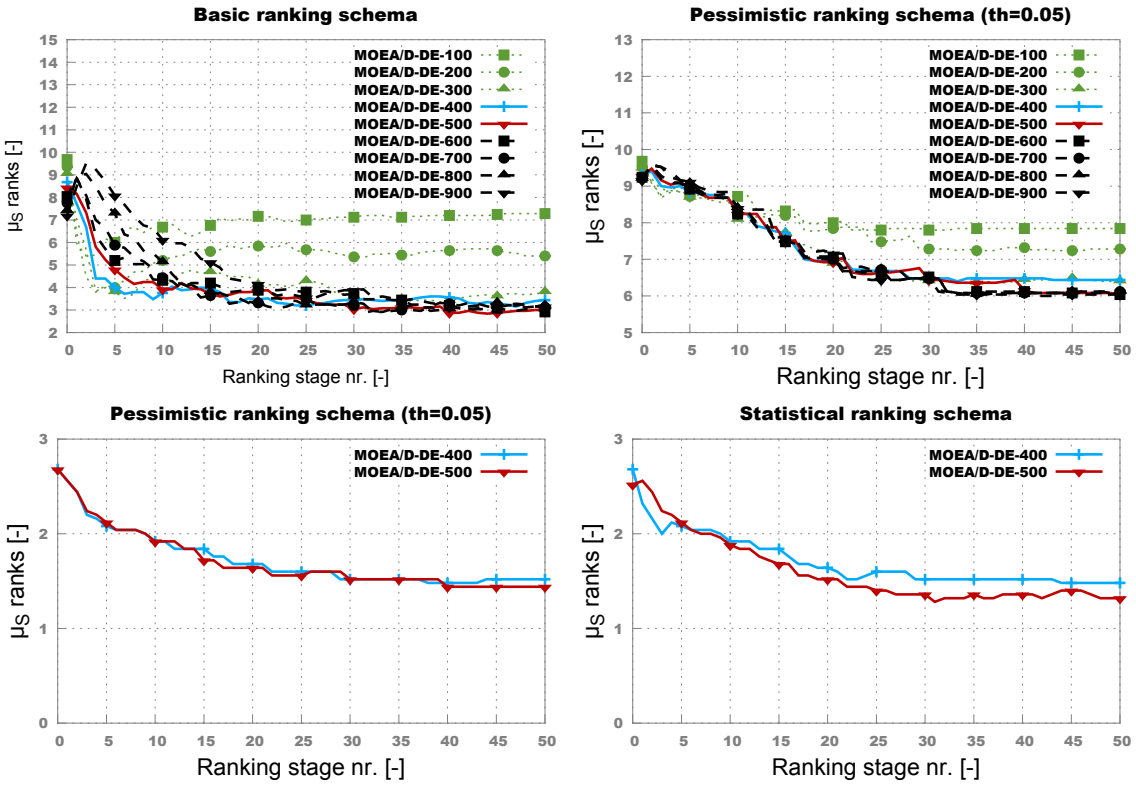


Figure 3.9: HRPCs obtained when testing the impact of applying MOEA/D-DE with various population sizes.

of 400 and 500 individuals more closely by constructing HRPCs only for these two cases. The results are presented in the bottom plots of Figure 3.9 an both the $th = 0.05$ pessimistic ranking schema and the statistical ranking schema indicate that using a fixed population size of 500 is the better choice for MOEA/D-DE.

In Figure 3.10 we present an HRPC-based analysis of the comparative performance of the literature recommended settings for MOEA/D-DE (i.e., population size of 300 for bi-objective MOOPs and 595 for problems with three objectives) and the previously determined best-fixed-population setting. The HRPCs show that the fixed-population version performs slightly worse only for $nfe \leq 5000$. Therefore, we shall generally use this (fairer) version of MOEA/D-DE with a fixed population of 500 for most of the comparative tests performed throughout Chapter 6.

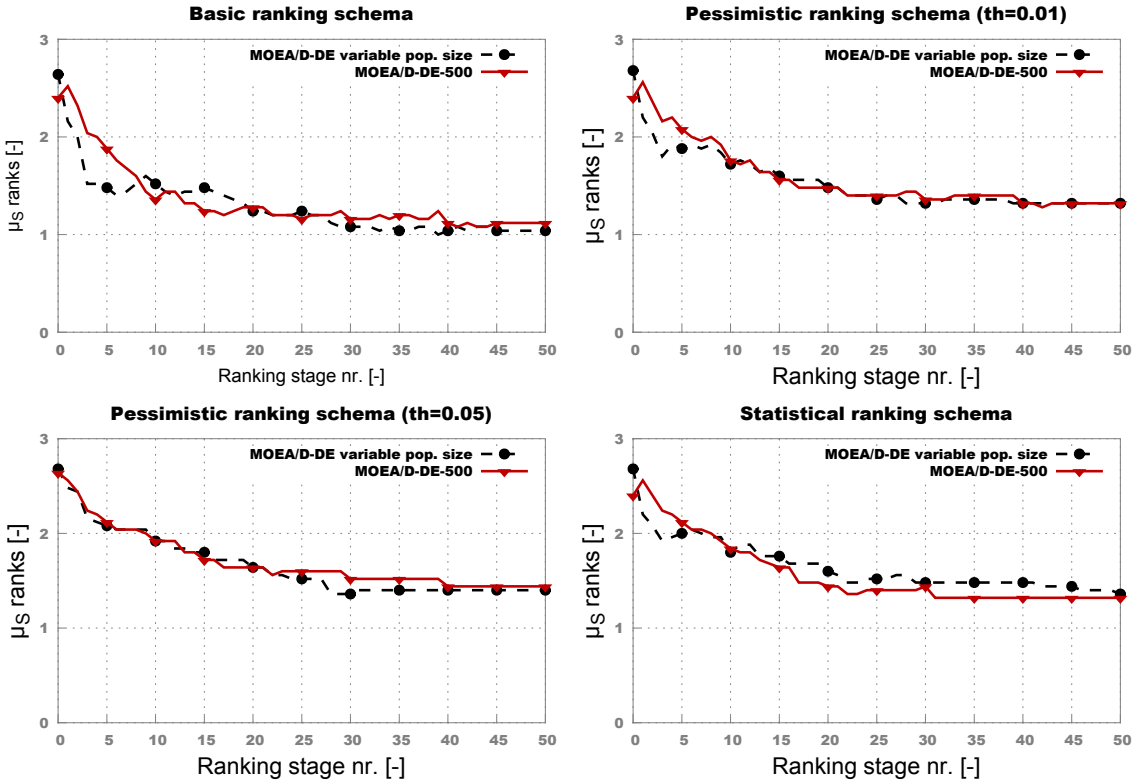


Figure 3.10: HRPCs obtained when comparing the MOEA/D-DE version with problem-depended population size and the MOEA/D-DE version with a fixed-to-500 population size.

Chapter 4

On-the-Fly Surrogate Modeling

4.1 General Idea

4.1.1 Approach and Performance Baseline

One of the best known approaches for improving the general run time of optimization runs and analytic frameworks that must use a computationally-intensive performance evaluation procedure is to construct metamodels / **surrogate models** that **can approximate the results of using the original quality assessment method at a fraction of the computational cost** [105] [106] . Other well documented studies on surrogate based analysis and optimization can also be found in [107] and [108].

With regard to our concrete MOO application domain, of particular importance is the recent (2013) PhD Thesis of Martin Pilát [109] that exclusively focuses on improving the performance of MOEAs via surrogate modeling. Pilát argues that there are at least two ways to augment an EA with surrogate models:

- *pre-selection* - in this case the surrogate model is used to pre-screen individuals in the population and the individuals that seem promising will be evaluated using the real (computationally -intensive) fitness function;

- *local search* - in this case the surrogate model is used as support (i.e., fitness estimator) by a local search strategy (e.g., gradient descent) that aims to improve the best current solution(s) in the population. Pilát mentions that the **generational approach to surrogate modeling** (in which **the trained surrogate model is used to replace the original computationally-intensive fitness estimation function for several generations**) can be regarded as a special case of local search.

Depending on how the surrogates are integrated within the MOOA one can distinguish between:

- *standard regression models* - that try to provide a numerical approximation of the results that would be obtained when applying the original fitness evaluation method;

- *aggregated surrogate models (ASM)* introduced in [110] - that directly estimate (without computing individual objective values) if a new individual lies in the sub-part of the objective space that is not Pareto-dominated by the current *PN* of the MOOA. Given this behavior, ASMs are primarily coupled with pre-selection based strategies.

Regression surrogates are by far the most popular approaches, but the newly introduced ASMs have produced impressive results in [111] and [112] (mainly on artificial problems) and are likely to gain more acceptance in the near future.

Within the large category of regression-intended surrogates, one can also differentiate between:

- *global models* - that aim to accurately estimate optimization objectives and constraints when considering vast sections of (or even the entire) variable space;

- *local models* - that aim to roughly estimate results in the limited area of variable space that contains (is centered on) the current *PN* of the MOOA. Local models are usually applied to create memetic (evolutionary) algorithms, as shown in [113].

Considering the nature of the CIMOOPs we generally aim to solve, **we have chosen to systematically explore a generational application of global surrogate models**. The main motivation for this is that:

- although fairly simple, the approach does have the potential to notably reduce the overall duration of our MOEA-based optimizations;

- in the long run, by obtaining accurate global approximation models for every objective and constraint proposed by our CIMOOPs, we will be able to (re-)construct fast-to-evaluate artificial MOOPs that display domain realistic fitness landscapes. These realistic MOOPs can then be used to perform valuable experiments that are not possible given the duration of real-life CIMOOPs optimization runs.

- investigating how to obtain good global surrogate regression models is of general (practical) interest for the mechatronics community as these global regression models can also be (re)used in other stages of the electrical drive design process like tolerance / sensitivity analysis [114].

Therefore, our plan is to substitute the computationally-intensive performance evaluation process described in Figure 3.2 with very-fast-to-evaluate surrogate design performance evaluators based on high-quality regression models. These surrogate models would act as direct mappings between the design parameters [e.g., $\mathbf{x} \in D^n$ where $\mathbf{x}$ is given in (3.1)] and the electrical drive target parameters which should be estimated [e.g., $o_1(\mathbf{x}) = \eta_{ed}(\mathbf{x})$, $o_2(\mathbf{x}) = T_{cogPP}(\mathbf{x})$ and various types of result-related constraints]. One **important requirement of our application domain / environment** is that, in order to be effective in their role of reducing the overall duration of the optimization runs, **the surrogate models need to be constructed on-the-fly, automatically, during the run of the MOOA**. The reason for this is that the required surrogate models are fairly specific for each considered CIMOOPs and

target parameter and they generally cannot be reused across different problems. Furthermore, the surrogate creation process should not require any human interaction in order to achieve (rapid) acceptance from the DMs involved in the electrical drive design process.

Since we are using regression models, it is obvious that their creation process will require some sort of training data. This training data is usually structured as a set of individual samples / records and each such sample must contain both the initial electrical motor design parameter values [as, for example, listed in (3.1)] and the corresponding target (objective and constraint) values computed using the process outlined in Figure 3.2. The "on-the-fly" restriction severely restricts the potential sources of such data samples to:

- applying a multi-dimensional sampling strategy to create individuals based on the concrete specifications of the given CIMOOP's n-dimensional domain;
- running the MOOA for a limited number of generations.

Our decision is to favor the second option since we believe that the samples (designs / individuals) that are the byproduct of the MOOA have a better chance of covering the "interesting sections" of the search space (i.e., those parts of D^n that encompass PS). Nevertheless, whenever applying evolutionary MOO approaches on CIMOOPs we also apply a multi-dimensional sampling technique, namely the Latin hypecube sampling (LHS) [115], to create the initial population of the MOEA.

Since the surrogate-based evaluation of a design is virtually instantaneous (when comparing to the FE-based one), the idea is to switch the MOOA to a surrogate-based fitness function that would enable it to solve CIMOOPs (i.e., explore the search space) much faster. However, because of some practical considerations, the resulting surrogate-enhanced MOO algorithm is slightly more complicated and we present it in more detail in the next section.

The performance baseline we generally aim to improve upon consists in a rather conventional application of a generational MOEA on CIMOOPs that are similar to those presented in Section 3.1.2. Throughout this chapter, we shall refer to this straightforward MOEA-based electrical drive design approach as **ConvOpt**. Concretely, the ConvOpt experiments were performed with a generational version of NSGA-II that used a population of size 50 and, a crossover probability of 0.9 and a standard parameterization of the SBX and PM genetic operators (please refer back to Section 2.3.3 for more details). During each optimization run, the algorithm was allowed to evolve 100 generations. The new surrogate-enhanced optimization approach also uses NSGA-II as its support MOEA[1] and was termed **HybridOpt**. The optimization runs we used for all the tests described in this chapter were performed on the first three CIMOOPs described in Section 3.1.2, viz. IndMOOP1, IndMOOP2 and IndMOOP3.

Before proceeding with the more in depth description of HybridOpt, we must mention that, even though the proposed surrogate methodology is fairly general and thus suitable to different a posteriori MOO approaches, for the remainder of this chapter we shall adopt a terminology that is in line with the general scope of this thesis – i.e, "MOEA" instead of

[1] Any MOEA could have been used. Our choice was influenced by the fact that we had a lot of valuable historic ConvOpt results obtained with NSGA-II and we wanted to make the surrogate-based approach as comparable as possible to the baseline.

"MOOA", "fitness evaluation" instead of "performance evaluation", "individuals" instead of "design variable vectors", etc.

4.1.2 Overview of the Surrogate-enhanced MOEA Framework

Figure 4.1 contains a comparative sketch of the conventional MOEA-based optimization process (ConvOpt) and of the new surrogate-enhanced version (HybridOpt) when wanting to run an optimization for $tGen \in \mathbb{N}$ generations. This sketch also puts into perspective the six major functional parts of our hybrid a posteriori MOO approach:

1. The *FE-based MOEA execution stage* – an initial and preferably low number of generations $feGen \in \mathbb{N}$, with $feGen < tGen$, is evolved by the support MOEA using the fitness function based on FE simulations. All the individuals computed in these $feGen$ generations will form the (training set) samples used to create the surrogate models.

2. The *surrogate model construction stage* – appropriate surrogate regression models are trained for all the considered target parameters according to a strategy that tries to balance model training time and model prediction quality.

3. The *surrogate-based MOEA execution stage* – using the newly trained surrogates to guide the search, the MOEA evolves at least[2] $sGen = tGen - feGen$ generations. Since the surrogate-based fitness function is much faster than its FE-based counterpart, enabling the predication of all the elicited target values within milliseconds, we also tried to gain a rapid improvement of final *PF* quality by:

   (a) increasing the total number of generations the MOEA computes during the run – i.e., choosing *sGen* such that $sGen > tGen - feGen$;

   (b) increasing the size of the population with which the MOEA operates during the surrogate-based execution stage;

4. The *surrogate-based PN computation stage* – a preliminary *PN* is extracted exclusively from the set of individuals evaluated using the surrogate models without taking into consideration the FE-evaluated individuals from the first $feGen$ generations. The idea behind this approach is to make the surrogate-only *PN* less prone to instabilities induced by highly likely surrogate prediction bias.

5. The *FE-based re-evaluation stage* – all the individuals from the surrogate-only *PN* are re-evaluated using the time consuming process that relies on FE simulation. The motivation for this is twofold:

   (a) During the fitness evaluation process presented in Figure 3.2, the check for geometric errors is done in the *"2D/3D model construction"* phase. This check

[2] In order to obtain a final *PN* that is based on the same total *nfe* as ConvOpt.

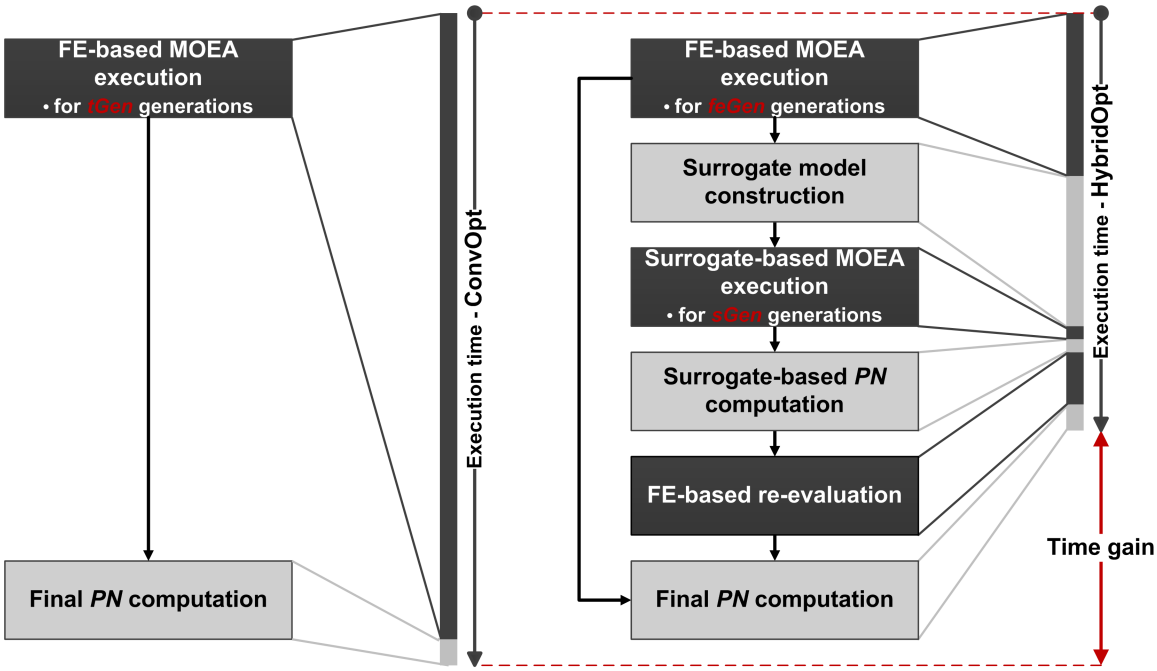


Figure 4.1: Diagram of a conventional optimization process (ConvOpt) and of the surrogate-enhanced hybrid optimization process (HybridOpt)

is done inherently inside a third party computer aided design (CAD) application and helps to remove most of the formalization constraints (described at the end of Section 2.1.2) by triggering the fail of the entire fitness evaluation process for design vectors that contain infeasible geometric combinations (e.g., magnet width larger than the rotor width). Since this modeling phase is also very time consuming and its main product – the 2D/3D meshes – is not used when performing a surrogate-based evaluation, it makes a lot of sense to also remove this phase of the fitness evaluation process when using surrogate models. This means that there is a high chance that some of the individuals represented in the surrogate-only *PN* are in fact infeasible (geometrically invalid).

(b) Regardless of the perceived quality level, any surrogate (approximation) model is bound to display some sort of prediction error when compared to the FE simulation results it aims to emulate. However, at the end of the optimization, the DM expects all the designs presented as Pareto optimal to display the same approximation error (i.e., the internal estimation error of the FE simulation software FEMAG™).

6. The *final PN computation stage* - the final *PN* of the optimization run is extracted from the combined set of all FE-evaluated designs, i.e., individuals from the first $feGen$ generations and FE-re-evaluated surrogate-based individuals.

It is important to note that the proposed HybridOpt process redefines the purpose of conducting FE simulations during the optimization run. As such, these very accurate

but extremely time intensive operations are no longer used for exclusively guiding the evolutionary optimization process. Instead, FE simulations are employed in the first and final phases of the run. In the beginning of the optimization process, they are used to initially steer the search process down the right path and to construct the training set necessary for constructing the surrogate models. In the final stage of the optimization, FE simulations are used for analyzing only those individuals (designs) that seem the most promising (i.e., are Pareto-optimal) when applying a surrogate-based fitness evaluation. As such, during the middle and especially final parts of the optimization process, when improvements generally come at a much higher computational cost, the surrogate models are used to guide the MOEA towards the interesting regions of the design space.

4.2 Constructing On-the-Fly Global Surrogate Models

4.2.1 Design Decisions

When considering the HybridOpt flow chart from Figure 4.1, there are at least three important general decisions that must be made when wishing to construct on-the-fly global surrogate models for fitness approximation:

1. What is better: to generate one single surrogate model that is able to predict all the elicited targets or to model each target parameter independently?
2. Which are the most appropriate regression techniques for creating surrogate models?
3. What is the best value of $feGen$ – the number of initial generations that must be computed using the FE-based fitness evaluation function?

In our case, the first design decision is also the simplest. Firstly, because (regression) methods that are inherently able to approximate multiple targets using only one model are notably scarce[3]. Secondly, and most importantly, because the *target parameters of our CIMOOPs are usually highly conflicting* (i.e., price vs. performance vs. reliability) and *exhibit different degrees of non-linearity*. Therefore, it is quite reasonable to create a **dedicated surrogate model for each separate target** – as also described in [116], for instance – and to envision a **two-tier modeling approach**:

- In the first stage we rapidly construct surrogate estimators based on simple linear regression models for all the elicited targets of the CIMOOP.
- In the second stage, for the targets on which the linear estimators do not display a good prediction performance, we construct more advanced surrogate models using non-linear modeling techniques.

This modeling approach also cuts into the second design decision and thus focuses it on determining which non-linear (regression) modeling technique is better suited for creating

[3]Neural network architectures that feature multiple output nodes are the most well known exponents.

our surrogates. There is a wide range of methods to choose from and in Section 4.2.5 we report off-line surrogate modeling results obtained with techniques like: feed-forward *artificial neural networks* (**ANN**s) [117], *support vector regression* (**SVR**) [118] and *radial basis function networks* (**RBFN**s) [119]. The pre-selection of these three methods is motivated by the fact that they are widely regarded [120] [121] as very powerful and general modeling tools. In order to gain more insight into the nature of the three studied CIMOOPs, we also performed tests with a regression orientated adaptation of the instance based (i.e., *k*-nearest neighbor) learning algorithm **IBk** [122] - a method that is generally not held in such high regard as the first three options.

All the off-line results from Section 4.2.5 clearly indicate that a specific type of ANNs known as *multi-layer perceptrons* (**MLP**s) generally display the best training time and quality performance when applied on the non-linear surrogate modeling targets. In light of this, we have adopted MLPs as the main non-linear modeling method to be used in HybridOpt. In Sections 4.2.3 and 4.2.4 we provide details regarding MLPs and how they can be used to generate very useful global surrogate approximation models.

The third design-related question is very important because the value of $feGen$ is directly correlated to the size of the data set used to train the surrogates. If the training set is too small, or not well distributed over the search space, one is likely to obtain surrogate models that do not generalize well. Conversely, if the training set is too large, the training time of the surrogate model is likely to increase considerably and one would perform far more FE simulations than necessary. More details and several possible answers can be found in Sections 4.2.6 and 4.4.1.

4.2.2 Data Sets and Target Parameters

The data sets we used for experimenting with surrogate model construction are byproducts of (legacy) ConvOpt MOO full runs on the electrical drive design problems IndMOOP1, IndMOOP2 and IndMOOP3. For each MOOP, data sets were obtained by aggregating every feasible individual (i.e., non-error design parameter vector) that was generated (evolved) together with its associated FE-computed target values. In particular, in Sections 4.2.5 and 4.2.6 we present our off-line surrogate modeling results on the three data sets that are the result of the best ConvOpt runs for each considered industrial MOOP.

Since each ConOpt run evolved a population of size 50 over 100 generations, we evaluated 5000 individuals (electrical drive designs) during each optimization. Because some of these designs were labeled as infeasible by the performance evaluation process, the actual size of the data sets when only considering the designs evolved in the first $xGen$ generations is smaller than $xGen \times 50$. The number of feasible individuals generated during the selected ConvOpt runs after 25, 33, 50, 75 and 100 generations is presented in Table 4.1.

As previously mentioned, for each CIMOOP the individual objectives and any external result-related constraints constitute the targets of the surrogate modeling process. In order to simplify referencing in forthcoming descriptions, all the eleven targets contained in the three considered CIMOOPs have been labeled with T1, T2, ... , T11. The first two columns of Table 4.3 contain the mapping between these target labels and the concrete electrical drive design objectives and constraints defined in Section 3.1.2.

Table 4.1: The number of feasible designs generated during the best ConvOpt runs.

Problem	Feasible designs after generation				
	25	33	50	75	100
IndMOOP1	1214	1595	2412	3599	4743
IndMOOP2	850	1197	1912	2831	3713
IndMOOP3	806	1103	1725	2586	3533

The three specific CIMOOPs were selected because the results of several ConvOpt MOO runs on them were available and because the DMs we closely collaborated with throughout our research endeavors considered that, together, these problems offer a good overall image of the standard degrees of difficulty one should expect from most electrical drive design MOOPs.

4.2.3 Structure and Training of MLP Surrogate Models

ANNs have been among the popular options for constructing surrogate models since more than a decade, as shown by the approaches in [123] and [124]. This comes as no surprise since ANNs are some of the most well-known and well-studied machine learning / prediction / statistical learning algorithms. They are generally used to estimate or approximate functions (of usually unknown form) that depend on several inputs. Their popularity is largely owned to the fact that they are proven to posesss the universal approximation capability [125] and they are known to perform well on non-linear and noisy data [126].

As briefly mentioned before, the particular type of ANN we experimented with is called a multi-layer perceptron (MLP). The MLP paradigm, although quite simplistic, is widely used and, over the years, has been successfully employed to create high quality prediction models in many applications fields as demonstrated in [127] and [128].

Like with many other statistical learning methods, the application of a MLP predictor as a surrogate model is straightforward. This is because, by design, the input of a MLP is a *data sample* (i.e., a variable vector) and its output comes in the form of one or more scalar values – the associated prediction(s). Throughout this section we shall mark an arbitrary data sample that is to be used as input by the MLP by $\mathbf{x}$. N.B. This notation has been chosen deliberately since throughout the present work $\mathbf{x} \in D^n$ denotes (I) an electrical drive design parameter vector and (II) an individual evolved during the run of a MOEA that is a potential solution candidate of the CIMOOP to be solved.

Generally, the MLP architecture contains one layer of input nodes (i.e., processing units), one layer of outputs units and one or more intermediate (hidden) layers. Please see Figure 4.2 for reference. MLPs use a feed-forward information flow principle that passes the input data from the units in the input layer to the unit(s) in the output layer indirectly, via the units from the hidden layer(s). Any connection between two nodes of the MLP, n_i and n_j, has an associated weight w_{ij} that represents the strength of that respective connection. Connections are not permitted between nodes from the same layer of the network.

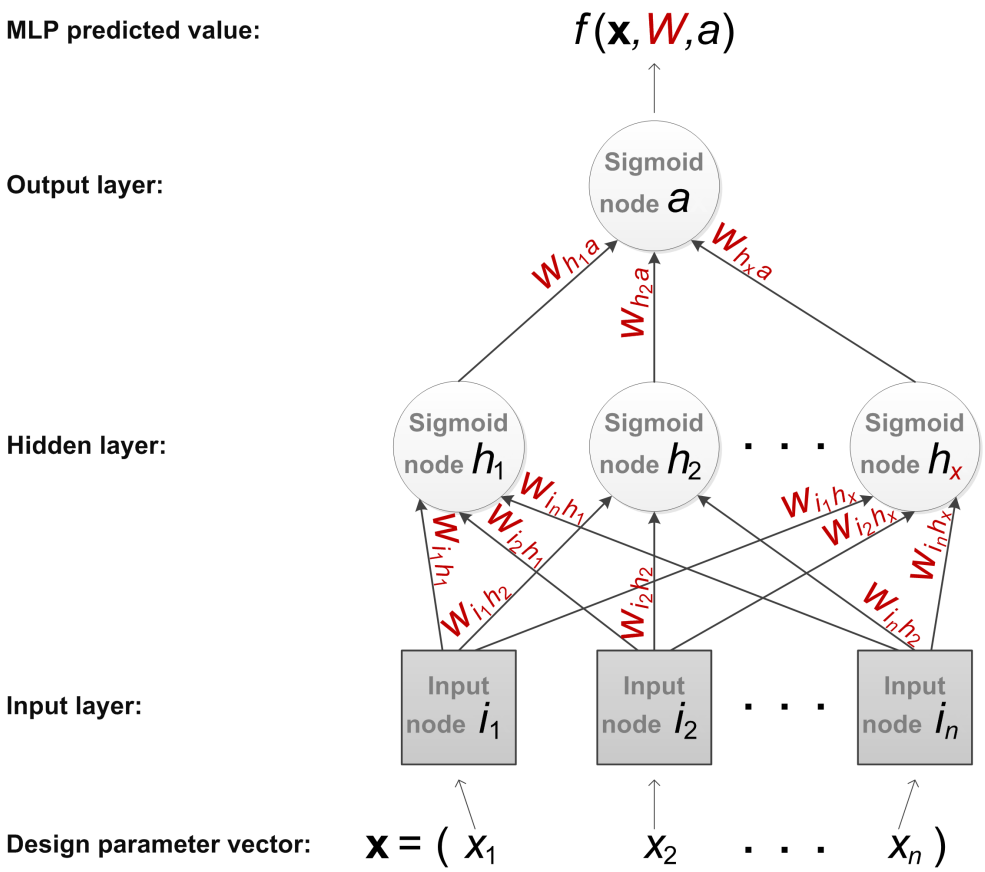


Figure 4.2: MLP model with one hidden layer and one output unit

A concrete MLP prediction model is fully defined by its specific architecture (number of hidden layers, number of nodes in each layer, number of output nodes, graph of connections between nodes) and by all the values of the associated connection weights. Usually, architecture related choices are made either:

- subjectively - as they depend on the experience and preferences of the person who carries out the MLP-based modeling.
- automatically - being the result of a (grid, stochastic, directed, undirected, etc) search between several possible options.

With regard to the associated connection weights, there are also several methods of discovering appropriate values and we shall proceed to describe the (standard) approach that we use in the case of our surrogate models. First and foremost, though, we must provide a little more information regarding the inner-workings of the MLP's nodes.

Given the node n_i, the set $Pred(n_i)$ contains all the nodes n_j that connect to node n_i – i.e., all the nodes n_j in the MLP for which w_{ji} exists. Similarly, the set $Succ(n_i)$ contains all the nodes n_k to which node n_i connects to – i.e., for which w_{ik} exists.

Each node n_i of the MLP is responsible for computing an output value based on a set of specific inputs. Given the fact that the MLPs we use are fully connected[4], it is obvious that, apart from the input data sample (i.e., $\mathbf{x}$), the output value of several nodes also depends on W – the set which reunites all the connection weights. In light of this, in order to explicitly state these dependencies on weights and input, we note the function that computes the output value of an arbitrary MLP node n_i with $f(\mathbf{x}, W, n_i)$. Based on the concrete definition of this output function, a MLP contains two types of nodes:

1. *Input nodes* - all the nodes in the input layer. The role of these nodes is to simply propagate into the network the appropriate values from the input data sample. In the case of our surrogate models, we always use as many input nodes as design variables in the data sample and thus $f(\mathbf{x}, W, n_i) = x_i$ for all $i \in \{1, \ldots, n\}$, where x_i is the i^{th} variable of $\mathbf{x} \in D^n$.

2. *Sigmoid nodes* - all the other nodes in the network. These nodes perform a two-step operation. Firstly, they compute a weighted sum of all the connections flowing into them (4.1). Secondly, the output is calculated via a non-linear logistic (sigmoid shaped) *activation function* (4.2):

$$\mathfrak{S}(\mathbf{x}, W, n_i) = \sum_{n_j \in Pred(n_i),\, w_{ji} \in W} f(\mathbf{x}, W, n_j) w_{ji} \tag{4.1}$$

$$f(\mathbf{x}, W, n_i) = \frac{1}{1 + e^{-\mathfrak{S}(\mathbf{x}, W, n_i)}} \tag{4.2}$$

Since we model each target parameter independently, the output layers of our MLP surrogate models only contain one node. We marked this node with a since its output value – i.e., $f(\mathbf{x}, W, a)$ – is also the final approximation result of the entire MLP model (i.e., the predicted regression value of the surrogate). Computing $f(\mathbf{x}, W, a)$ for a given data sample $\mathbf{x} \in D^n$ is a simple and very fast feed-forward process that starts by requiring all the input nodes of the MLP to activate – i.e., compute their simplistic output function. Afterwards, the feed-forward process continues in a stepped manner that requires each sigmoid node in the MLP to compute its output value using (4.1) and (4.2) as soon as all its inputs become available.

Having a better understanding of the importance of the connections weights (i.e., W), it is time to describe how to find appropriate values for them. Typically, **all the connection weights of the MLP** are initialized with small random values and then they **are subsequently adjusted through a training process**. The main idea of the training process is to adjust W such as to improve the accuracy of the predictions made by the MLP.

Given the general MOOP definition from (2.1), let us train a surrogate model for the target parameter $o_1(\mathbf{x})$, $\mathbf{x} \in D^n$. This training process requires the usage of a **(training) data set** (not: $T \in D^n$). The main requirement we impose on T is that for every data sample $\mathbf{x} \in T$, we also know the value of $o_1(\mathbf{x})$. This is because the prediction accuracy of a MLP can be

[4]For every hidden or output node n_i, $Pred(n_i)$ contains all the nodes in the previous layer of the network.

estimated by passing every $\mathbf{x} \in T$ through the network and then computing a cumulative error metric over the entire training set[5] (i.e., we consider a batch learning approach). Given previous notation conventions and the *squared-error loss function*, a convenient form of the cumulative error metric over T is:

$$E(W) = \frac{1}{2} \sum_{\mathbf{x} \in T} [o_1(\mathbf{x}) - f(\mathbf{x}, W, a)]^2 \tag{4.3}$$

The standard approach in the field of ANNs for adjusting W based on $E(W)$ is the backpropagation method proposed by Paul J. Werbos in his 1974 PhD thesis [129]. In essence, backpropagation is a gradient-based iterative (optimization) method. At each iteration t, the first step is to compute the error metric $E_t(W)$ over the training data set T using (4.3). Afterwards, each weight $w_{ij}(t) \in W$ in the MLP will be updated[6] according to the following formulae:

$$\begin{aligned} \Delta w_{ij}(t) &= -\eta \, \delta(n_j) f(\mathbf{x}, W, n_i) \\ w_{ij}(t) &= \begin{cases} w_{ij}(t) + \Delta w_{ij}(t) \text{ , if } t = 1 \\ w_{ij}(t) + \Delta w_{ij}(t) + \alpha \Delta w_{ij}(t-1) \text{ , if } t > 1 \end{cases} \end{aligned} \tag{4.4}$$

where $\eta \in (0,1]$ is a constant called the *learning rate* and with $\alpha \in [0,1)$ we mark the control parameter of the empirical backpropagation enhancement known as *momentum*. Using an appropriate momentum setting can help the gradient-based method to converge faster and to avoid some local minima. The function $\delta(n_j)$ denotes the *gradient* and shows the cumulative impact that the weighted inputs flowing into node n_j have on $E_t(W)$. $\delta(n_j)$ is computed differently depending if n_j is an output or hidden node:

$$\delta(n_j) = \begin{cases} \sum\limits_{\mathbf{x} \in T} \{ f\left(\mathbf{x}, W, n_j\right) \left[1 - f\left(\mathbf{x}, W, n_j\right)\right] \left[f\left(\mathbf{x}, W, n_j\right) - o_1\left(\mathbf{x}\right) \right] \} & \text{, if } n_j = a \\ \sum\limits_{\mathbf{x} \in T} \{ f\left(\mathbf{x}, W, n_j\right) \left[1 - f\left(\mathbf{x}, W, n_j\right)\right] \sum\limits_{n_k \in Succ(n_j), w_{jk} \in W} w_{jk} \delta(n_k) \} & \text{, otherwise} \end{cases} \tag{4.5}$$

The hoped-for result of the training process is that the resulting MLP model (architecture + weight values) can approximate accurately the value of $o_1(\mathbf{x}), \forall \mathbf{x} \in D^n$. This means that for a new data sample $\mathbf{y} \in D^n$ that is not necessarily part of T, $f(\mathbf{y}, W, a) \approx o_1(\mathbf{y})$.

Like most iterative methods, the backpropagation-based training supports several stopping criteria. The most popular options are imposing limitations on t – the number of performed iterations, imposing limitations on the total computation time of the training process and setting a minimum error threshold for $E_t(W)$. In the case of our surrogate modeling tasks, we have chosen to adopt an early stopping mechanism that terminates the execution whenever the prediction error computed over a validation subset V does

[5] A lower value of the error metric indicates a higher prediction accuracy.

[6] The update procedure starts with the weights of the connections that lead into the output node and then continues backwards (hence the name) with the weights of the connections that lead into the previous hidden layer etc.

not improve over 200 consecutive iterations. This validation subset is constructed at the beginning of the training process by randomly sampling (with removal) 20% of the instances from the training set T. Adopting this validation-based stopping criterion is a well known practice in the field of machine learning / modeling and various empirical results suggest that it may have a benefit in helping to prevent the overfitting[7] of the trained models [130].

4.2.4 An Automatic Selection Procedure for MLP Surrogate Models

As mentioned in Section 4.1, decisions pertaining to the concrete architecture of each MLP model must be made automatically.

In order to reduce the search space of possible MLP architectures, we firstly refer to two important theoretical results. In [131] the authors show that MLPs with two hidden layers can approximate any arbitrary function with arbitrary accuracy and in [132], George Cybenko proves that any bounded continuous function can be approximated by a MLP with a single hidden layer and a finite number of hidden sigmoid nodes. Secondly, although we have not performed a detailed mathematical analysis of the (interactions between all the) physical phenomena that govern the targets parameters involved in the CIMOOPs we deal with, all the modeling experiments we have carried out[8] provide strong empirical evidence that our general modeling tasks do not require the usage of MLPs with two hidden layers. N.B. This choice (restriction) does not alter the generality of the model selection procedure we describe in this section.

Given the general MLP architecture and the previously mentioned limitations (i.e., one output node, a single hidden layer, full connections between neighboring layers), the most important model-pertinent decision remaining is to determine how many nodes to place in the hidden layer of the MLP. Of course, this choice also affects the learning rate (η) and momentum (α) values that must be selected in order to obtain a successful training process. In practice, finding the right parameter combination boils down to experimentation, usually combined with some sort of expert knowledge.

In the case of our surrogates, in order to automatically **determine the number of hidden units and appropriate values for η and α**, we first construct a fairly large set of models for different parameter value combinations obtained via a **best-parameter grid search** (please see Section 4.2.5 for exact settings). Secondly, we apply a **non-standard selection strategy** aimed at finding a **high-quality robust surrogate model**. Such a surrogate model should simultaneously display:

- *an appropriate structural complexity* – for many machine learning / prediction / statistical learning paradigms (including MLPs), a rather high structural complexity of the trained model is often positively correlated with unwanted effects like overfitting

[7] Simplistically, a regression model exhibits overfitting if it does not generalize well, – i.e., it does not display the expected prediction accuracy when applied to (unseen) data samples different than the ones in its training set. For a more accurate description of this and other statistical machine learning principles, like validation sets and approximation bias, please see the excellent introduction and overview by Hastie, Tibshirani and Friedman [130]

[8] On the five problems presented in Section 3.1.2 as well as on several others.

and longer training and evaluation times. In our case, a MLP model is considered to be less complex than another if the former has fewer units in the hidden layer. Ties are broken in favor of the model that requires less computation time to train.

- *a stable predictive behavior* – the ability to generate *Pareto-consistent predictions* across the entire domain of its intended inputs.

In order to clearly describe what we understand by the syntagm "**Pareto-consistent predictions**" let us consider a very simple toy MOOP with two objectives [marked as $o_1(\mathbf{x})$ and $o_2(\mathbf{x})$] and four surrogate models (2 for each objective): SM1, SM2, SM3 and SM4. In Table 4.2 we present ten data samples (variable vectors) – marked by $\mathbf{x^a}, \mathbf{x^b}, \ldots, \mathbf{x^i}$ – and their associated true (e.g., FE-computed) objective values and surrogate-estimated objective values.

Table 4.2: Examples of true and surrogate-approximated target values for ten variable vectors corresponding to a toy MOOP. Highlighted values mark perfect surrogate approximations

Sample	Values for objective o_1			Values for objective o_2		
	true	SM1	SM2	true	SM3	SM4
$\mathbf{x^a}$	1.0	15.0	2.0	3.0	7.0	3.1
$\mathbf{x^b}$	2.0	15.2	2.0	2.0	6.0	3.1
$\mathbf{x^c}$	3.0	15.7	3.0	1.0	2.0	3.2
$\mathbf{x^d}$	4.0	15.8	4.0	11.0	24.0	11.0
$\mathbf{x^e}$	5.0	16.1	5.0	12.0	27.0	12.0
$\mathbf{x^f}$	6.0	20.0	6.0	13.0	30.0	13.0
$\mathbf{x^g}$	7.0	21.0	7.0	14.0	32.0	14.0
$\mathbf{x^h}$	8.0	21.5	1.7	15.0	33.0	3.0
$\mathbf{x^i}$	9.0	22.3	1.8	16.0	37.0	2.9
$\mathbf{x^j}$	10.0	23.0	1.9	17.0	41.0	2.8

After skimming the tabular data it is easy to observe that surrogate models SM1 and SM3 have a general tendency (bias) to produce approximation values that are much higher than their corresponding true target values. Surrogate models SM2 and SM4 seem to be far better predictors and for 6 data samples in the case of SM2 and 4 samples in the case of SM4 they produce perfect approximations of the corresponding true target values. These empiric observations are easily confirmed by any error metric that relies on a squared-error loss function like the popular *mean squared error* (MSE) or the cumulative error metric (4.3) we defined earlier. For instance, when applying the latter we obtain the cumulative error values 29.28 for SM1, 8.89 for SM2, 34.50 for SM3 and 16.18 for SM4. In light of all this evidence, one might be tempted to select SM2 and SM4 as the most appropriate surrogate pair for the toy MOOP. This in fact is the worst possible choice since if one would want to extract a Pareto non-dominated set from $\mathbf{x^a}, \mathbf{x^b}, \ldots, \mathbf{x^i}$,:

- based on the true objective values, one would choose $PN^{true} = \{\mathbf{x^a}, \mathbf{x^b}, \mathbf{x^c}\}$;

- based on the SM2 and SM4 surrogate approximated objective values, one would choose $PN^{(SM2,SM4)} = \{\mathbf{x^h}, \mathbf{x^i}, \mathbf{x^j}\}$;

When considering the real Pareto ranking of the ten data samples, $PN^{(SM2,SM4)}$ is not just a bad choice, it is the worst possible one. This is because it labels as Pareto non-dominated the three variable vectors that based on their real / true objective values are in fact the three worst performers (i.e., most Pareto-dominated) of the entire sample set. Hence, even though SM2 and SM4 seem to be far more *accurate* than their counterparts, the predictions of SM2 and SM4 are anything but Pareto consistent as they *do not preserve the (real) Pareto relation*.

One of the most important observations based on the above-described scenario is that **a squared-error loss function can be expected to provide a somewhat misleading estimation of surrogate approximation quality (usefulness) inside a Pareto comparison context**. Furthermore, when considering a generational applications of (global) surrogate modeling (like the one in HybridOpt), making larger approximation errors that share the same overall trend (i.e., are well correlated) with the true target values is far more important than making smaller approximation errors that minimize a square-loss function but are not trend-consistent and finally alter the Pareto ordering (and the MOEA search behavior during the surrogate-based stage of the run). For instance, when applying a goodness-of-fit measure that is bias invariant, like the well-known *Pearson's coefficient of determination* (not: R^2), to the same surrogate approximations from Table 4.2, we obtain the values 0.95 for SM1, 0.01 for SM2, 0.99 for SM3 and 0.29 for SM4. Since values of R^2 closer to 1 indicate a stronger positive correlation, values closer to -1 indicate a stronger negative correlation and values closer to 0 indicate no correlation, based on these new R^2 results, one would clearly choose SM1 and SM3 as the best surrogate pair for the given problem. The resulting $PN^{(SM1,SM3)} = \{\mathbf{x^a}, \mathbf{x^b}, \mathbf{x^c}\}$ is a perfect match of PN^{true}, thus providing a good example why **a correlation-based approach is to be preferred when estimating the quality (usefulness) of global surrogate models that are used inside Pareto comparison contexts**.

Given these considerations, our choice to estimate surrogate prediction quality via a Pareto-consistent metric like R^2 should become obvious, especially when taking into account the importance of selecting and re-evaluating good designs during the *surrogate-based PN computation* and the *FE-based re-evaluation* stages of HybridOpt. More precisely, in order to estimate the quality of a MLP surrogate model, we employ a well-known 10-fold cross-validation data partitioning strategy [130] and we compute the R^2 values obtained by the surrogate over each of the individual 10 folds (when the respective fold is used for testing during the cross-validation process). The quality estimate assigned to the surrogate model at the end of the cross-validation procedure is given by:

$$q_m = \mu(R^2) - \sigma(R^2) \tag{4.6}$$

We use observed standard deviation of R^2 over the 10 cross-validation folds as a penalty in (4.6) because we want to favor models that, on average, exhibit a stable predictive behavior across the entire sample space (i.e., domain of possible inputs). The reasoning behind this is that a significant value of $\sigma(R^2)$ indicates that the surrogate model is not producing equally good (Pareto-consistent) predictions across all regions of the sample space. In our case, the

existence of such locally-biased surrogate models is fairly probable given the origins of their training data sets – i.e., byproducts of highly elitist evolutionary processes that disregard unfit individuals and thus avoid exploring *apparently uninteresting regions* of the (design parameter) sample space.

It is of paramount importance to emphasize the fact that although our choice of the final surrogate models is based on R^2-estimated prediction quality (usefulness) rather than MSE-estimated prediction accuracy, all the surrogate models are trained by actively trying to maximize accuracy (as we described in detail for MLPs in the previous section). This means that extreme cases like the one illustrated in Table 4.2 are highly unlikely. Therefore, all the surrogate models generated during the grid searches that display very good (high) values of R^2 (and especially q_m) also generally display some of the best (lowest) MSE results. Nevertheless, since the overall goal is to find the surrogates that are the most useful for speeding-up the MOEA-based search, we prefer to reference the high "**quality of surrogate predictions**" with the understanding that we primarily address the **good Pareto-consistency** of these predictions but also keep in mind their extremely likely **high accuracy**.

Having explained how the application domain (i.e., building surrogates over MOEA-generated data sets) influences our preferences regarding the interplay between model quality (Pareto-based comparison usefulness), model accuracy and model (structural) complexity, we can now provide the details of our proposed automatic selection procedure for global MLP surrogate models. In the first step, all the surrogate models obtained during the best-parameter grid search are ranked according to their quality, as estimated by (4.6). Afterwards, a *quality threshold* is computed as the mean estimated quality of the best performing $p_q\%$ of all surrogate models[9]. The finally selected surrogate is the least structurally complex model that has an estimated q_m value higher than the quality threshold. Figure 4.3 contains a graphical description of the selection procedure that should confirm the gut feeling one might have by now that we indeed treat the surrogate selection process as a MOOP in its own right: the (conflicting) aims are the quality and the structural simplicity we demand from the generated surrogates. Furthermore, the selection strategy we have previously described can be seen as an adaption of the ε-constraint principle from (2.16) as we minimize structural complexity subject to a quality-based (ε) threshold.

The general idea of our proposed model selection procedure (i.e., balancing estimated model quality and complexity) also bears similarities to a model selection strategy for regression trees proposed by Breiman et al. in [133]. The major difference is that we use a pessimistic (deviation adjusted) correlation measure and we compute the selection threshold using a broader model basis in order to avoid the cases where the quality standard would be set according to a single highly complex (and likely overfitted) model that only displays a marginally higher q_m value.

For the comparative results of the newly proposed model selection procedure vs. the much simpler (and standard) option of just choosing the model that displays the best 10-fold cross-validation R^2 performance please see the results reported in Section 4.4.2.

It is noteworthy to mention that our automatic model selection procedure can easily be

[9]For the tests reported in Sections 4.2.6 and 4.3.2 we used the setting $p_q = 2$ and in Section 4.4.1 we also experiment with the setting $p_q = 5$.

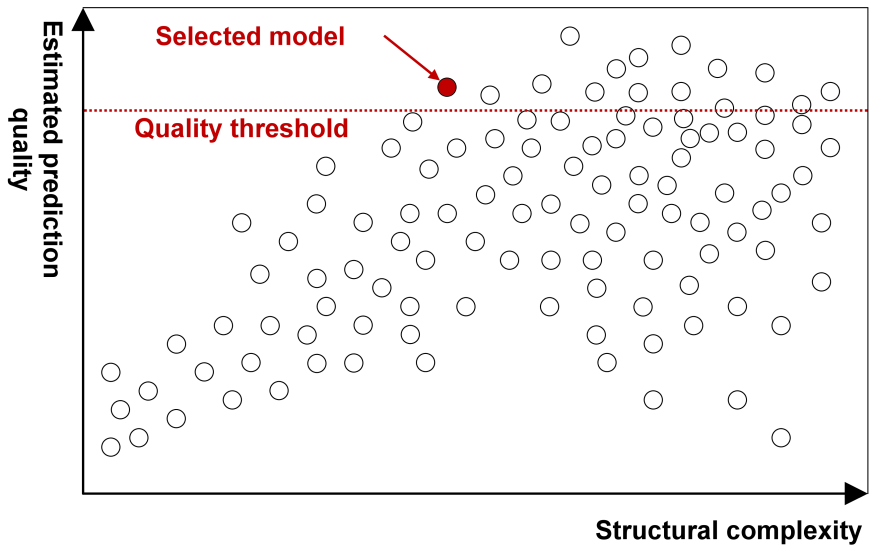


Figure 4.3: An illustration of the proposed surrogate model selection strategy when considering the predictors obtained after the best-parameter grid search.

adapted when opting for another surrogate modeling method. The only requirement is to select another (method-dependent) indicator (or set of indicators) for measuring complexity. For example, when opting for SVR-based surrogates, one might consider the C parameter and/or the number of required support vectors as suitable complexity indicators.

4.2.5 Comparative Performance of MLP Surrogate Models

In order to assess the predictive performance of the four non-linear modeling methods we have pre-selected as viable options for constructing global surrogate models, we performed off-line regression experiments with the three (ConvOpt-generated) data sets presented in Section 4.2.2 and the eleven target parameters they contain in total.

Given our general strategy of enhancing MOOAs using on-the-fly global surrogate prediction models, we considered that the most appropriate challenge was to:

- partition each of the three data sets into training subsets (that contained individuals from the first 33 generations) and test subsets (that contained the individuals of the remaining 67 generations);
- construct surrogates with each modeling method using only data from the training subsets;
- estimate the predictive performance of the "best" surrogate constructed (trained) with each modeling method by computing its R^2 over the associated test subsets.

The sizes of all training and test subsets can be inferred from Table 4.1. For each non-linear modeling method, the "best" performing model for each target was determined by applying a best-parameter grid search and by choosing the highest-quality model based on

average R^2 performance during 10-fold cross validation. We adopted this standard model selection procedure instead of the one previously proposed in Section 4.2.4 because we do not want the general comparison to be influenced by the fact that our MLP-tailored selection procedure could not be (or was not) fully adapted to the particularities of the other non-linear modeling techniques and would thus be perceived as hindering their real performance.

In the case of the MLP surrogates, the best-parameter grid search was conducted as follows:

- the number of hidden units was varied between 2 and $2\times n$ (where n is the number of design variables in the modeled CIMOOP) – i.e., $n = 6$ for IndMOOP1, $n = 7$ for IndMOOP2 and $n = 10$ for IndMOOP13);
- the learning rate (η) was varied between 0.05 and 0.40 with a step size of 0.05;
- the momentum parameter (α) was varied between 0.0 and 0.7 with a step size of 0.1.

This means that we constructed 704 surrogate models for each target parameter proposed by IndMOOP1, 832 models for each target parameter proposed by IndMOOP2 and 1216 models for each target parameter of IndMOOP13. During the backpropagation-based training processes we employed the validation-based termination criterion described at the end of Section 4.2.3. Whenever the early stopping criterion was not reached, the training process was stopped after 5000 iterations.

In the case of the SVR-based surrogates, we created 675 surrogate models for each target parameter as, during the grid search, we varied:

- the values of the general complexity parameter (C) between $\{2^{-4}, 2^{-3}, \ldots, 2^{9}, 2^{10}\}$;
- the values of the (width) controlling parameter of the RBF kernel (γ) between $\{2^{-5}, 2^{-4}, \ldots, 2^{2}, 2^{3}\}$;
- ε from the ε-intensive loss function between $\{0.001, 0.005, 0.01, 0.025, 0.05\}$.

During the grid searches conducted for RBFNs we constructed 918 models for each target parameter as we varied the number of clusters between $\{2, 3, 4, 5, 10, 20, \ldots, 500\}$ and the allowed minimum standard deviation inside the cluster between $\{0.25, 0.5, 1, 2, \ldots, 15\}$.

When training the IBk-based surrogates, we created 900 surrogates during each best-parameter grid search. We varied the number of nearest neighbors between 1 and 300 (with a stept size of 1) and we also experimented with three different distance weighting options: weight by 1 / distance, weight by 1 - distance and no weighting.

In order to offer some insight into the "degree of difficulty" associated with each modeling experiment, we also report the results obtained with surrogates based on linear regression models (also trained using the individuals from the first 33 generations). The MLP, SVR and RBFN implementations we used are largely based on the ones provided by the WEKA open source machine learning platform [134].

The obtained surrogate (regression) modeling results are presented in Table 4.3. The performance of the linear regression models indicate that there are several target parameters

for which non-linear modeling is required. Given the motivations from the previous section regarding the importance of Pareto-consistent predictions and several empirical observations regarding our CIMOOPs, after the first stage of our two-tier surrogate modeling strategy, we mark a target parameter as non-linear if a linear regression model is not able to achieve a training R^2 value of at least 0.95. The three CIMOOP-based data sets contain a total of six non-linear targets: T1, T2, T4, T5, T10 and T11.

Table 4.3: Linear and non-linear surrogate (regression) modeling results on data sets that resulted from ConvOpt runs on three CIMOOPs. The best results for each individual target or average over individual targets are highlighted.

CIMOOP obj./constr.	Target	Classif.	R^2 on test data				
			Linear	MLP	SVR	RBFN	IBk
IndMOOP1-$o_1(\mathbf{x})$	T1	non-lin	0.7353	0.9864	0.9330	0.9029	0.8744
IndMOOP1-$o_2(\mathbf{x})$	T2	non-lin	0.6048	0.9530	0.9540	0.8992	0.9040
IndMOOP1-$o_3(\mathbf{x})$	T3	linear	0.9777	0.9992	0.9997	0.9999	0.9660
IndMOOP1-$o_4(\mathbf{x})$	T4	non-lin	0.6390	0.9674	0.9640	0.9099	0.9044
IndMOOP2-$o_1(\mathbf{x})$	T5	non-lin	0.8548	0.9923	0.9960	0.9859	0.9749
IndMOOP2-$o_2(\mathbf{x})$	T6	linear	0.9916	0.9997	0.9997	0.9998	0.9904
IndMOOP2-$c_1(\mathbf{x})$	T7	linear	0.9990	0.9999	0.9998	0.9999	0.9254
IndMOOP3-$o_1(\mathbf{x})$	T8	linear	0.9970	0.9999	0.9997	0.9999	0.9689
IndMOOP3-$o_2(\mathbf{x})$	T9	linear	0.9514	0.9998	0.9995	0.9999	0.8822
IndMOOP3-$o_3(\mathbf{x})$	T10	non-lin	0.8526	0.9799	0.9839	0.9804	0.8791
IndMOOP3-$o_4(\mathbf{x})$	T11	non-lin	0.8355	0.9564	0.9552	0.9521	0.8794
Average over the 5 linear targets			0.9830	0.9997	0.9997	0.9997	0.9466
Average over the 6 non-lin. targets			0.7540	0.9720	0.9640	0.9384	0.9026
Average over all the 11 targets			0.8580	0.9847	0.9802	0.9664	0.9226
Rank over the 6 non-lin. targets			5	1	2	3	4

Since IBk is not widely regarded as a very robust modeling method, the fact that it is not able to generally match the modeling performance of the other three methods does not come as a surprise. However, the data from Table 4.3 shows that IBk is generally able to outperform by a large margin the linear regression models when only considering the six targets labeled as non-linear (average R^2 of 0.9026 for IBk and 0.7540 for linear regression). The notable exceptions are targets T10 and T11, both belonging to IndMOOP3. One may conjecture that the reason for this is a mixture of three conditions.

1. IndMOOP3 seems to be a MOOP heavily affected by the formalization constraints

mentioned in Section 2.1.2 as Table 4.1 indicates that nearly one third of the designs generated by ConvOpt (i.e., NSGA-II) when trying to solve it were infeasible.

2. As indicated in Section 3.1.2, IndMOOP3 proposes only constrained optimization objectives, meaning that the evolutionary process converges slower (individuals that violate the constraints of the problem are ignored / quickly disregarded by ConvOpt and thus do not contribute at all to the evolutionary process).

3. The domain of the problem is larger ($n = 10$) than the ones of the other two considered CIMOOPs.

Furthermore, it makes a lot of sense to assume that the combined effect of all these conditions is that the data set corresponding to IndMOOP3 provides a much sparser representation of the sample space (i.e., CIMOOP domain) and that this is in fact the real reason for which a k-nearest neighbor method like IBk delivers a quite poor performance. An important observation regarding IndMOOP3 results is that MLPs and SVR appear to deal much better with sparser representations of CIMOOPs domains.

In fact, when considering only the six non-linear targets, our tests show that, **on average, MLPs and SVR produce the best non-linear surrogate models (with a slight advantage towards the MLPs models)**. RBFNs perform worse, especially on the target parameters proposed by IndMOOP1, viz. T1, T2 and T4. On these three targets, RBFNs has a performance that is quite similar to that of IBk.

In Section 3.1.2 we briefly mentioned that in order to significantly speed-up the run time of our optimization algorithms we try to parallelize as much as possible of the FE-based electrical drive performance evaluation process. Concretely, we use a high throughput computer cluster system managed using HTCondor™ [135]. Since best-parameter grid searches are a very important part of the surrogate construction process, we wanted to also obtain a basic estimate of the duration of these (usually computationally-intensive) operations for MLPs, SVR and RBFNs. As such, when distributing the grid searches over the computer cluster system, we measured the total wall-clock time required by each modeling method to perform all the six grid searches for the non-linear targets. When using at most 25 computation nodes:

- the MLPs searches required 178.79 minutes in total, resulting in an average training time of 0.67 minutes per model;
- the SVR searches required 220.86 minutes in total, resulting in an average training time of 1.09 minutes per model;
- the RBFNs searches required 224.40 minutes in total, resulting in an average training time of 0.81 minutes per model.

It should be mentioned that given the quite heterogeneous nature (i.e., highly diverse hardware performance) of the nodes (i.e., computers) in the cluster and the fact that we exercised no special control regarding the composition of the execution pool or the job scheduling protocol, the results synthesized above should only by interpreted as very

rough estimates of the true computational requirements associated with the grid-searches. Nevertheless, the obtained results do suggest that the **MLP-based modeling tends to be faster when compared with SVR or RBFN** (given the particularities of our CIMOOPs, software implementation and best-parameter grid searches defined for each method).

In light of the presented quality and time-wise results, after performing these tests, **we adopted MLPs as our default global surrogate modeling method** for non-linear targets.

4.2.6 The Stability over Time of MLP Surrogate Models

Having decided on modeling each target parameter independently and applying a two-tier approach that uses linear and MLP-based regression models, the only major surrogate-design decision left (from those outlined in Section 4.2.1) is to choose an appropriate value for $feGen$ – the parameter that denotes after how many FE-evaluated generations (i.e., or more generally, FE-evaluated individuals), we should start the on-the-fly surrogate creation process required by the HybridOpt architecture (proposed in Section 4.1.2).

As we wanted to select a good value for $feGen$, we performed several experiments aimed at quantifying the influence of this parameter on the quality and stability of the resulting global MLP-based surrogates. We only report results that concern MLPs as experiments have shown that the size and design space coverage of the training data sets (both of which are directly influenced by the value of $feGen$) do not have a very serious impact on the prediction quality of global linear regression models applied on linear targets[10].

For each of the considered 11 target parameters we constructed 50 different surrogate modeling scenarios. For scenario number i, the MLP surrogate was trained only using the individuals evolved in the first i generations (of the ConvOpt run). The corresponding approximation quality (i.e., R^2) was estimated using a separate test set that reunited all the individuals from the remaining (100-i) associated generations. For each modeling scenario, we applied our proposed on-the-fly MLP-construction methodology[11] in order to obtain a high-quality robust surrogate model.

In Figure 4.4 we plot the individual R^2 values that were obtained for each experiment. The charts show that, for all the 11 targets, the MLP-based surrogates display a *stable logarithmic saturation* behavior. Furthermore, when focusing on the 6 non-linear targets, viz. T1, T2, T4, T5, T10 and T11, the plotted values also suggest that there is a *promising search range* for $feGen$ between 20 and 30 as choosing a value from this interval ensures the creation of very high-quality surrogate models for all the considered targets.

In order to try and single out the most appropriate value for $feGen$, we conducted a second series of tests only with the surrogate models from the aforementioned promising search range. In these tests we wanted to inspect a little closer the stability of surrogate approximations and, for each surrogate model, we computed individual R^2 values over each (test) generation whose individuals were not part of the training process. For example, for

[10] For the linear modeling situations, a setting of $feGen = 10$ would be sufficient when considering all the CIMOOPs we have ever experimented with.

[11] This consists of the best-parameter grid search for MLPs presented in Section 4.2.5 and the automatic final model selection procedure described in Section 4.2.4

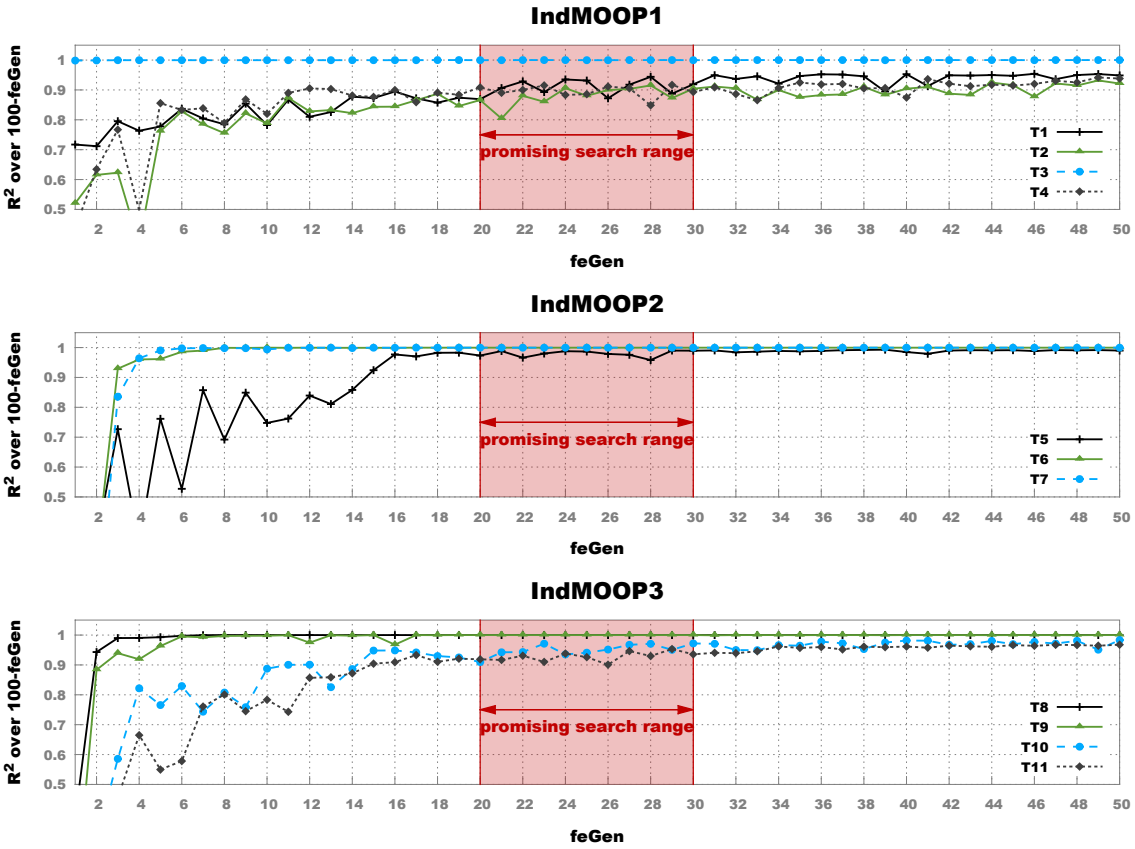


Figure 4.4: Evolution of the coefficient of determination computed over the remaining $100 - feGen$ generations for the best MLP surrogate models trained using the first $feGen \leq 50$ generations

surrogates trained with the setting $feGen = 22$ we computed 78 generational R^2 values, starting with the one for the individuals evolved in 23^{rd} generation of the original ConvOpt run and ending with the one for the individuals evolved in the 100^{th} generation of the ConvOpt run. Figure 4.5 contains the plots of the generational R^2 values obtained for T2 and T11, the non-linear targets that, according to the results from Table 4.3, are the most challenging to model.

Our final decision was to fix $feGen = 25$ since this was **the smallest value for which the surrogate models exhibited both a very high predictive quality and stability**. This conclusion was made after taking into consideration the generational R^2 values obtained for generations 31 to 100 over all six non-linear targets. Since we analyzed 11 surrogate models for each target (i.e., MLPs trained using the first 20 generations, the first 21 generations, ..., the first 30 generations) we obtained a total of $70 \times 6 \times 11 = 4620$ generational R^2 results. From the 420 generational R^2 associated with the six surrogate models constructed with the setting $feGen = 25$, 397 values (i.e., 94.52%) are higher than 0.9. From the 2100 generational R^2 values associated with surrogate models constructed with $feGen < 25$, only 892 values (42.48%) are higher than their matching counterparts constructed with $feGen =$

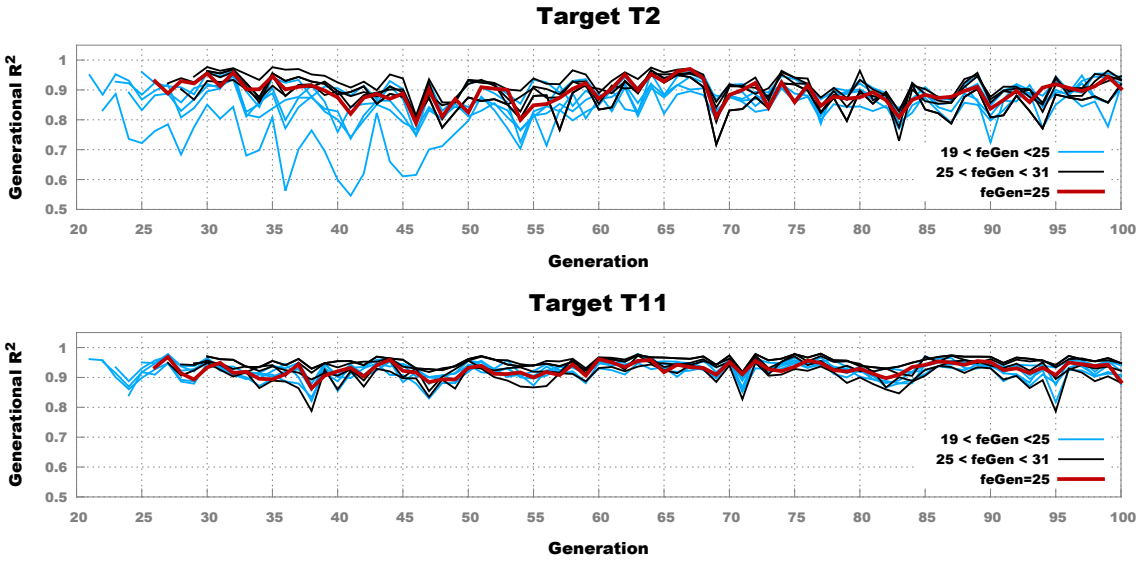


Figure 4.5: Evolution of the generational coefficient of determination for MLP surrogate models trained using the first 20 to 30 generations for the most difficult to model targets

25. From the 2100 generational R^2 values associated withe surrogate models constructed with $feGen > 25$, only 1207 values (57.48%) are higher than their matching counterparts constructed with $feGen = 25$. Lower settings of $feGen$ obtain fewer generational R^2 results higher than 0.9 and for $feGen > 25$ one does not notice much (or even any) improvement with regard to the three mentioned performance indicators (percentages).

Obviously, the choice $feGen = 25$ is fairly subjective, even though the procedure at the end of which it was made can be considered quite general. Nevertheless, having a rough estimate for the last open surrogate design question enabled us to perform (more interesting) on-line comparative tests.

4.3 MLP-enhanced NSGA-II

4.3.1 Algorithmic Description

In Algorithm 2 we present a more concrete description of HybridOpt – the surrogate-enhanced MOEA framework initially described in Section 4.1.2. We mention beforehand that, in order to simplify (and shorten) the presentation, apart from method calls and the normal assignment operator (i.e., "$\leftarrow$"), we also use a special binding operator denoted by "$\equiv$". The role of the latter is to mark a link between an arbitrary variable and a specific method with the implied meaning that all future references to the variable will be redirected to the associated method.

Algorithm 2 contains two methods. The main function is called **HYBRIDOPT**( ) and incorporates the base logic of our proposed approach. The secondary function is called **NSGA-II-SEARCH**( ) and is responsible for implementing the particular MOEA search

Algorithm 2 Surrogate-enhanced NSGA-II

```
1: function HYBRIDOPT(problem, popSize_FE, popSize_S, feGen, sGen)
2:     P ← INITIALIZEPOPULATION(popSize_FE, problem)
3:     fitnessEval ≡ FE-EVALUATOR(x, problem)
4:     ⟨P, Feasible_FE⟩ ← NSGA-II-SEARCH(feGen, popSize_FE, P, fitnessEval)
5:     Configurations ← INITIALIZEMLPGRIDSEARCH(problem)
6:     BestSurrogates ← ∅
7:     for all target ∈ problem do
8:         Models ← ∅
9:         for all c ∈ Configurations do
10:            Models ← Models ∪ TRAINMLPSURROGATE(c, target, Feasible_FE)
11:        end for
12:        BestSurrogates ← BestSurrogates ∪ SELECTBESTMODEL(Models)
13:    end for
14:    fitnessEval ≡ MLP-EVALUATOR(x, problem, BestSurrogates)
15:    ⟨P, Feasible_MLP⟩ ← NSGA-II-SEARCH(sGen, popSize_S, P, fitnessEval)
16:    OptimalSet_MLP ← EXTRACTPN(Feasible_MLP)
17:    fitnessEval ≡ FE-EVALUATOR(problem)
18:    OptimalSet_MLP ← EVALUATEFITNESS(OptimalSet_MLP, fitnessEval)
19:    return EXTRACTPN(Feasible_FE ∪ OptimalSet_MLP)
20: end function

21: function NSGA-II-SEARCH(xGen, popSize, InitialPopulation, fitnessEvaluator)
22:    t ← 0
23:    P_s ← InitialPopulation
24:    P_s ← EVALUATEFITNESS(P_s, fitnessEvaluator)
25:    FeasibleIndividuals ← ϕ
26:    while t < xGen do
27:        O_s ← CREATEOFFSPRING(P_s, popSize)
28:        O_s ← EVALUATEFITNESS(O_s, fitnessEvaluator)
29:        FeasibleIndividuals ← FeasibleIndividuals ∪ O_s
30:        P_s ← NONDOMINATEDSORTING(P_s ∪ O_s, popSize)
31:        t ← t + 1
32:    end while
33:    return ⟨P_s, FeasibleIndividuals⟩
34: end function
```

strategy we have chosen to experiment with. As the name suggests, we are using a (generational) version of NSGA-II that only contains minor (domain-required) modifications of the standard approach presented in Section 2.3.3. The most important of these modifications is that the return value consists is an ordered pair (2-tuple) that contains the last parent population (i.e, P_s) and a set of all the feasible individuals that have been generated during the search. Apart from this, the specific FE-based or MLP-based fitness assessment procedure

that must be applied during the evolutionary cycle is provided as an input parameter (not: $fitnessEvaluator$). The other three input parameters of **NSGA-II-SEARCH**( ) are:

- *xGen* – the number of generations to be computed;
- *popSize* – the size of the population with which the MOEA operates;
- *InitialPopulation* – a set containing the starting population of the evolutionary search process. N.B. If $popSize \neq |InitialPopulation|$ the necessary adjustments will be made automatically during successive executions of the **NONDOMINATEDSORTING**( ) function;

The main **HYBRIDOPT**( ) function contains the following input parameters:

- *problem* – the description of the CIMOOP to be solved including information regarding design parameters and optimization target;
- $popSize_{FE}$ – the size of the NSGA-II population during the FE-based part of the run;
- $popSize_S$ – the size of the NSGA-II population during the secondary surrogate-based part of the run;
- $feGen$ – the number of generations to be computed in the FE-based part of the run;
- *sGen* – the number of generations to be computed in the surrogate-based part of the run.

Inside Algorithm 2, **EVALUATEFITNESS**(*Set*, *evaluator*) is an auxiliary method of particular importance. Its inputs are a *Set* of unevaluated individuals (i.e., parameter vectors that encode electrical drive designs) and an *evaluator* that is bound to a concrete fitness assessment function. The main operations performed by **EVALUATEFITNESS**( ) are to apply the *evaluator* on every individual $\mathbf{x} \in Set$ and to return a filtered subset containing only the individuals from the original *Set* that have been flagged as feasible. Naturally, each individual in the returned set also contains values for all the target parameters of the *problem* to be solved. N.B. The knowledge regarding the *problem* resides inside the concrete (FE or surrogate-based) fitness assessment function that is bound to the *evaluator* input parameter.

Apart from **EVALUATEFITNESS**( ), there are nine more auxiliary methods that are referenced throughout Algorithm 2:

1. **INTITIALIZEPOPULATION**(*popSize*, *problem*) – this function initializes a total of *popSize* individuals according to a Latin hypercube sampling strategy applied on the *n*-dimensional domain of the *problem* to be solved.
2. **FE-EVALUATOR**(**x**, *problem*) – this function computes all the *problem*-specific target parameter values of a given individual **x** using the FE-based performance evaluation sketched in Figure 3.2. If **x** is unfeasible, this evaluator will flag it correspondingly.

3. **INITIALIZEMLPGRIDSEARCH**(*problem*) – this function returns a set containing all the MLP training configurations that are to be tested during the best parameter grid search. The number of specific configurations is *problem*-dependent as previously described in Section 4.2.5.

4. **TRAINMLPSURROGATE**(*configuration*, *target*, *Set*) – based on the supplied parameter *configuration* and training *Set*, this function creates a MLP model for the given *target* using the backpropagation-based process presented in Section 4.2.3.

5. **SELECTBESTMODEL**(*Set*) – given a *Set* of MLP surrogate models, this method applies the automatic model selection procedure described in Section 4.2.4 to select and return a high-quality robust surrogate.

6. **MLP-EVALUATOR**(**x**, *problem*, *Set*) – this function computes all the *problem*-specific target parameter values of a given individual **x** using a *Set* of target-specific surrogate models. In light of the arguments presented in Section 4.1.2, all individuals processed by this surrogate-based fitness evaluator are flagged as feasible.

7. **EXTRACTPN**(*Set*) – this method extracts and returns the *PN* from a given *Set* of individuals.

8. **CREATEOFFSPRING**(*Set*, *count*) – starting from a parent *Set* of individuals, this function generates a given *count* of offspring via the standard NSGA-II mutation operators, viz. SBX and PM, described in Section 2.3.3.

9. **NONDOMINATEDSORTING**(*Set*, *count*) – this method restricts an initial *Set* of individuals to a certain *count* via the non-dominated sorting strategy proposed by NSGA-II.

4.3.2 Comparative Performance on CIMOOPs

In order to check the success of hybridizing MOEAs with global surrogate models given the particularities of our industrial MOOPs, we performed on-line ConvOpt and HybridOpt tests on IndMOOP1, IndMOOP2 and IndMOOP3.

We remind the reader that in the case of ConvOpt (i.e., domain-adapted NSGA-II) we used a population size of 50, we ran an optimization for 100 generations and we extracted the *PN* from all the feasible individuals generated during the run. In the case of HybridOpt, we implemented Algorithm 2 and ran the main **HYBRIDOPT**( ) function with different call arguments.

For each experiment, we analyzed the final *PN* obtained at the end of the run using the hypervolume (Ind_H) and the generational spread indicators (Ind_{GS}) introduced in Section 3.2.1. Since we are dealing with real-life MOOPs with unknown solutions, the PF_{true} required in (3.10) to compute the hypervolume indicator was obtained by aggregating (I) best-performing legacy design solutions and (II) expectations of various DMs involved in the electrical drive design process. In the case of IndMOOP3 neither optimization run was

able to truly fulfill the DMs' expectations and this is reflected in the generally low Ind_H values obtained for this problem.

For these on-line experiments, we also present information regarding the average wall-clock time (in minutes) required by ConvOpt and HybridOpt to perform the optimizations when distributing the fitness and grid-search computations over an average of 40 HTCondor™-managed computation nodes. The notation used for this new time-related performance indicator is: Ind_{time}

During the initial HybridOpt tests we used the settings: $popSize_{FE} = 50$, $popSize_S = 50$, $feGen = 25$ and $sGen = 75$. We mark this HybridOpt parameterization option as $ParSet - A$. In Table 4.4 we present the comparative performance of ConvOpt and HybridOpt $ParSet - A$. The results are averaged over five independent runs for each MOOP and they indicate that the performance of our surrogate-enhanced optimization strategy is very good for problems IndMOOP1 and IndMOOP2. In these two cases, on average, the HybridOpt *PN*s have comparable Ind_H values ($\approx 1.5\%$ worse), better Ind_{GS} values and were computed 2.72 times faster (IndMOOP1) and 3.61 times faster (IndMOOP2) than their ConvOpt counterparts. In other words, HybridOpt is able to reduce the average wall-clock optimization time by $\approx 63\%$ for IndMOOP1 and $\approx 72\%$ for IndMOOP2 while maintaining overall solution (i.e., *PN*) quality.

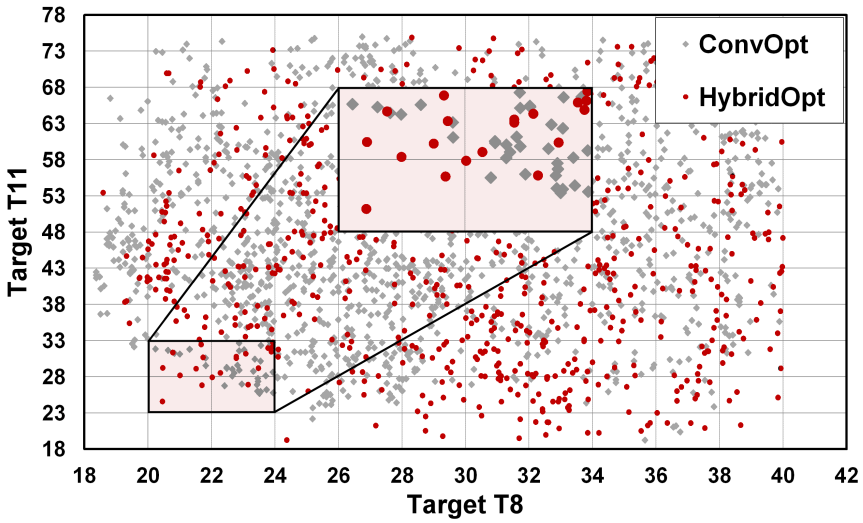
Table 4.4: The averaged performance over five runs of the MOEA-based conventional and hybrid MOO processes. For each problem, we highlight the best result for each performance indicator.

Indicator	IndMOOP1		IndMOOP2		IndMOOP3	
	ConvOpt	HybridOpt *ParSet-A*	ConvOpt	HybridOpt *ParSet-A*	ConvOpt	HybridOpt *ParSet-A*
Ind_H	0.9532	0.9393	0.8916	0.8840	0.4225	0.3691
Ind_{GS}	0.7985	0.6211	0.8545	0.8311	0.4120	0.4473
Ind_{time}	2696	991	3798	1052	4245	2318

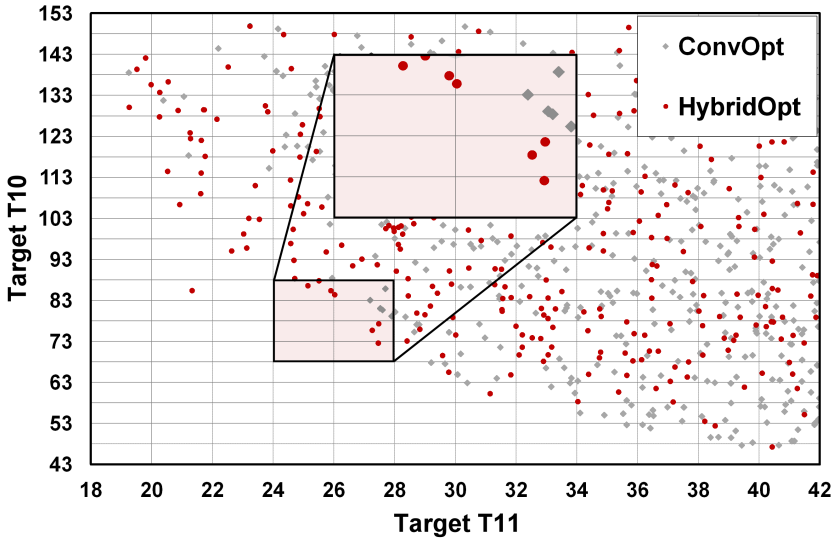
On the highly constrained problem IndMOOP3, the surrogate-enhanced optimization process performs worse as, on average, its reported Ind_H values are $\approx 13\%$ lower than those of ConvOpt. The data from Table 4.4 also indicates that for IndMOOP3, ConvOpt also produces solutions (*PN*s and associated *PF*s) that display better Ind_{GS} values. Nevertheless there are also two benefits associated with the HybridOpt $ParSet - A$ runs:

1. the wall-clock duration of the optimizations is reduced by $\approx 46\%$;

2. the surrogate-enhanced evolutionary process is still able to discover valuable (electrical drive) designs for IndMOOP3 in regions that have turned out to be very interesting to the DMs. For instance, in Figure 4.6 we present two different comparative 2D projections of the best *PF*s (according to Ind_H) discovered by ConvOpt and HybridOpt $ParSet - A$ and we highlight two regions of interest, as defined by DMs.



(a) 2D PF projection no. 1



(b) 2D PF projection no. 2

Figure 4.6: 2D Projections of final PNs obtained after applying ConvOpt and HybridOpt on IndMOOP3. We highlighted and magnified two regions (from objective space) that were of special interest to the DMs.

The overall mediocre performance of HybridOpt $ParSet - A$ on IndMOOP3 can largely be attributed to the aforementioned fact that this MOOP is very affected by the formalization constraints mentioned in Section 2.1.2. These constraints determine a very high ratio of geometrically invalid individuals to be generated during the surrogate-based part of each run. Consequently, during the FE-based re-evaluation stage of HybridOpt, many designs from the surrogate-only PN are marked as invalid.

Wanting to improve on these initial HybridOpt results, we performed extra experiments with two more parameter configurations:

1. $ParSet-B$: $popSize_{FE} = 50$, $popSize_S = 100$, $feGen = 25$ and $sGen = 75$

2. $ParSet-C$: $popSize_{FE} = 50$, $popSize_S = 100$, $feGen = 25$ and $sGen = 175$

The obvious idea behind these HybridOpt configurations was to allow for more designs to be generated (and tested) during the surrogate-based part of each run.

Table 4.5 presents obtained results and they indicate that this basic tactic of generating more designs during the surrogate-based execution is rather successful. One noticeable downside of this brute-force approach, especially for MOOPs with several (i.e., > 3) objectives, is that the resulting surrogate-only *PN*s also tend to become larger as they are extracted from a much larger set of evaluated individuals. This directly translates into a longer FE-based re-evaluation stage that reduces the average speed improvement associated with HybridOpt to:

- $\approx 51\%$ for IndMOOP1, $\approx 69\%$ for IndMOOP2 and $\approx 22\%$ for IndMOOP3, when applying $ParSet-B$;
- $\approx 27\%$ for IndMOOP1, $\approx 63\%$ for IndMOOP2 and $\approx 14\%$ for IndMOOP3 when applying $ParSet-C$.

Table 4.5: The averaged performance over five runs of HybridOpt when allowing for more individuals to be evaluated during the surrogate-based parts of the run. The results that are not better than the complementary results produced by ConvOpt are highlighted.

Indicator	IndMOOP1		IndMOOP2		IndMOOP3	
	HybridOpt *ParSet-B*	HybridOpt *ParSet-C*	HybridOpt *ParSet-B*	HybridOpt *ParSet-C*	HybridOpt *ParSet-B*	HybridOpt *ParSet-C*
Ind_H	0.9534	0.9535	0.9053	0.9114	0.3910	0.4082
Ind_{GS}	0.6103	0.5896	0.7442	0.6814	0.3981	0.3912
Ind_{time}	1332	1968	1156	1375	3315	3631

More importantly, when using the $ParSet-B$ and $ParSet-C$ parameterizations, on average, HybridOpt is also able to slightly outperform (by at most $\approx 2\%$) ConvOpt with regard to the Ind_H indicator. On the more problematic IndMOOP3, although seriously improved, the *PN*s produced by HybridOpt are only better with regard to the Ind_{GS} metric.

4.4 Improving Surrogate Modeling Efficiency

4.4.1 Faster Surrogate Model Construction Strategies

First and foremost, we must remark that **the results obtained with HybridOpt during the comparative on-line MOO tests generally validate our proposed methodology** to

create and use on-the-fly global surrogate models to enhance the performance of a standard MOEA when attempting to solve CIMOOPs.

Nevertheless, previously reported on-line and off-line modeling results (like those from Section 4.2.5) also indicate that the proposed on-the-fly surrogate construction process is extremely time-intensive and thus affects the overall optimization speed-ups that can be achieved by HybridOpt. The computational burden is of course largely owed to the best-parameter grid search that must be performed when trying to create a very good surrogate model for a given non-linear target parameter.

There are only two possible options to considerably reduce the duration of the best-parameter grid searches. The first one is to significantly lower the total number of allowed parameter combinations by reducing the ranges and / or the number of steps. This approach would be very risky as it would very likely diminish the general performance of the entire surrogate modeling process. Furthermore, the approach would not mitigate the frequent problem that the entire (cluster-distributed) surrogate creation process must wait after a few very computationally-intensive surrogate training jobs that have been generated during the grid search. In extreme cases, such training jobs may last more than 15 minutes for example. This happens because the previously-described surrogate construction strategy requires the termination of all model training jobs before proceeding with the model selection step. Of course one could forcefully terminate very long model training jobs but, given the heterogeneous nature of the cluster system, there are no guarantees that by doing this, one is not in fact disposing of the best surrogate models.

A second approach aimed at reducing the general duration of the grid searches is to reduce the size of the data sets used when training the potential surrogate models. Given the particularities of our modeling needs, we have shown in Section 4.2.6 that constructing useful training sets requires the computation of several (e.g., 25) FE simulation-based generations. However we did not explore whether the rough value $feGen = 25$ is influenced by:

- *density related aspects* - the MOEA requires 25 generations to evolve enough individuals in the "interesting sections" of design space, thus enabling the construction of good global surrogate approximation models;
- *exploration related aspects* - the MOEA requires 25 generations to cover / explore a large enough part of the design space, thus enable the construction of good global surrogate models.

If the latter case were (even partially) true, one can theoretically restrict the size of the training sets used to construct global surrogate models without seriously influencing the prediction quality exhibited by the latter. In order to empirically check this aspect, we considered three types of training sets:

1. $Full(feGen)$ - a training set that contains all the feasible individuals (data samples) found during the first $feGen$ generations of the MOEA run.
2. $Rand(count, feGen)$ - a training set that contains at most *count* samples randomly extracted from $full(feGen)$

3. $Trim(count, feGen)$ - a training set that contains at most *count* samples extracted from $full(N)$ in three different steps:

   (a) in the first step we extract n_{targ} samples that correspond to the data samples that contain the worst (highest) values for each of the elicited objectives of the MOOP we wish to solve. These anti-optimal samples are the same ones one would need to synthesize the $\mathbf{p}^{\mathbf{ref}}$ point required when computing (3.9).

   (b) in the second step we aim to extract samples that correspond to the *PN* of $Full(feGen)$. If the size of this *PN* is larger than $n_{PS} = count - n_{targets}$, we apply a Pareto-based method (like the non-dominated sorting [65] or environmental selection [66] strategies described in Section 2.3.3) to select (and subsequently extract) exactly n_{PS} individuals.

   (c) in the third (optional) step, we randomly extract $n_{rand} = count - n_{targets} - n_{PS}$ samples from the ones remaining in $Full(feGen)$

   Based on the described procedure, it should be obvious that the more elaborate $Trim(count, feGen)$ training set actively tries to incorporate MOO domain knowledge to reduce the size but preserve as much of the density and search space exploration characteristics of the original [i.e., $Full(feGen)$] data set.

By using the syntagm "as much of" in the previous statement, we are in fact indirectly acknowledging that a certain deterioration of data quality is to be expected in the case of the Pareto-trimmed (and more generally reduced) training sets. In order to try and compensate this, one of the most natural (and simple to implement) ideas is to extend the model selection process described in Section 4.2.4 and to adapt it for constructing ensemble surrogate predictors. For testing this idea, given a set of potential surrogate models that resulted from a best-parameter grid search, we experimented with four different model selection strategies:

1. si^{best} - selects the single surrogate model with the highest 10-fold cross-validation R^2 performance. This is more or less the standard model selection strategy that we also previously applied in Section 4.2.5.

2. $si^{thr(p_q)}$ - selects a single surrogate model according to the threshold-based method described in Section 4.2.4. The p_q parameter indicates the percentage of top performing models that is used to compute the threshold value – e.g., $p_q = 5$ indicates that the best performing 5% from all the models computed during the grid search will be used for establishing the quality threshold.

3. $en^{best}(s)$ - selects a number of s individual (base) models that will form an ensemble surrogate predictor. These base models are chosen in decreasing order of their R^2 cross-validation performance and therefore, $en^{best}(s)$ can be seen as a basic extension of si^{best}. The final prediction of the ensemble is a simple average over the individual predictions of the base models. N.B. There is virtually no extra computational cost associated with this ensemble strategy since all the individual base models are created by default during the best-parameter grid search.

4. $en^{thr(p_q)}(s)$ - also selects s base models in order to construct an averaging ensemble predictor. In this case, as we aim to extend the $si^{thr(p_q)}$ strategy, after computing the threshold, individual models that display the best prediction quality vs. complexity trade-off are selected iteratively (starting with the least complex one) until the model count limit (i.e., s) is reached or until the model with the highest estimated prediction quality is selected. In the case of the latter, the ensemble structure is filled by applying a circular (round-robin) on the already selected base models. Figure 4.7 presents a graphical description of this ensemble selection procedure.

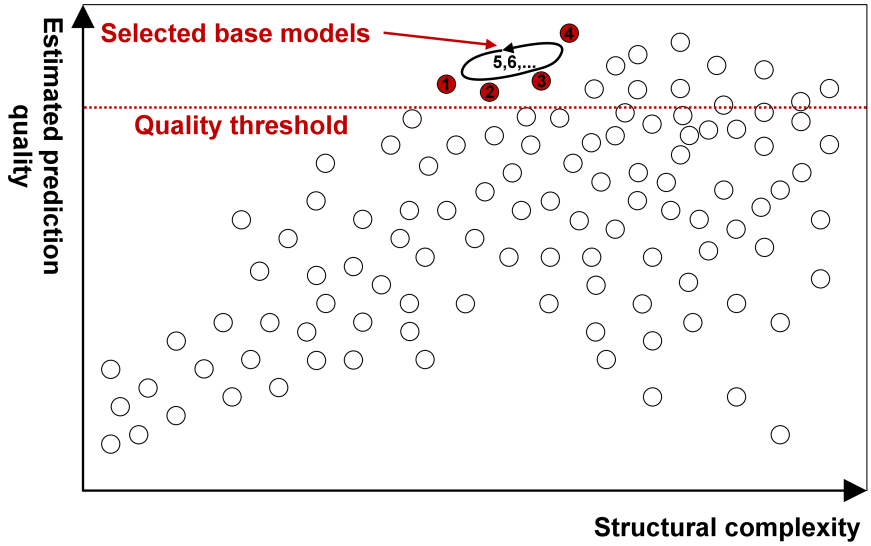


Figure 4.7: An illustration of the proposed $en^{THR(p_q)}(s)$ model selection strategy. If our goal is to create an ensemble of size 10 (i.e., $s = 10$), the final surrogate predictor will contain three copies (instances) of each of the base models "1" and "2" and two copies for each of the base models "3" and "4".

We define a **surrogate model construction strategy** as being **a combination of** a specific **modeling method**, a specific **model selection strategy** and a specific **training set** structure. For example, we mark with $MLP - en^{thr(5)}_{Trim(250,25)}(10)$ a strategy that creates an ensemble surrogate predictor of 10 MLP base models chosen according to a 5% high-quality threshold and trained on a Pareto-trimmed data set of no more than 250 samples selected from all the feasible designs evaluated in the first 25 generations of a MOEA run.

4.4.2 Comparative Performance

In order to assess the performance of the new ensemble-based surrogate construction strategies and also confirm some of the findings reported in Sections 4.2.5 and 4.2.6, we decided to perform the off-line modeling experiments reported in this section on three new CIMOOP-based data sets.

These new data sets also correspond to optimizations of IndMOOP1, IndMOOP2 and IndMOOP3. Therefore, they contain the same eleven targets presented in Table 4.3. The

three optimization runs that produced the new data sets also ran for 100 generations with a population size of 50 but the particular MOEA that we applied was SPEA2. For each problem, the time-wise evolution of feasible designs generated during the SPEA2 run is fairly similar to the one reported for NSGA-II in Section 4.2.2. Concrete values are presented in Table 4.6 and, based on them, it is quite fair to assume that the performances of different modeling techniques should not differ too much from those reported in Section 4.2.5.

Table 4.6: The number of feasible designs generated during the SPEA2 runs.

Problem	Feasible designs after generation				
	25	33	50	75	100
IndMOOP1	1206	1598	2430	3605	4732
IndMOOP2	897	1222	1855	2743	3622
IndMOOP3	799	1146	1745	2685	3582

We have performed all the surrogate modeling experiments from this section with the setting $feGen = 25$ and, while maintaining the MLP best-parameter grid search we slightly modified the best-parameter grid searches for SVR and RBFNs that have been defined in Section 4.2.5. This was done in order to remove a few of the parameter combinations that did not prove successful with regard to the targets of the three CIMOOPs and to test if other potential parameter combinations would be better. For the new experiments, during the grid searches we created:

1. 630 models for each target when using SVR as we varied C between $\{2^{-4},\ldots,2^{9}\}$, γ between $\{2^{-3},2^{-2},\ldots,2^{5}\}$ and ε between $\{0.005,0.01,0.025,0.05,0.1\}$;

2. 513 models for each target when using RBFNs as we varied the number of clusters between $\{1,\ldots,5,10,15,\ldots,50,60,\ldots 100,125\ldots,200,250,300,400,500\}$ and the min. cluster standard deviation between $\{0.05,0.1,0.50,1.0,\ldots,5,6,\ldots,10,15,20\}$.

This time, all the surrogate modeling tests were executed on the same machine (8-core CPU, 4GB of RAM) in a multi-threaded environment that used a maximum of 6 threads simultaneously. The time required to complete each surrogate training task was measured independently and therefore, when also factoring in the homogeneous hardware environment, the time-wise results we report are much more trustworthy than the cluster-based ones from Section 4.2.5.

In a first series of experiments we wanted to (empirically) corroborate three of our earlier assumptions and findings:

1. MLPs tend to produce slightly more accurate models than the other non-linear modeling methods;

2. the choice to start the surrogate modeling stage after computing an initial number of $feGen = 25$ FE-based generations is correct as it enables the construction of accurate surrogate models;

3. the automatic model strategy proposed in Section 4.2.4 [marked with $MLP - si^{thr(p_q)}_{Full(25)}$ when using the latest notations] is able to deliver some improvement over the standard approach of choosing the best cross-validation performer [marked with $MLP - si^{best}_{Full(25)}$ when using the latest notations].

In Table 4.7 we present surrogate modeling results obtained with MLPs, SVR and RBFN on the six non-linear targets proposed by IndMOOP1, IndMOOP2 and IndMOOP3. Linear regression results are also provided for comparison. All the tabular data seems to confirm earlier findings as, when considering the average performance over all six targets, $MLP - si^{thr(5)}_{Full(25)}$ delivers the best performance (i.e., average $R^2 = 0.9690$) achieving an estimated prediction quality that is very close to $R^2 = 0.9720$ – the complementary result from Table 4.3 we reported for $MLP - si^{best}_{Full(33)}$ on the data sets produced by NSGA-II optimization runs.

Table 4.7: Information regarding the estimated prediction quality of single-model surrogates.

Target	R^2 on test data				
	Linear	$MLP - si^{best}_{Full(25)}$	$MLP - si^{thr(5)}_{Full(25)}$	$SVR - si^{best}_{Full(25)}$	$RBF - si^{best}_{Full(25)}$
T1	0.8594	0.9636	0.9619	0.8917	0.8842
T2	0.7712	0.9390	0.9531	0.8886	0.7687
T4	0.7909	0.9434	0.9577	0.9584	0.8117
T5	0.9201	0.9943	0.9957	0.9964	0.9874
T10	0.9245	0.9894	0.9816	0.9831	0.9784
T11	0.8934	0.9616	0.9644	0.9637	0.9500
Average	0.8599	0.9652	0.9690	0.9469	0.8967
Rank	5	2	1	3	4

A second round of experiments was aimed at evaluating the performance of ensemble-based surrogates. The first two columns of Table 4.8 indicate that **by averaging the results of the best performing 10 models constructed during the MLP best-parameter grid searches, we generally improve surrogate prediction quality** when using the $Full(25)$ training sets. Interestingly, the model selection method based on thresholding [i.e., $MLP - en^{thr(5)}_{Full(25)}(10)$] also displays a slight edge over the strategy that simply selects the 10 most accurate cross-validation models [i.e., $MLP - en^{best}_{Full(25)}(10)$].

The last (and most interesting) experiments we performed investigated if ensemble-based surrogate models trained on reduced training sets can also deliver a very good and time-wise stable prediction quality. As such, starting from $Full(25)$ we constructed training

Table 4.8: Information regarding the estimated prediction quality of ensemble-based surrogates.

Target	R^2 on test data			
	$MLP-en^{best}_{Full(25)}(10)$	$MLP-en^{thr(5)}_{Full(25)}(10)$	$MLP-en^{thr(5)}_{Rand(250,25)}(10)$	$MLP-en^{thr(5)}_{Trim(250,25)}(10)$
T1	0.9586	0.9637	0.9525 ± 0.0054	0.9460 ± 0.0028
T2	0.9556	0.9549	0.9510 ± 0.0015	0.9517 ± 0.0050
T4	0.9573	0.9596	0.9502 ± 0.0108	0.9562 ± 0.0032
T5	0.9961	0.9960	0.9936 ± 0.0034	0.9920 ± 0.0035
T10	0.9889	0.9887	0.9865 ± 0.0020	0.9856 ± 0
T11	0.9763	0.9752	0.9739 ± 0.0010	0.9772 ± 0
Average	0.9721	0.9730	0.9676	0.9681
Rank	2	1	4	3

sets containing only 250 samples [i.e., $Rand(25)$ and $Trim(250,25)$] on which we applied the $MLP-en^{thr(5)}(10)$ surrogate model construction strategy. Because both methods of constructing reduced training usually feature a stochastic component, we performed five independent runs of each surrogate modeling experiment and in the last two columns of Table 4.8 we preset the R^2 averages and standard deviations. The results show that for the considered parameters (i.e., 250 samples out of the ones produced with $feGen = 25$), there is virtually no difference between applying a random or a Pareto-based trimming of the original training sets. Nevertheless the achieved R^2 values indicate that both of the ensemble-based surrogates display a very high average prediction quality (close to the one of $MLP-si^{thr(5)}_{Full(25)}$ – the best performing single-model surrogate). Furthermore, the generational R^2 plots from (the left side of) Figures 4.8 and 4.9 indicate that ensemble-based surrogate models created using reduced (Pareto-trimmed) training sets[12] are able to deliver a very stable predictive behavior that matches the one of $MLP-en^{thr(5)}_{Full(25)}(10)$ – i.e., the best performing global surrogate model construction strategy we have evaluated.

The last two columns of Table 4.9 show that when using a reduced data set of only 250 data samples, the average individual (base) model training time is reduced by $\approx 80\%$ from 60.08 seconds to 12.74 seconds. Furthermore, in the right side plots of Figures 4.8 and 4.9 we present the distributions of MLP model training times when using the $Full(25)$

[12]For each target, given the five independent runs we performed with $MLP-en^{thr(5)}_{Trim(250,25)}(10)$, we selected the median ensemble model according to training R^2 performance.

and $Trim(250,25)$ training sets during the best-parameter grid searches. Since in the case of $Trim(250,25)$, only a handful of models (from the 5376 constructed in total) require a training time larger than 75 seconds, the distribution plots present compelling evidence that using a smaller training set also drastically reduces the variance of surrogate model training times. This is extremely important in our case because it makes load balancing a lot easier when we distribute the best-parameter grid search over the cluster computing environment.

Table 4.9: Information regarding the average training times of the surrogate models and the total wall-clock time required by the best-parameter grid searches.

Target	Surrogate model training time [s]							
	$MLP-Full(25)$		$SVR-Full(25)$		$RBF-Full(25)$		$MLP-Trim(250,25)$	
	avg.	total	avg.	total	avg.	total	avg.	total
T1	88.22	70627	307.96	195248	112.59	62603	12.35	10662
T2	85.84	74081	291.99	185125	113.66	63539	15.51	13447
T4	73.93	63661	330.39	209140	114.46	63640	12.81	11064
T5	32.25	28313	120.82	77930	302.05	170960	12.34	10875
T10	45.96	55883	106.68	68810	212.85	120478	12.46	15152
T11	34.28	43298	114.70	74100	217.32	123007	11.02	13399
Avg.	60.08	55977	212.09	135059	178.82	100705	12.74	12433
Rank	2	2	4	4	3	3	1	1

To summarize, given the three considered CIMOOPs, viz. IndMOOP1, IndMOOP2 and IndMOOP3, the above presented experiments show that:

- a strategy that uses an ensemble of MLPs trained over Pareto-trimmed data sets – i.e., $MLP-en^{thr(5)}_{Trim(250,25)}(10)$ – produces global surrogate models that display the best trade-off between prediction quality and training time;
- when only focusing on the quality and stability over time of surrogate predictions, a strategy that creates an ensemble of MLPs trained using all the available samples – i.e., $MLP-en^{thr(5)}_{Full(25)}(10)$ – is the overall best performer.

.

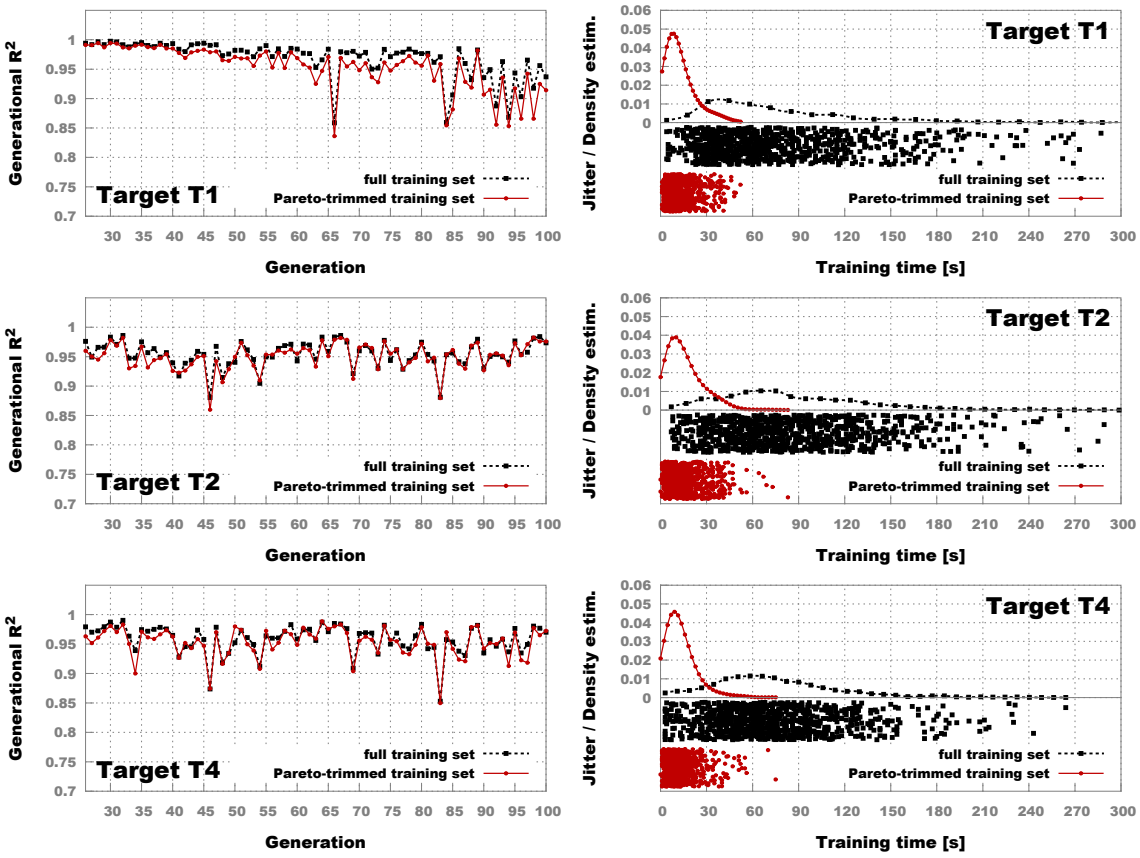


Figure 4.8: Comparative prediction quality (generational R^2) and training time performance of two surrogate ensemble models, $MLP-en^{thr(5)}_{Full(25)}(10)$ and $MLP-en^{thr(5)}_{Trim(250,25)}(10)$, for the non-linear targets of IndMOOP1.

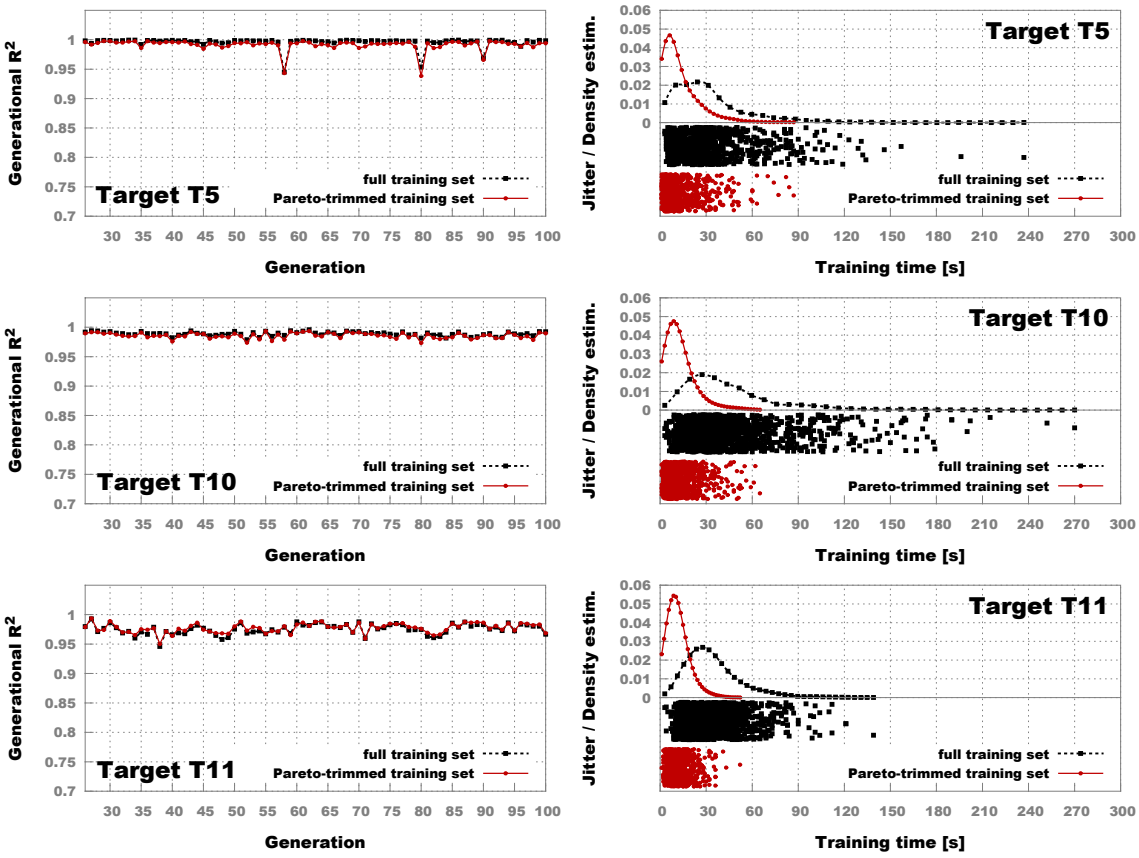


Figure 4.9: Comparative prediction quality (generational R^2) and training time performance of two surrogate ensemble models, $MLP-en^{thr(5)}_{Full(25)}(10)$ and $MLP-en^{thr(5)}_{Trim(250,25)}(10)$, for the non-linear targets of IndMOOP2 and IndMOOP3.

4.4.3 Remarks Regarding Surrogate Modeling

Throughout the current chapter we have presented (and improved) one possible method for hybridizing a standard MOEA with surrogate (regression) models in order to considerably speed-up the optimization process when dealing with CIMOOPs. Of course, our approach is not the only possibility, even when considering particularities like the need to construct the surrogates on-the-fly and to only present final solutions that are validated by an exact fitness estimation process (like the one from Figure 3.2 in our case). For example, in [136], Bittner and Hahn consider CIMOOPs from the exact application domain as us and describe a multi-objective PSO hybridized with an ingenious Kriging-based [137] surrogate that self-adapts in an on-line fashion: at each iteration (computational cycle) of the PSO, the best 30% new particles (i.e., individuals / design vectors) that are deemed as the most promising (able to advance the current Ind_H) are re-evaluated with FE simulations and the Kriging-based surrogate is updated. Considering the introductory descriptions from Section 4.1, this MOO approach applies surrogates in a pre-selection context.

On the one side, the approach from [136] is very interesting and would probably fair quite well on a problem like IndMOOP3 that suffers heavily from formalization constraints that determine the evolution of many infeasible designs. This is because the pre-selection (PSO-Kriging) approach would not allow an infeasible design with very good surrogate-estimated performance to influence the general search strategy since the erroneous design would be disregarded within one iteration. On the other side, when compared to HybridOpt, the pre-selection based optimization procedure is very likely to deliver smaller computational improvements (for comparable *PN* performance) on CIMOOPs that are better defined and suitable for constructing accurate global surrogate models. This is because, its inherent (very fine) step-by-step "FE-evaluation $\rightarrow$ modeling $\rightarrow$ surrogate-evaluation" strategy is likely to perform many unneeded FE simulations (for designs that will not be part of the final *PN*).

Nevertheless, given the very good performances described in [136], we also began to **consider placing the proposed HybridOpt procedure inside a (rather coarse) on-line learning paradigm**[13] that requires some surrogate retraining stages. In Chapter 7, we discuss this matter in more detail as we reshape the HybridOpt process in a fashion that bears similarities to the one proposed by Nain and Deb in [138].

Furthermore, in order to alleviate the problem of potentially infeasible designs crucially affecting the behavior of the MOEA during the surrogate-based parts of the run, we decided to **construct surrogate feasibility models**. The role of this new type of surrogates is to assess (classify) if a given individual (design parameter vector) is erroneous or not. Of course, a wealth of classification methods can be used for this task and by performing a systematic analysis over CIMOOPs of interest (as described in Section 4.2.5 for the regression case), one should be able to narrow down the list to a few very promising classification paradigms. Given the fact that ANNs in general are very likely to at least be short listed as one of the top contenders, we decided to construct surrogate feasibility classifiers by adapting the MLP-centered strategy described throughout Section 4.2 as

[13] Especially since we can considerably minimize the duration of the surrogate modeling stage by switching to ensemble-based averaging predictors trained over reduced data sets.

follows:

- the training sets T were modified to include both the feasible and the infeasible designs generated during the FE-based part of the MOEA run;
- the (training) data samples associated with feasible designs are labeled with "1.0" and the ones associated with infeasible designs are labeled with "0.0" meaning that, considering the notations from (4.3), the objective $o_1(\mathbf{x})$ of the surrogate feasibility modeling process is defined as:

$$o_1(\mathbf{x}) = \begin{cases} 1.0 & \text{, if } \mathbf{x} \in D^n \\ 0.0 & \text{, otherwise} \end{cases} \tag{4.7}$$

- given a sample $\mathbf{x}$ to be tested, it is labeled as feasible by a single-model surrogate if and only if $|f(\mathbf{x},W,a) - 1.0| \geq |f(\mathbf{x},W,a) - 0.0|$, where $f(\mathbf{x},W,a)$ marks the output of the MLP model;
- the final feasibility decision of an ensemble of MLPs is made by first computing the individual option of each base model and then opting for the majority decision (i.e., we apply a classical equal voting strategy);

By introducing new concepts like fast-to-train ensemble-based surrogates and MLP-based feasibility classifiers, we are able to seriously improve the general efficiency of the surrogate modeling process. In Chapter 7, we present on-line results that illustrate how the improved on-the-fly surrogate modeling process significantly boosts the performance of a generally competitive MOEA that is applied on a very difficult CIMOOP.

Although quite successful in practice, we have also encountered a few CIMOOPs on which the HybridOpt surrogate modeling strategy did not perform very well. This was because of what we perceive as a general inability to easily construct accurate global regression models[14] for the non-linear targets contained in these electrical drive design MOOPs.

In order to generally improve the duration of MOEA-based optimizations (especially on CIMOOPs that prove challenging for surrogate modeling), in the next two chapters we explore potential enhancements that are particularly focused on speeding-up the general search mechanism proposed by MOEAs – i.e., reducing the *nfe* required for finding good *PN*s.

[14]We applied MLPs (also featuring two hidden layers), SVR and RBFNs with considerably expanded best-parameter grid searches on training sets obtained with the setting $25 \leq feGen \leq 50$ and only managed to obtain average R^2 values < 0.65 on the complementary test subsets.

Chapter 5

Making the Correct Parallelization Choice

5.1 Motivation and General Idea

5.1.1 Parallel and Distributed MOEAs

The simplest idea that displays immediate benefits when wanting to reduce the run-time of a MOEA on a CIMOOP is to parallelize / distribute the computations over a cluster or a grid environment. There are several well-established paradigms (i.e., architectural and conceptual models) that describe how such a parallelization can be performed. The available options range from very simplistic *master-slave parallelization* (**MSP**) approaches to increasingly more specialized (complex) island, diffusion, hierarchical and hybrid parallelization models. For an in depth description of these parallelization options and useful examples of how they can be (have been) applied in the context of MOO, please see Chapter 8 of [13].

Throughout this chapter we are only concerned with the very simplistic MSP paradigm. When used to "enhance" run-time of an EA this approach dictates that: all fitness evaluations are distributed between several slave nodes (computational units) and all the evolutionary operations (i.e., parent selection, crossover, mutation, selection for survival) are performed on a master node (computational unit). The simplicity of this model makes it very easy to implement and to adapt to the particularities of the available hardware infrastructure. Therefore the MSP model is the first choice of most practitioners and it is widely used.

When applied on EAs, the MSP paradigm offers practitioners two main parallelization choices:

- a *generational master-slave parallelization scheme* (**GEN-MSPS**) that does not alter the algorithmic behavior of the EA one wishes to parallelize;
- a *steady-state asynchronous master-slave parallelization scheme* (**SSA-MSPS**) that usually changes the algorithmic behavior of the EA by forcing it to adopt an evolutionary strategy that is very similar similar to the steady-state selection scheme described by Goldberg [139].

Considering the proposed general EA structure from Algorithm 1, the generational evolutionary model is obtained with the setting $genSize = popSize$ and Goldberg's steady-state scheme is obtained with the setting $genSize = 1$. Therefore, adopting one or the other algorithmic behaviors is usually simply a matter of preference. Deciding which of these two EA behaviors delivers better results is an age old dilemma in the field of evolutionary computation. As such, while many advocate the usage of generational EAs, in their study from 2008 [140], Durillo et al. show evidence that applying a (synchronous) steady state approach when performing a MOEA run can bring improvements in terms of achievable *PN* quality.

The aim of the research (analysis) reported in this chapter is to offer MOEA practitioners (and DMs) valuable (or at least interesting) insights that can help them make the right MSP choice based on the particularities of their MOO processes.

5.1.2 Why Focus on the Basic MSP Paradigm?

The question in the title is quite legitimate, especially when considering the impressive results reported in [13] that were obtained with more advanced parallelization models.

The answer is that, apart from its wide usage and large potential for immediate adoption when first switching to a parallel / distributed MOEA, the MSP paradigm is:

- interesting to study in the context of MOEAs because of its *natural interaction* with the $(\mu + \lambda)$ Pareto-based strategy that governs most MOEAs. When choosing to adopt GEN-MSPS, one is sure to obtain the time-wise improvement brought by moving to a parallel hardware infrastructure. When opting for SSA-MSPS (for reasons we shall soon present), one is also changing (maybe unwittingly) the governing principle of the MOEA to a $(\mu + \lambda^{1+})$ Pareto-based strategy with the hope of achieving an even greater time-wise improvement.

- *seriously understudied*, when taking into account that making the right choice for GEN-MSPS or SSA-MSPS can seriously impact the performance of the MOO run.

- an important integral part of other, more advanced, hybrid parallelization models (as we briefly described in Chapter 7) and understanding how to make a correct choice MSP has an obvious positive impact on the hybrid parallel model as well.

All in all, many empirical tests on EAs have shown that the result obtained when opting for SSA-MSPS instead of GEN-MSPS can be significantly better or significantly worse, depending from case to case, and we would like to determine the most important factors that influence this outcome in the case of a $(\mu + \lambda)$ MOEAs. More specifically, given a certain optimization setup, the goal of our analysis to determine which of the two MSP schemata is more likely to help the MOEA reach a better *PN*.

N.B. All the MOEAs presented in detail in Section 2.3 with the exception of MOEA/D fall within the $(\mu + \lambda)$ class. Because of it's unique evolutionary strategy centered around cooperation between neighboring solutions, MOEA/D is a very interesting candidate for a diffusion-based parallelization.

5.1.3 The Proposed Analysis

Before proceeding with the description of our proposed method for comparing GEN-MSPS and SSA-MSPS performance, we must first introduce some basic concepts.

When considering a process that is parallelized / distributed via a MSP architecture, one must distinguish among two types of computational tasks:

1. **remote tasks** - very *time-intensive computations* that are performed on the slave nodes;

2. **sequential tasks** - include all the operations that must be performed on the master-node in order to *create*, *dispatch* and *retrieve* remote computation tasks;

When considering parallel and distributed MOEAs, the *"create"* part of the sequential tasks mainly includes the application of genetic operators. Apart from these, in real-world master-slave parallelization setups for MOEAs, the duration of the sequential tasks can also be affected by the fact that lengthy pre-evaluation steps must be performed locally (on the master node) for each generated individual, before dispatching the individual for remote fitness evaluation on the slave nodes. For example, in the case of our CIMOOPs, during the electrical drive performance evaluation process illustrated in Figure 3.2, the "2D/3D model construction" must be performed locally (on the master node) because of license-related issues related to the used CAD software. Whenever the average duration of the sequential tasks carried out on the master node is significant with regards to the average duration of the fitness evaluation tasks, the optimization process is said to display a low **parallelization ratio**. In this case, the speed-up that can be achieved by employing a parallel / distributed architecture is affected (q.v. "*Amdahl's law*").

Apart from the parallelization ratio, another aspect that must be considered refers to the **heterogeneity of the time-wise distributions of the** remote and sequential tasks. As already established, in the case of MOEAs, the remote computational tasks are represented by the **fitness evaluation** functions (procedures). Literature that focuses on the effects of fitness function time-wise heterogeneity on making a good MSP choice for MOEAs is scarce. Nevertheless, a study by Yagoubi et al. from 2011 [141] shows some empirical evidence that, for MOOPs that display a heterogeneous (i.e., non-constant) time-wise distribution of the fitness function, the steady state asynchronous parallelization is somewhat better in terms of convergence (Pareto quality and global run-time) than the generational approach. A key focus point of our analysis is this important empirical result from [141].

The diagram in Figure 5.1 provides a general illustration of the computation cycles generated by both GEN-MSPS and SSA-MSPS when applying these parallelization schemata on a generic $(\mu + \lambda)$-styled MOEA. As expected (given previous remarks), these cycles are nearly identical and the only difference is in the type of synchronization blocks that regulate the MOEA dynamics:

- the λ-sync block used by GEN-MSPS requires that λ offspring must be stored in the "offspring insertion pool" before the selection for survival operation can take place and, subsequently, a number of λ new offspring can be generated;

- the 1^+-sync block used by SSA-MSPS allows the selection for survival operation to take place when at least one offspring is stored in the "offspring insertion pool". After the operation takes place, as many new offspring are generated as those that have taken part in the selection for survival process.

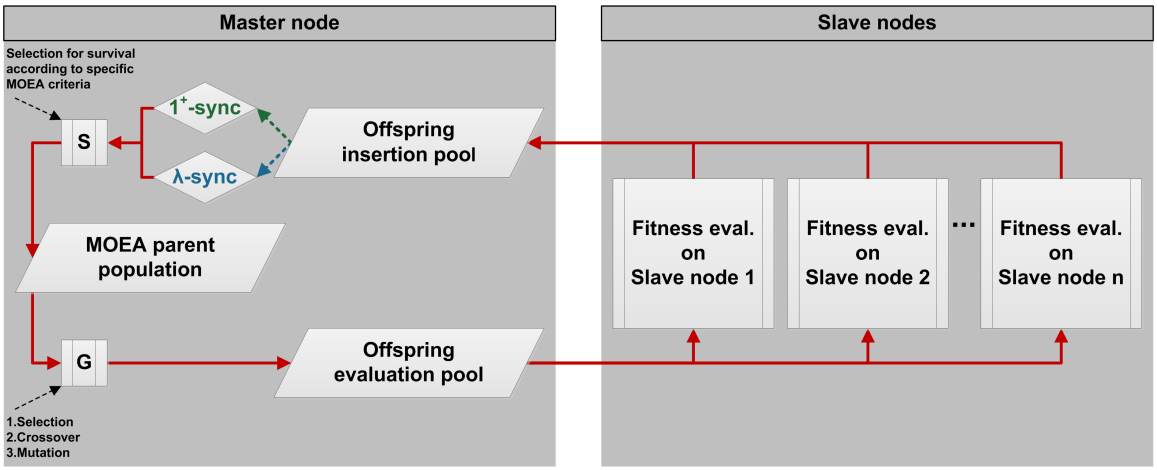


Figure 5.1: Diagram of the GEN-MSPS (the λ-sync block) and SSA-MSPS (the 1^+-sync block) computation cycles.

Figure 5.2 provides a sketch / didactic example of how individuals are processed by the two MSP schemata. The more flexible synchronization step of SSA-MSPS enables this method to evaluate more individuals per time interval than GEN-MSPS (e.g., in Figure 5.2 SSA-MSPS could evaluate 4 more offspring in the remaining time interval).

The downside of using SSA-MSPS is that, intuitively, the same lack of generational synchronization is expected to make SSA-MSPS achieve worse results in terms of *PN* quality after evaluating a fixed number of individuals. A quick example as to why this might happen is represented by the case of "Ind-9" from the aforementioned figure. In the case of GEN-MSPS, this individual is able to theoretically "profit" from the combined "search knowledge" of eight individuals that have been generated (and evaluated) before him. In the case of the SSA-MSPS, "Ind-9" can only profit from the "search knowledge" of six individuals as, because of the 1^+-sync block, "Ind-7" and "Ind-8" have not been evaluated by the time "Ind-9" was generated. Although rather trivial, these examples are very useful to describe the basic effect of the two different synchronization blocks:

- SSA-MSPS displays a **quantitative improvement** with regard to GEN-MSPS as the former can compute more individuals in a fixed time interval;
- SSA-MSPS displays a **qualitative deficit** with regard to GEN-MSPS as individual number s generated by SSA-MSPS profits from less "search knowledge" than individual number s generated by GEN-MSPS.

The main idea our MSP analysis is to study the quantitative and qualitative aspects in more detail in order to discover if there are any factors that can drastically influence the interaction between them.

We consider that performing our analysis from this dual point of view is very useful since quantitative aspects tend to be related more to the physical and software constraints of the available parallel/distributed computing architecture (i.e., they are more or less fixed), while the qualitative aspects are mainly dependent on the chosen MOEA, the chosen algorithm parameterization, and the complexity of the actual MOOP to be solved. Therefore, the qualitative aspects exhibit a higher variability as two of the factors that directly influence them (i.e., MOEA and MOEA parameterization) can be selected freely.

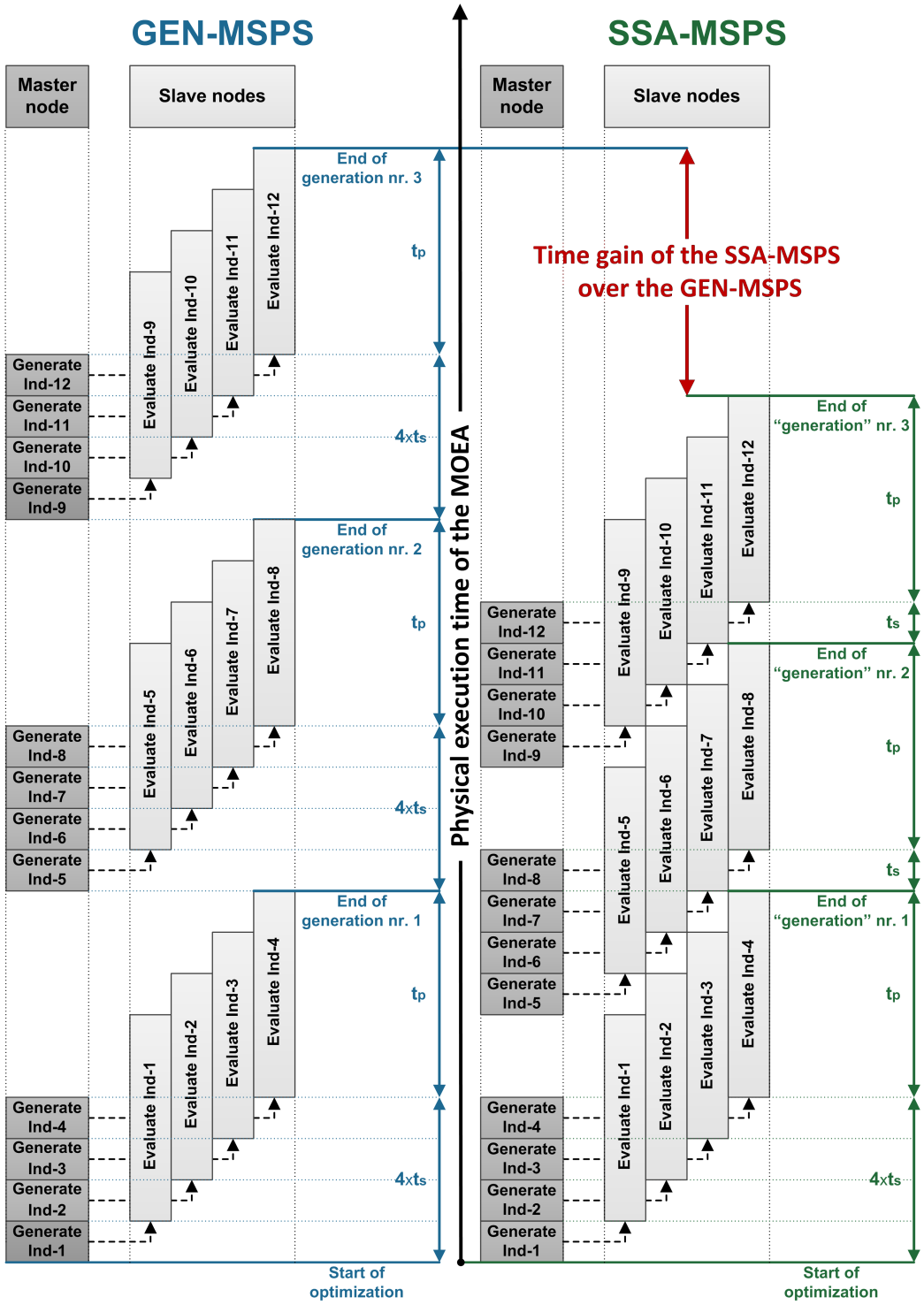


Figure 5.2: The comparative computation steps of GEN-MSPS and SSA-MSPS for 3 generations of size 4 in a distributed computing environment with one master node and 4 slave nodes.

5.2 The Quantitative Performance

5.2.1 The Basic Model

At first we attempt to derive a basic quantitative performance model that, given a time interval, is able to indicate how many more individuals is SSA-MSPS able to compute when comparing to GEN-MSPS.

We consider a generic $(\mu + \lambda))$-styled MOEA that is parallelized / distributed over a computing environment with more than λ slave nodes (i.e., we assume that the available number of slave nodes is not the bottleneck of the optimization setup). We mark with $t_p > 0$ the duration (in time units) of distributing and performing the fitness evaluation of any individual on any slave node (i.e., the duration of the remote computation tasks). We also mark with $t_s > 0$ the cumulative duration of the sequential computation tasks (i.e., genetic operations + possible pre-evaluation tasks) that are performed on the master node in order to create one individual. For the time being, we assume that t_s and t_p are constant (i.e., we have a homogeneous time-wise distribution of both the fitness evaluation and the individual creation functions). The *parallelization ratio* is defined as:

$$r = \lceil \frac{t_p}{t_s} \rceil \tag{5.1}$$

N.B. Under the above mentioned restrictions, when considering the GEN-MSPS approach, it is quite straightforward that it doesn't need more than $r+1$ slave nodes simultaneously as the first slave will finish its fitness evaluation by the time individual number $r+2$ is generated on the master node . The reasoning in this section is made under the restriction $r+1 \geq \lambda$ and that there are more that λ slave nodes available for performing the remote fitness evaluations.

Assuming that other miscellaneous computation times are negligible with regards to (or integrated in) t_s and t_p, the total time required to compute any generation of λ individuals using the GEN-MSPS is $(\lambda \times t_s) + t_p$. In case of the SSA-MSPS, the time required to compute the first λ individuals is also $(\lambda \times t_s) + t_p$, but the time required to compute any of the next batches of λ individuals is $(t_s + t_p)$, as sketched in Figure 5.2. Therefore, when wishing to compute *maxGen* generations, the overall computation time is

1. $(\lambda \times t_s + t_p) \times maxGen$ in the case of GEN-MSPS;
2. $(\lambda \times t_s + t_p) + (t_s + t_p) \times (maxGen - 1)$ in case of SSA-MSPS.

After equalizing these computation times and performing the necessary calculations, we have that in the time interval required by GEN-MSPS to compute $maxGen \times \lambda$ individuals, SSA-MSPS can compute $\Delta_{struct}\%$ more individuals, where Δ_{struct} is given by:

$$\Delta_{struct} = \frac{(maxGen - 1) \times (\lambda - 1) \times t_s}{maxGen \times (t_s + t_p)} \times 100 \tag{5.2}$$

We shall refer to 5.2 as the **structural improvement** that SSA-MSPS has over GEN-MSPS in terms of computed individuals per fixed time interval.

It is important to note that while Δ_{struct} does depend on the number of generations to be computed, the dominant factors that influence Δ_{struct} are the ratio between t_s and t_p (i.e., the parallelization ratio r) and the population size (i.e., λ). When dealing with CIMOOPs, a rather small choice of λ (i.e. 50 to 250) is usually the norm.

When fixing $\lambda = 100$, $maxGen = 500$, and $t_s = 1$, by varying the value of t_p, we can compute the dependency of Δ_{struct} on the parallelization ratio r. The corresponding values are presented in the left plot of Figure 5.3 as the *basic model curve*. Unsurprisingly, these values indicate that *the quantitative improvement associated with SSA-MSPS decreases exponentially with the parallelization ratio*.

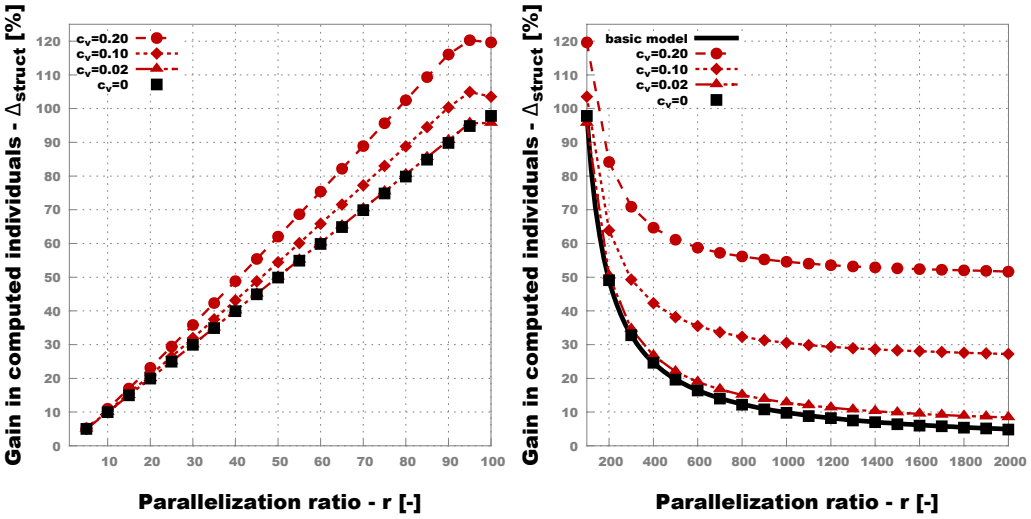


Figure 5.3: Δ_{struct} plots for different parallelization ratios and different degrees of variance (i.e. c_v) in the time-wise distribution of the fitness evaluation function

Although valuable in establishing a baseline for the comparison between GEN-MSPS and SSA-MSPS, this simplistic comparison has one severe limitation: it is strongly influenced by the idealistic assumption that the duration of the fitness evaluation tasks is constant.Therefore, in the next section we address this issue in order to improve our quantitative performance model.

5.2.2 The Effect of Variance

At first, we performed tests in order to validate the theoretical model proposed in the previous section. Using a homogeneous time-wise fitness distribution, we simulated the time required by GEN-MSPS and SSA-MSPS runs when considering various values of the parallelization ratio for the same settings used previously $\lambda = 100$, $maxGen = 500$, and $t_s = 1$. The obtained results (Figure 5.3 - the $c_v = 0$ data points for $r > 100$) confirm the Δ_{struct} behavior indicated by the theoretical model from (5.2). Furthermore, the simulation also allowed us to easily estimate Δ_{struct} for values of r smaller than λ (i.e., the left plot of Figure 5.3). In this case, Δ_{struct} displays a linear behavior that is directly proportional to r.

Table 5.1: The observed variance-specific lower thresholds of Δ_{struct}

c_v [-]	Lower threshold for Δ_{struct} [%]	Δ_{struct} for $r = 10^6$ [%]
0.20	48.9700	49.1040
0.10	24.4500	24.5300
0.05	12.1800	12.2140
0.02	4.8140	4.8300

Secondly, we wanted to quantify the influence of having ever larger degrees of heterogeneity (i.e. variance) in the time-wise distribution of the fitness evaluation function. Therefore, we performed new simulations where the fitness evaluation of each individual took t_p milliseconds and $t_p \sim \mathcal{N}(m, \sigma)$. By fixing $t_s = 1$, we obtain $r \sim m$ and. When scaling up m we also modified σ in order to keep the coefficient of variation, $c_v = \frac{\sigma}{m}$, constant at preset values. By using this simple technique we were able to effectively control both the amount of variation in the time-wise distribution of the fitness function and the parallelization ratio. The maximum amount of variance that we could consider under the normal distribution assumption was given by $c_v = 0.2$ - a higher value would result in sampling negative values (which doesn't make sense for a fitness duration). Because of the induced stochasticity, for each value of r we performed 100 tests and we report averaged results.

The plot in Figure 5.3 shows how Δ_{struct} behaves for four different variance levels (i.e., values of c_v). The curves clearly indicate that the exponential decrease of Δ_{struct} is dampened by increased levels of variance. Further experiments have also shown that for $(r > \lambda)$, when having variance in the time-wise fitness distribution function, after reaching a lower threshold, the value of Δ_{struct} tends to stabilize. We have run simulations up to $r = 10^6$ with a step size of 500. In Table 5.1, we report the lower thresholds of Δ_{struct} for different variance levels. We mention that, in the absence of variance, for $r > 49500$, $\Delta_{struct} = 0.0\%$ because, although SSA-MSPS computes the required 50000 individuals faster than GEN-MSPS, no extra individual can be computed by SSA-MSPS in the remaining time interval.

In conclusion, the theoretical model given in (5.2) provides an accurate lower limit for Δ_{struct} but the value of Δ_{struct}. Nevertheless, Δ_{struct} is significantly higher for a given parallelization ratio r when having variance in the time-wise distribution of the fitness function. Furthermore, in the presence of variance, Δ_{struct} is lower bounded by variance-specific thresholds that display a remarkable stability even at very high values of r.

5.3 The Qualitative Performance - Empirical Results

5.3.1 The Testing Framework

The qualitative performance of the two considered master-slave parallelization schemes is harder to quantify as it depends on the concrete MOOP to be solved, on the used MOEA and on the parameterization of the MOEA. In the following paragraphs we describe the details of the performance evaluation framework we propose in order to estimate the qualitative performance.

Firstly, we mention that we conducted our experiments (50000 *nfe* / optimization run) using NSGA-II and SPEA2 with standard parameterizations and a population / archive size of 100. The motivation for using NSGA-II is that this MOEA is widely used and experimented with, so the empirical results obtained with it should be of interest to a larger audience. The reasons for experimenting with SPEA2 are related to the fact that, in the next chapter, we use this MOEA as a subpart of a more complex MOO solver that is later also parallelized. Therefore, the performance of SPEA2 with regard to GEN-MSPS and SSA-MSPS is of considerable interest to us. Given the stochastic nature of MOEAs, for each comparative test that was performed, we made 100 repeats of each experiment (i.e., MOOP-MOEA run) and we always report over the averaged results.

Secondly, we carried out our tests on 15 benckmark MOOPs selected from those described in Section 3.1.1. The 15 selected problems propose different degrees of difficulty and, subsequently, different convergence behaviors for the two MOEAs that we experiment with. The computation of the fitness values for all 15 problems is very fast on any modern processor. In order to make the MOEAs exhibit the desired test behavior, the fitness computation times were artificially increased.

Thirdly, we must mention that we are particularly interested in studying the early and middle-stage convergence behavior of MOEAs when applying GEN-MSPS and SSA-MSPS. This is because in a real-life optimization scenario, that is time-constrained, a practitioner is likely to stop the optimization process as soon as he/she notices that small improvements come at an ever increasing computational cost (i.e. the MOEA enters the late-stage of convergence). We have arbitrarily defined the limits $0.15 \leq Ind_H \leq 0.85$ in order to define what is our region of interest with regard to MOEA convergence.

Fourthly, we introduce a new Ind_H-derived qualitative performance indicator that denotes the *SSA qualitative deficit* at the Ind_H *marker* p:

$$\Delta_{qual}(p) = \left(\frac{nfe_{SSA}(p)}{nfe_{GEN}(p)} - 1 \right) \times 100,\ p \in (0,1] \tag{5.3}$$

where, for a fixed MOOP and a fixed MOEA, $nfe_{SSA}(p)$ computes the number of fitness evaluations that must be performed when applying SSA-MSPS in order to reach a current *PN* with a Ind_H-measured quality of at least p. $nfe_{GEN}(p)$ computes the complementary value for GEN-MSPS.

5.3.2 Basic Qualitative Performance

In the first series of performed tests, using a constant fitness distribution (i.e. $c_v = 0$), we computed the Ind_H-estimated quality of the MOEA parent population consecutive batches of 100 individuals. The results obtained with NSGA-II are presented in the left subplots [marked with (a)] from Figures 5.4, 5.5 and 5.6. The results obtained with SPEA2 are presented in the left subplots [marked with (a)] from Figures 5.4 5.5 and 5.6.

We consider that a more useful perspective for presenting the same comparative convergence behavior information can be constructed by plotting the Δ_{qual} values of the MOEA parent population as shown (the $c_v = 0$ lines) in the subplots marked with (b) from the previously mentioned figures.

For the sake of brevity, our next observations are made based on the empirical results obtained with NSGA-II. The results obtained with SPEA2 largely display the same trend, a fact than can be confirmed by the presented graphical and tabular data.

The first important observation is that for 10 of the 15 test problems, more precisely for those presented in Figures 5.4 5.5, NSGA-II displays a good convergence behavior as it bypasses the initial and middle stages of convergence and is able to reach the final stage of convergence (i.e., $p > 0.85$, where p **denotes a** Ind_H **marker**). On the 5 MOOPs presented in Figure 5.6, the convergence behavior is different:

- for DTLZ3 - the MOEA is only able to reach p values of ≈ 0.40 after 50000 fitness evaluations (continuing the run would eventually enable it to reach a late stage of convergence);
- for LZ09-F1, LZ09-F8, and LZ09-F9 - the MOEA seems unable (with the given settings) to reach p values > 0.85 and the charts indicate premature convergence;
- for WFG7 - the MOEA is able to reach the late stage of convergence, but random initializations of the initial population already display relative p values of ≈ 0.40;

For the remaining of this work we shall refer to the 10 MOOPs from Figure 5.4 and Figure 5.5 as the **successfully solved problems** and to the 5 problems from Figure 5.6 as the **special case problems**.

The results of these initial tests confirm some of the findings from [142], in the sense that, the GEN-MSPS is able to achieve a higher quality Pareto front than SSA-MSPS after the same number of evolved individuals in the early and middle stages of convergence for all the 10 successfully solved problems. This observation also holds for 4 of the special case problems. In the case of LZ09-F8, the NSGA-II convergence graphs indicate that SSA-MSPS has a better performance when wanting to reach p-values > 0.30.

Furthermore, when abstracting the behavioral shifts and numeric artifacts that characterize the early and late stages of convergence, we notice that Δ_{qual} values are quite constant (within a 10% range) for each successfully solved test problem. When also considering the special case problems, we can state that, **in general,** Δ_{qual} **values do not display a trend that increases with** p.

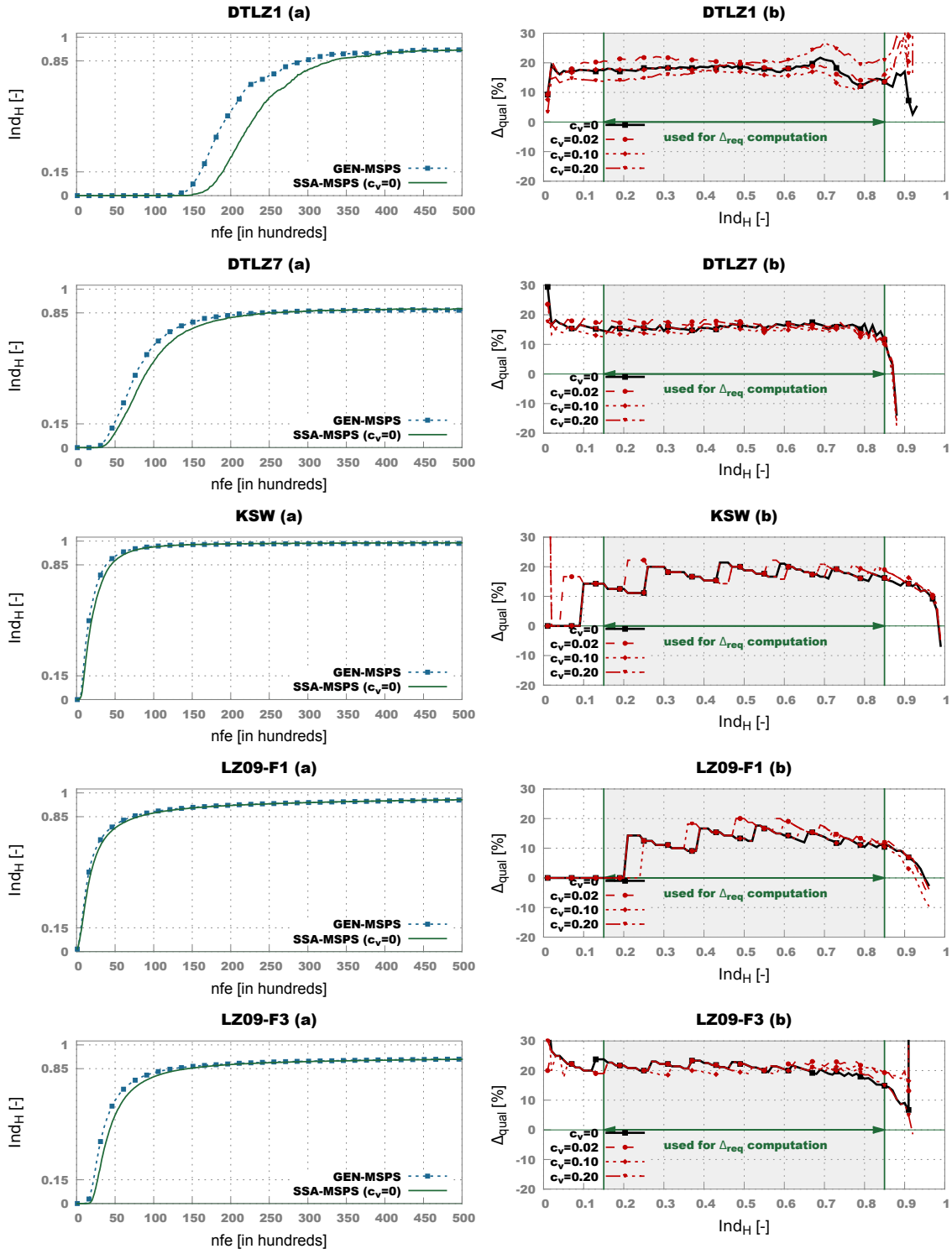


Figure 5.4: NSGA-II qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

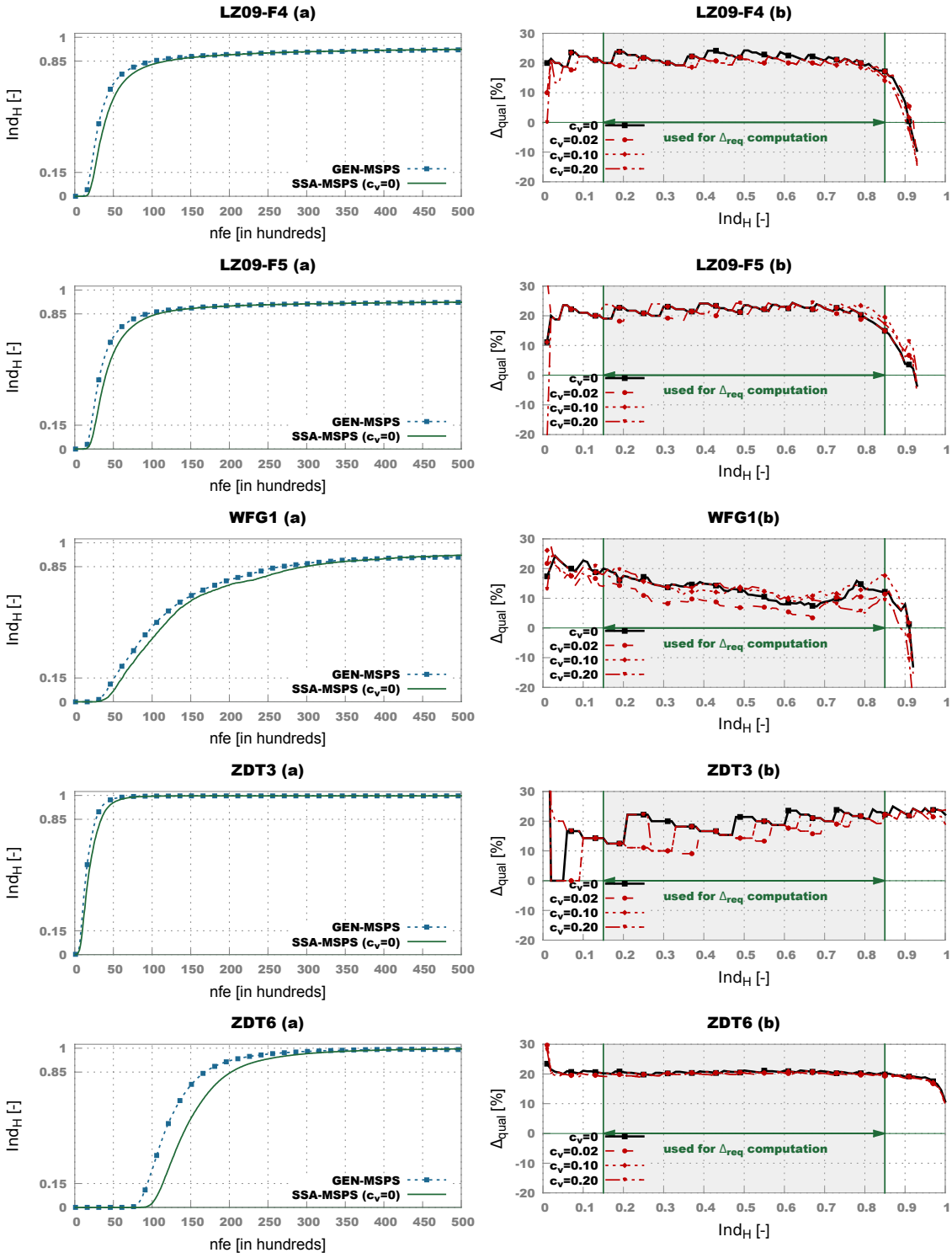


Figure 5.5: NSGA-II qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

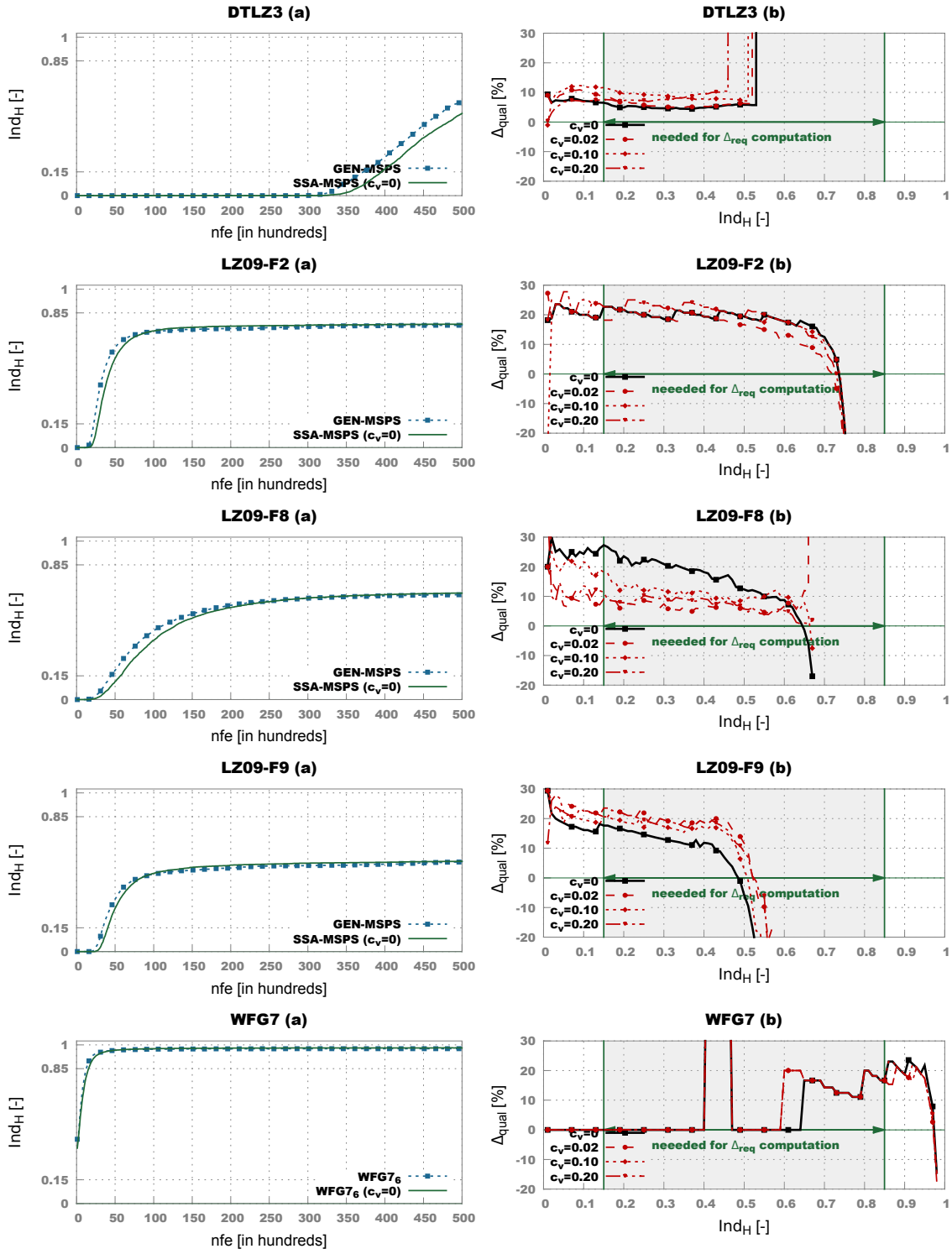


Figure 5.6: NSGA-II qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

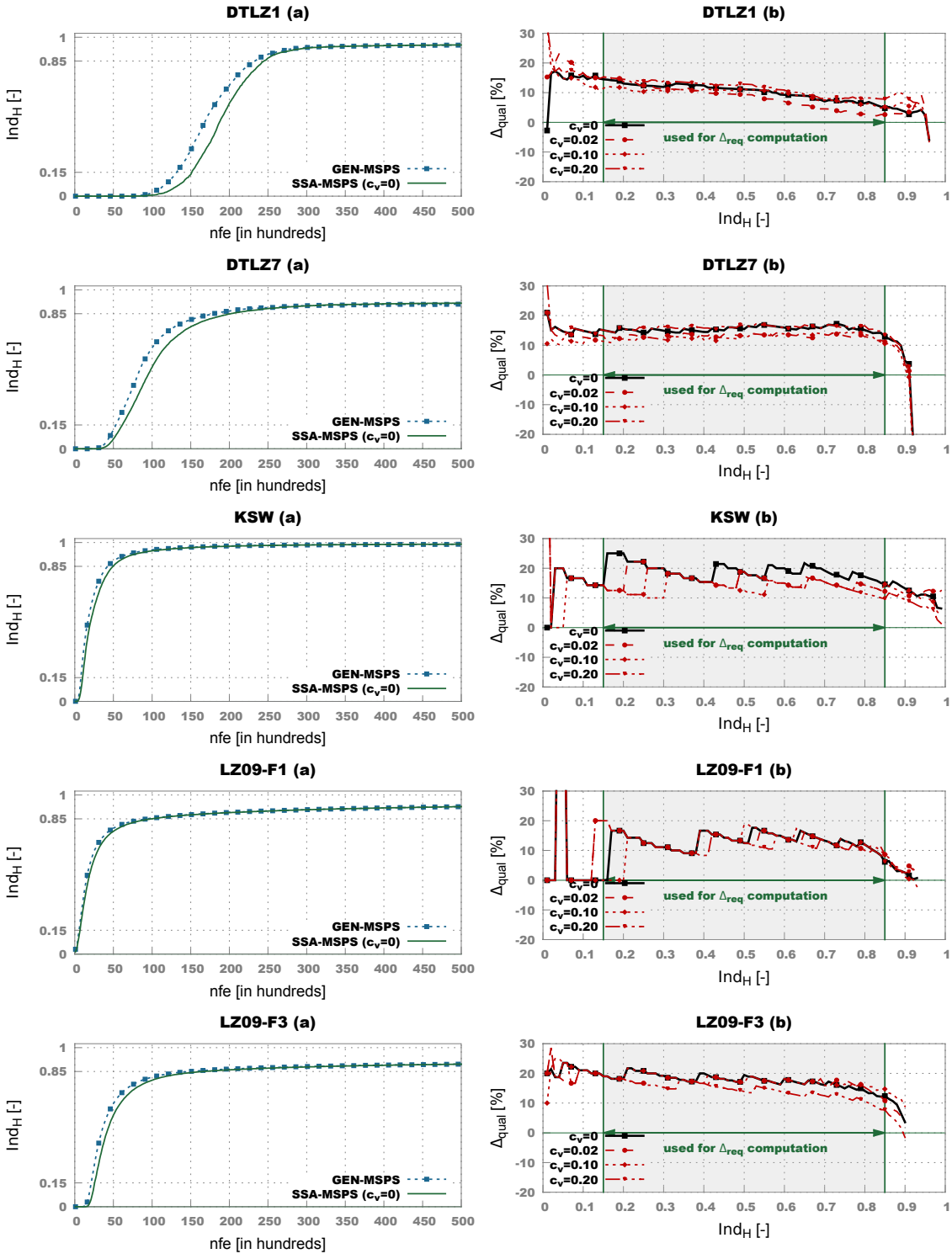


Figure 5.7: SPEA2 qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

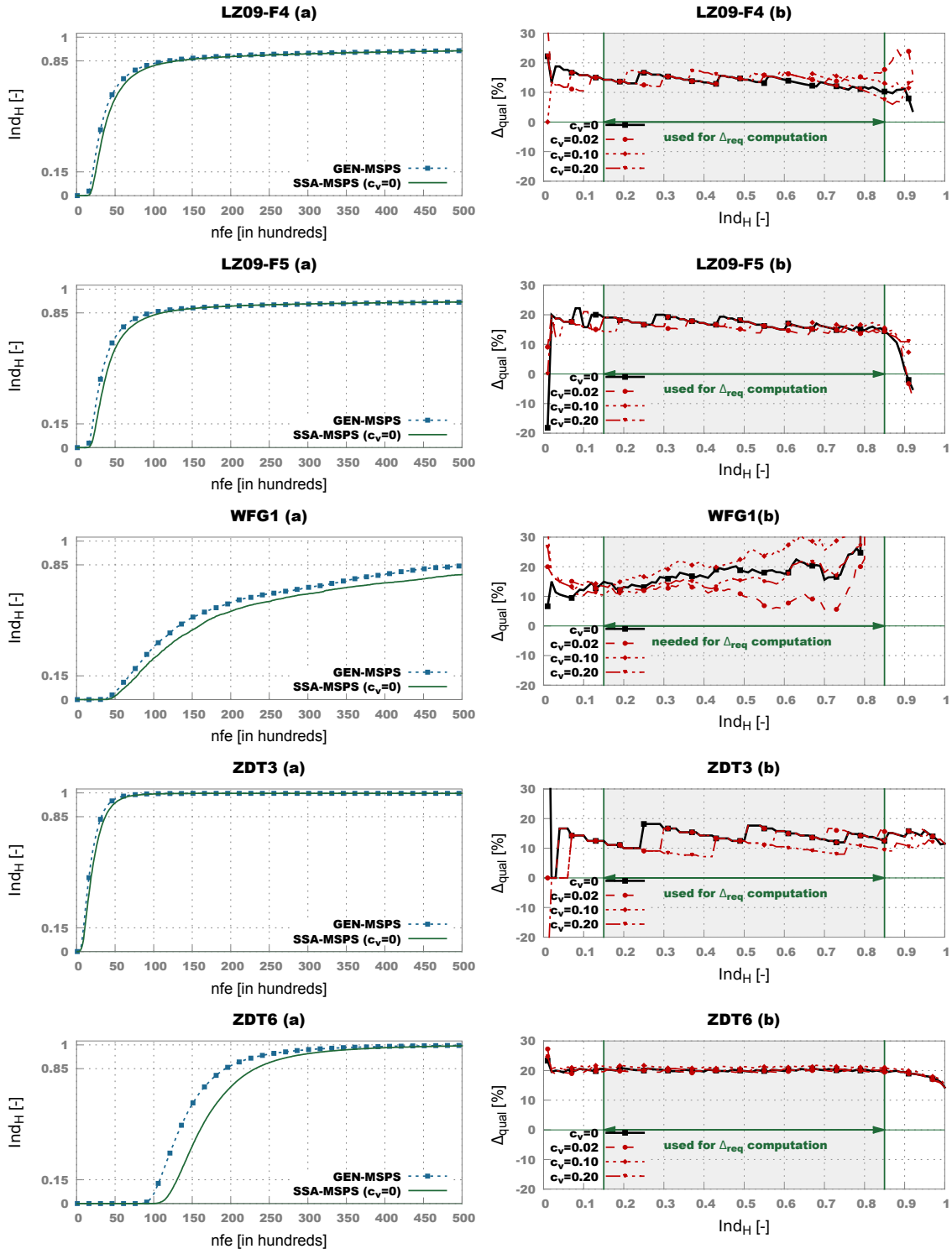


Figure 5.8: SPEA2 qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

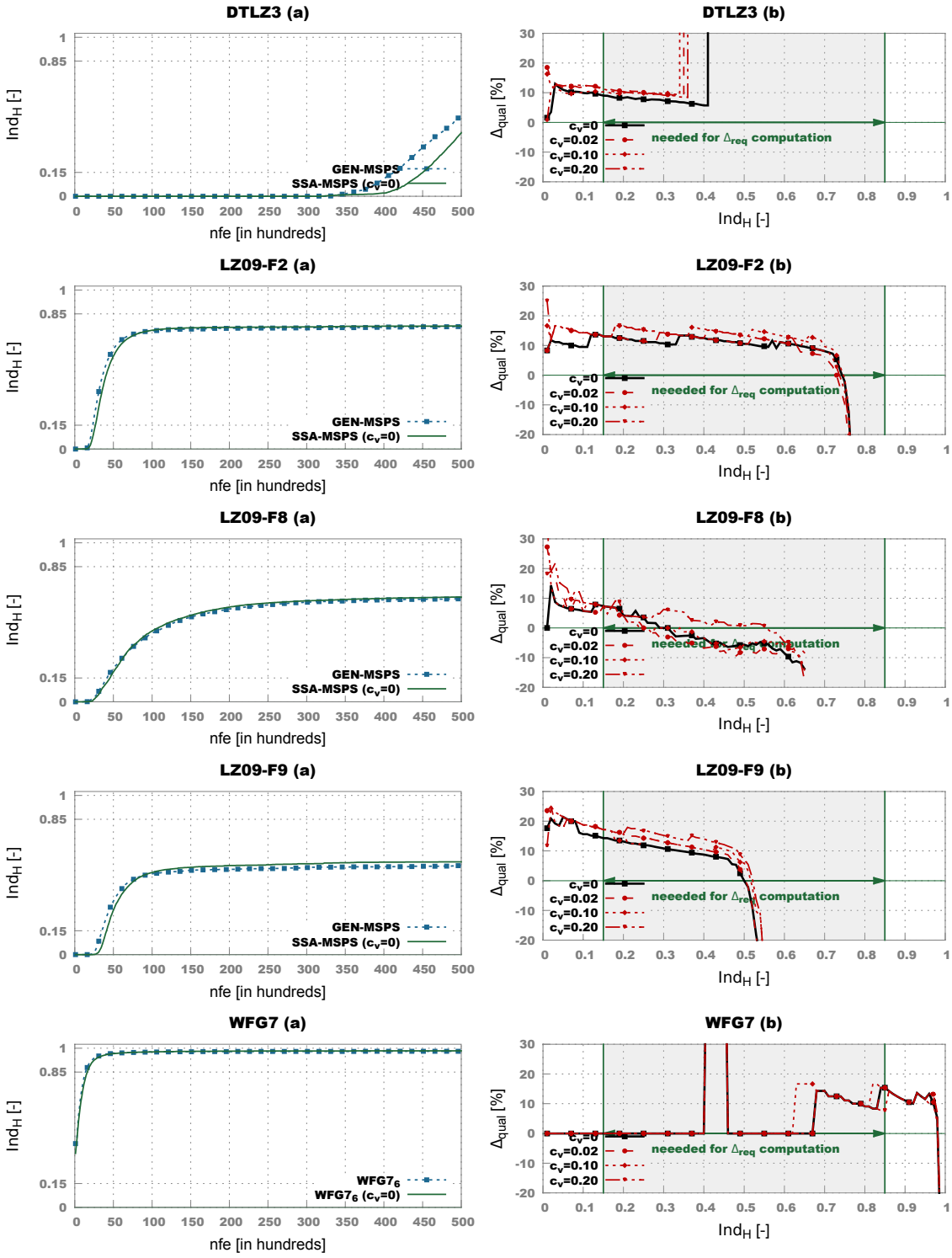


Figure 5.9: SPEA2 qualitative performance plots: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

Considering the somewhat constant behavior of Δ_{qual} for the successfully solved MOOPs, we constructed an additional quality metric. By averaging the individual Δ_{qual} values associated with the middle stage of convergence (i.e., $p =\in \{0.16, 0.17, \ldots 0.85\}$), we obtain the **average required SSA improvement** for a MOOP-MOEA combination:

$$\Delta_{req} = \frac{1}{70} \sum_{p=16}^{85} \Delta_{qual}(p) \tag{5.4}$$

The new Δ_{req} metric is important as it is a rough indicator of how many more individuals per time interval SSA-MSPS must compute in order to mach the results that would be produced by GEN-MSPS in the same time interval.

5.3.3 The Effect of Variance

The second series of tests that we have performed in order to gain more insight into the qualitative performance of the two parallelization schemes is again related to the influence of having variance in the time-wise fitness distribution function.

The results obtained using NSGA-II and SPEA2 are also presented in the subplots marked with (b) from Figures 5.4, 5.5, 5.6, 5.7, Figure 5.8 and 5.9. A quick look over all 30 plots reveals that the effect of variance is not so important in the case of the qualitative performance.

The values of Δ_{req} for the successfully solved problems for both NSGA-II and SPEA2 are shown in Table 5.2. Given the formulation of Δ_{req} from 5.4, the metric can not be computed for the special case MOOPs. We point out that when using SPEA2, WFG1 becomes a special case problem and, as such, computing Δ_{req} values for the SPEA2-WFG1 combination does not make sense. Nevertheless, the data from Table 5.2 clearly indicates that, in the case of the qualitative performance, variance in the time-wise distribution of the fitness evaluation function has a negligible effect as:

- Δ_{req} is not directly proportional to the amount of variance and for 7 out of the 19 MOOP-MOEA combinations, the highest average Δ_{req} value corresponds to the experiment with zero variance (i.e., $c_v = 0$);
- in 13 out of 19 cases, the observed average changes induced on Δ_{req} by having some level of variance are not statistically significant. More precisely, we checked if the difference between the highest and the lowest Δ_{req} values for any MOOP-MOEA combination is statistically significant given a one-sided Mann-Whitney-Wilcoxon test with a considered significance level of 0.05.

All these observations create a stark contrast when comparing with the powerful effect that variance has on the quantitative performance and provide a solid indicator that **SSA-MSPS should be favored in the presence of significant variance in the time-wise distribution of the fitness function**.

Table 5.2: Averaged values of the Δ_{req} metric over 50 runs for different levels of variance in the time-wise distribution of the fitness evaluation function. For each MOOP-MOEA combination, the highest value is highlighted and marked with "$^+$" if the difference between it and the lowest Δ_{req} value of the combination is statistically significant.

	Δ_{req} for NSGA-II at $c_v = [\%]$				Δ_{req} for SPEA2 at $c_v = [\%]$			
Problem	0	0.02	0.10	0.20	0	0.02	0.10	0.20
DTLZ1	17.61	18.94	16.28	19.30	10.17	8.20	9.48	11.40
DTLZ7	15.61	16.54^+	14.32	15.25	15.41	13.12	12.90	15.96^+
KSW10	17.22	18.35	17.33	17.19	18.95^+	15.94	13.84	14.77
LZ09-F1	12.36	13.81	11.80	14.03	13.25	13.02	12.63	11.93
LZ09-F3	20.45	21.67	20.35	20.82	17.42	17.11	17.54	14.19
LZ09-F4	21.54	20.24	20.24	19.84	13.59	14.42	14.47	14.38
LZ09-F5	21.73	20.62	22.20	22.37	16.92	15.54	16.51	16.44
WFG1	12.58	8.77	13.27	12.14	-	-	-	-
ZDT3	19.92^+	15.29	16.67	17.99	14.33^+	14.07	13.11	9.90
ZDT6	20.53	19.98	20.25	19.96	20.11	19.78	21.19^+	20.23

5.4 Remarks regarding MSP Schemata

Across all 15 test problems and regardless of the induced variance, our results indicate that **SSA-MSPS performs quite similar to GEN-MSPS towards the end of the runs**. The average performance of SSA-MSPS is even marginally better (i.e., $\Delta_{qual} < 0$) for several problems (notably: DTLZ7, LZ09-F4, LZ09-F2, LZ09-F9, and WFG7). In particular, for the 10 successfully solved MOOPs, this means that it makes no difference which parallelization methods is applied if the optimization is ran long enough to allow GEN-MSPS to enter the late stage of convergence (i.e., $p > 0.85$).

This identical performance exhibited towards the end of the runs and the generally stable behavior of the qualitative deficit exhibited by SSA-MSPS allows for the following reasoning regarding the comparative performance of GEN-MSPS and SSA-MSPS: **if, for a given optimization scenario, the quantitative improvement of SSA-MSPS (Δ_{struct}) can overcompensate the qualitative deficit of SSA-MSPS (Δ_{req}), we can say that, on average, SSA-MSPS is the better parallelization choice.** This is because, when ($\Delta_{struct} > \Delta_{req}$), the percentage of extra individuals that *can* be evaluated when using SSA-MSPS (i.e., Δ_{struct}) is larger than the percentage of extra individuals that *must* be evaluated (i.e., Δ_{req}) in order to reach Pareto non-dominated sets of similar quality.

5.5 Application on a CIMOOOP

We have applied the previously described qualitative and quantitative analyses on a slightly modified version of IndMOOP1.

Based on 10^5 samples, the average duration of the fitness evaluations (i.e., t_p)tasks performed in our high throughput computing environment is 391.94 seconds and the average duration of the sequential computation tasks (i.e., t_s) is 21.48 seconds. This leads to a rather small parallelization ratio of only 18.25. The reason for the rather high value and rather strange distribution (Figure 5.10 - left plot) of t_s lays with the previously mentioned software licensing restrictions. The distribution of t_p (Figure 5.10 - right plot) is also quite heterogeneous ($c_v = 0.23$) as the cluster computers used to perform the FE simulations have different processing performances. After integrating all this data into the simulation framework described in Section 5.2, the results indicate that, on average, we should expect a Δ_{struct} of 19.47%. Taking into account that in all the tests from Section 5.3, Δ_{req} is hardly ever higher than 20%, this observation would motivate us to apply SSA-MSPS for this problem.

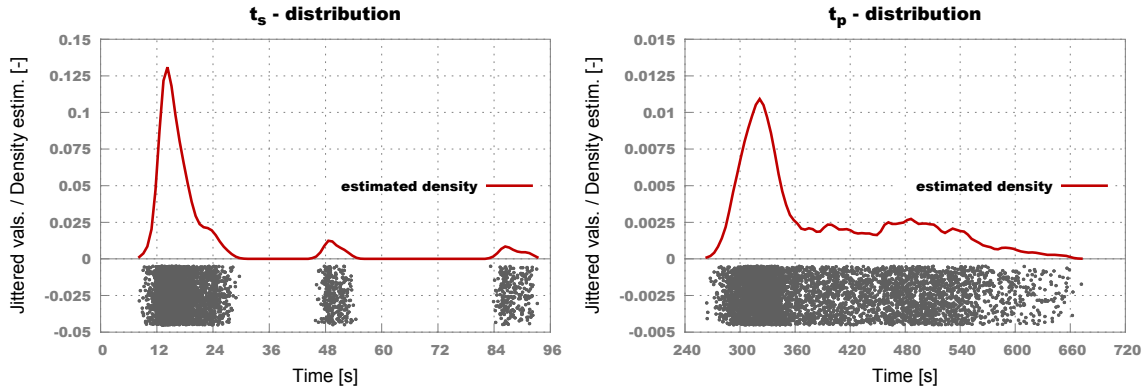


Figure 5.10: Kernel density estimations of the time-wise distributions of the sequential (t_s) and of the remote (t_p) computation tasks for IndMoop1

We performed 15 SPEA2 optimization runs with each MSP schema. The algorithm was parameterized using the same settings described in Section 5.3.1. Over these real-life runs, we measured a Δ_{struct} of 20.13% that is quite close to the prediction (19.47%). The qualitative performance plots are presented in Figure 5.11 and, surprisingly, they show that, after evaluating the same number of individuals, SSA-MSPS is able to produce better Pareto fronts than GEN-MSPS. The reason for this may lay with the number of infeasible designs generated during the search.

It is worth mentioning that the actual global run-times of the 30 real-life optimization runs confirm the computed Δ_{struct} and Δ_{req} values as the SSA-MSPS runs are (on average) able to achieve the *PN*-quality obtained with GEN-MSPS $\approx 25\%$ faster.

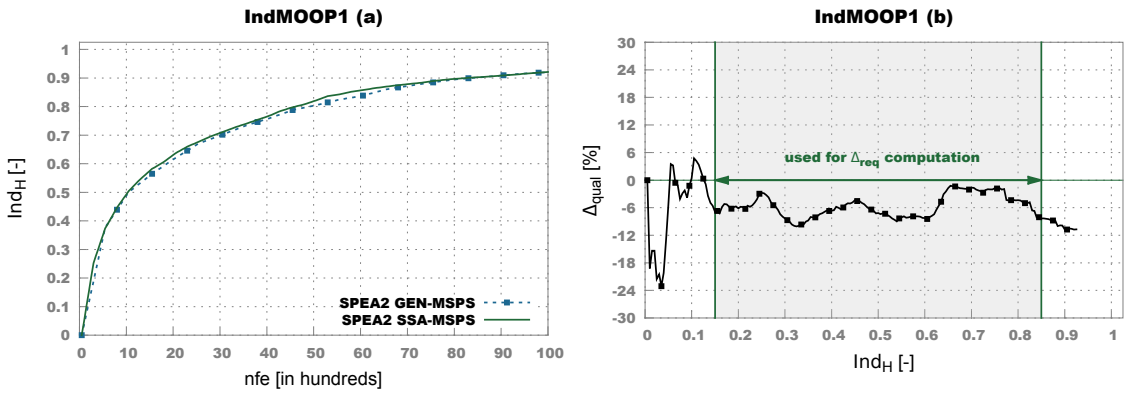


Figure 5.11: SPEA2 qualitative performance plots for IndMOOP1: (a) - generation-wise hypervolume performance, (b) - Δ_{qual} results

"In varietate concordia" / "In varietate unitas!"

Motto of the European Union / Ernesto Teodoro Moneta

Chapter 6

Coevolutionary Enhancements

6.1 Motivation and General Idea

Apart from Holland's Schema Theorem mentioned at the end of Section 2.3.1, one of the most influential / important theoretical results related to evolutionary optimization (and search / optimization in general) can be found in the 1997 article of Wolpert and Macready [103] in which the authors prove their now famous "**No Free Lunch Theorems for Optimization**" (NFL). In the own words of Wolpert and Macready from [143], "*the 'No Free Lunch' (NFL) theorems state that any two [search / optimization] algorithms are equivalent when their performance is averaged across all possible problems*". In order to clarify the disheartening (and often wrong) conclusions associated with the NFL theorems over the years, in [144], Wolpert clarifies that: "*while the NFL theorems have strong implications if one believes in a uniform distribution over optimization problems, in no sense should they be interpreted as advocating such a distribution*".

A very popular interpretation of these theorems is that, from a theoretical perspective, it generally makes sense to pair specialized optimization strategies with problems-to-be-solved as there exists no universal optimization strategy that is better[1] than all other strategies across all possible problems. Practically, given a real-life optimization problem – P1 – with an unknown solution and the demand to solve it using as few objective evaluations as possible, the NFL theorems suggest that, since a best-for-all-problems universal solver does not exist, one should focus on (finding and adapting / creating and) experimenting with (increasingly better) problem-specialized solvers that can speculate the particularities of P1 during the optimization. In light of this, the NFL theorems have been added to the select circle of what scientists humorously dubbed "*full employment theorems*"[2]. N.B. The NFL theorems do by no means dismiss the fact that a certain optimization algorithm –Alg-A– can perform *better* than another one – Alg-B – (or to all other ones for that matter) across a subset of optimization problems. The NFL theorems simply entail that if this superior performance

[1] Requires less objective evaluations to find the solution.

[2] Applying oneself to the study of problems governed by these theorems is likely to lead to a prolific publishing record or, like in the present case, to a few interesting results that can hopefully justify a PhD degree.

of Alg-A is confirmed (observed), then, there also exists another subset of optimization problems on which Alg-A performs correspondingly worse than Alg-B[3]. Nevertheless, if the set of problems on which Alg-A outperforms its competitors is of major practical importance and the set on which it underperforms is not (yet), then searching for / developing Alg-A is indeed worthwhile. For interesting theoretical arguments as to why informed search methods (hill climbing, simulated annealing, EAs, etc.) perform better than random search in nearly all reported cases please consult [146].

With regard to (general) optimization strategies that can be parametrized (like EAs), since different parameter settings can notably influence the concrete search behavior, the NFL theorems also provide quite compelling evidence in favor of (the empirically proven) need to (fine) tune[4] solvers and thus "specialize" them on the particularities of the problem(s) at hand when wanting to considerably improve the (*nfe*-measured) efficiency of the optimization process. Of course, most of the solving strategies regarded as being highly competitive also have the inherent quality of being quite robust with regard to their parameterization – i.e., when using standard (literature recommended) settings, they display a good balance between optimization efficiency and final solution quality across a wide range of problems.

Nevertheless, all the previous arguments are necessary in order to outline a very important practical aspect related to the inherent difficulty of solving CIMOOPs that is not usually emphasized: **the very long (wall-clock) optimization times associated with these types of problems generally make any attempts to parameter-tune an a posteriori MOOA rather impractical**. Thus, practitioners usually must rely on the standard parameter settings of the applied MOOA. when wanting to solve CIMOOPs. Furthermore, in our case, taking into account that several MOO practitioners and DMs from the electrical drive design community do not (and need not!) have an extensive background with regard to the internal workings and parameterization sensibilities of MOO solvers, **developing new MOOAs that are both highly efficient and very robust with regard to parameterization has been identified as an important research goal**.

In order to achieve it, we examined (and experimented with) a few already proven concepts that aim to enhance a standard EA process by:

- allowing the algorithm to apply different genetic operators (and parameterizations of these operators) throughout the search [147];
- enabling the EA to dynamically choose which genetic operators to apply based on previous performance [148];
- applying self-adaptive genetic operators [149];
- implementing a coevolutionary-centered search strategy;

In our case, **the coevolutionary option was able to deliver the most promising results** even when encapsulated in a very basic construct. Therefore, in Section 6.2 we describe our

[3]For example, in [145], Oltean describes a NFL-motivated approach for constructing artificial optimization problems (i.e., objective functions) on which random search outperforms EAs.

[4]In the case of an EA this can mean experimenting with different population sizes, different genetic operators, different parameterizations related to genetic operators and / or their application ratios, etc.

initial (and fairly successful) coevolutionary MOEA proposal and in Section 6.3 we describe DECMO2, a more advanced coevolutionary-centered approach that has proven very efficient and robust when considering all the artificial MOOPs mentioned in Section 3.1.1.

In nature, coevolution denotes a process through which two or more species that find themselves inside a predatory, parasitic, or symbiotic relationship are gradually forced to adapt (i.e., evolve) in virtue of their interactions. As such, coevolution can be seen as the part (or flavor) of the general evolutionary process that is responsible for (over-)specializing a population in response to the biotic dynamics of the environment it operates in. The role of this over-specialization is to allow a species to survive (in the predatory or parasitic case) or to thrive (in the symbiotic case). For an introduction and overview of biological coevolution please consult [150].

Because of the generality of the underlaying principal on which it is based (i.e., an action and reaction cycle that triggers inner-changes in the participants), coevolution has also been adapted and used outside biology in areas like astronomy, social sciences (economics, politics, sociology, etc.), and computer science.

Within computer science, among other uses, coevolution is a very popular choice for improving the performance of population-based (heuristic) optimization methods. Two main classes of subpopulation-based coevolutionary optimization algorithms are easily distinguishable:

1. *competitive coevolution* approaches - in which the fitness of an individual is the result of (adversarial) "encounters" (comparisons) with other individuals (possibly from another population) [151] [152];

2. *cooperative coevolution* approaches - in which the fitness of an individual from one population is the result of collaboration with individuals from other populations [153].

Practical overviews of coevolutionary optimization and MOO approaches can be found in Chapter 3 (more precisely, Section 3.5) from [47] and Chapter 6 from [154]. As a side note, the latter reference is very interesting as Sean Luke explicitly states (the quite common conception) that fitness sharing and crowding strategies "*are coevolutionary in nature*" since they assess the survivability of individual **x** based on the potential presence in the population of another individual **y** that is deemed as "too close" to **x** in objective space. Under this reasoning, all MOEAs from Section 2.3 that use a Goldberg-inspired selection for survival strategy, viz. NSGA-II, SPEA2, DEMO and GDE3, can be seen as coevolutionary approaches.

Apart from the many (real-life) optimization contexts where coevolutionary approaches have proven their worth over the years, our attention towards these methods was also drawn by another interesting theoretical result reported by Wolpert and Macready: the NFL theorems do not generally hold in particular cases of coevolutionary optimization that involve "'*self play' problems*" [143]. In these types of problems, coevolution is used to enable a set of given individuals (game strategies) to "'*cooperate' to train one of them as champion*"[5]. The "champion" strategy will then be pitted against an "antagonist" in a

[5]This type of interaction is actually classified as *1-population competitive coevolution* by Sean Luke [154]

subsequent multiplayer (turn-based) game. The goals is to obtain a champion that performs as well as possible in the subsequent game. Wolpert and Macready indicate as concrete self-play application domains (where their new results can be of interest) checkers and chess and provide two references to this respect: [155] and [156]. Although our application domain (i.e.,MOO) does not resemble the ones investigated in [143][6], after lecturing this work, we found very enticing the idea of trying to construct a "champion" MOEA that can efficiently combine some of the most important MOEA-related developments that have been proposed over the years. The self-play application domain and the turn-based games it concerns also inspired the design of the HRPC methodology we introduced in Section 3.3.2. To complete the analogy, the main goal of our "champion" MOEA is to perform better than other state-of-the-art "antagonist" MOEAs during (especially the initial stages of) the multi-stage Ind_H-comparison races that the HRPCs illustrate. This is because, by virtue of design, **HRPCs favor algorithms that display a better than average performance across as many of the considered test problems as possible**. A MOEA that fairs very well with regard to HRPCs when using a fixed parameterization over a large (diverse and complicated) problem set is very likely to be a top pick for a DM in search for a black-box a posteriori MOO method.

The starting point for building our "champion" algorithm was the simple observation (hinted in Section 2.3.4). that the only differences between the generational SPEA2 algorithm [66] and the generational GDE3 algorithm [73] lies in the choice of genetic operators. Nevertheless, the (exploratory) performance of these two MOEAs on several MOOPs is quite different. Given a new CIMOOP to be solved, the MOEA practitioner (i.e., electrical drive DMs, in our case) would of course prefer to use the best performing of the two options. However, if no extra information related to the problem can help him decide between the two, (s)he will eventually be forced to make a (more or less educated) guess. Our initial idea for solving this dilemma was to try and develop a simple hybrid approach that essentially:

- runs both algorithms in parallel (i.e., *coevolves* two distinct subpopulations);
- is able to profit from the clear superior performance that might be exhibited by one of them (i.e., *stores and shares* elite solutions among the subpopulations).

6.2 Initial Approach: The DECMO Algorithm

6.2.1 Method Description

Let us mark with P the first subpopulation of our coevolutionary MOEA and with Q the second subpopulation. Both subpopulations are of equal size: $p_{size} = |P| = q_{size} = |Q|$. The final PN of our **differential evolution-based coevolutionary multi-objective optimization algorithm** (**DECMO**) is obtained by applying the environmental selection operator on the union of the two subpopulations: $PN_{\text{DECMO}} = E_{sel}(P \cup Q, p_{size} + q_{size})$.

Subpopulation P is evolved according to the (generational) SPEA2 evolutionary model described in Section 2.3.3. This means that, at a given generation $t, t \geq 1$, we apply the

[6] And as such we make absolutely no claims that the findings reported there apply in our case.

SBX and PM genetic operators on the individuals in P in order to obtain an offspring subpopulation P'. The SPEA2 subpopulation of generation $t+1$ is obtained by applying the environmental selection operator (also described in Section 2.3.3) on the union of parents and offspring: $P = E_{sel}(P \cup P', p_{size})$.

Subpopulation Q is evolved according to a DEMO/GD3 evolutionary model that, as mentioned in Section 2.3.4, retains the environmental selection operator, but adopts a DE-based search space exploration. The particular DE strategy (i.e., operator combination) suggested by DEMO/GDE3 is *DE/rand/1/bin*. At a given generation $t, t \geq 1$, we begin by performing the initializations: $Q' \leftarrow \Phi$ and $Q'' \leftarrow Q$. Afterwards, the DE-based evolutionary model requires that, as long as $Q'' \neq \Phi$, we randomly extract $\mathbf{x} \in Q''$ and perform three operations:

1. Construct a mutant vector $\mathbf{v}$ according to the *rand/1* part of the DE strategy by randomly selecting three individuals $\mathbf{z^a}, \mathbf{z^b}, \mathbf{z^c} \in Q$ such that $\mathbf{z^a} \neq \mathbf{z^b} \neq \mathbf{z^c} \neq \mathbf{x}$ and computing:
$$\mathbf{v} = \mathbf{z^a} + F(\mathbf{z^b} - \mathbf{z^c}) \tag{6.1}$$
where $F > 0$ is a numeric control parameter.

2. Generate a trial (offspring) vector $\mathbf{y}$ via the *bin*omial crossover part of the DE strategy:
$$y_i = \begin{cases} v_i & \text{if } u^i < CR \text{ or } i = j \\ x_i & \text{if } u^i \geq CR \text{ and } i \neq j \end{cases}, \tag{6.2}$$
where j is a randomly selected integer from $\{1, \ldots, n\}$, $u^1, \ldots, u^n$ are independent random variable uniformly distributed in $[0,1]$, and $CR \in [0,1]$ is a numeric control parameter.

3. Remove $\mathbf{x}$ from the list of individuals from the current generation that must be processed – i.e., $Q'' = Q'' \setminus \{\mathbf{x}\}$ – and update Q' using the formula:
$$Q' = \begin{cases} Q' \cup \{\mathbf{x}\} & \text{if } \mathbf{x} \preceq \mathbf{y} \\ Q' \cup \{\mathbf{y}\} & \text{if } \mathbf{y} \preceq \mathbf{x} \\ Q' \cup \{\mathbf{x}\} \cup \{\mathbf{y}\} & \text{if } \mathbf{x} \not\preceq \mathbf{y} \text{ and } \mathbf{y} \not\preceq \mathbf{x} \end{cases} \tag{6.3}$$

At the end of the cycle that processes every element of Q'', usually, $|Q'| > q_{size}$. In order for this to happen, it is sufficient to have a single occurrence where $\mathbf{x}$ and its associate (offspring) trial vector $\mathbf{y}$ are not dominating each other and thus, according to (6.3), both individuals are added to Q'. As we operate under the assumption of a fixed (sub)population size, in order to select the individuals that will form the subpopulation of the next generation, the DEMO[7]/GDE3 evolutionary model requires the application of the environmental selection operator: $Q = E_{sel}(Q', q_{size})$.

[7]We are referring to the DEMO$^{\text{SP2}}$ variant from [72]. The DEMO$^{\text{NS-II}}$ variant presented in the same paper uses the non-dominated sorting mechanism proposed by NSGA-II.

In order to construct an efficient cooperative coevolutionary process, one of the most important tasks is to design a good *fitness sharing* mechanism. After experimenting with several options, we found that a dual fitness sharing mechanism was able to ensure a very stable and competitive performance. This mechanism consists in:

1. *weak sharing stages* – that occur at every new generation and consist in *attempt*ing to insert in each subpopulation one random individual from the complementary subpopulation;

2. *strong sharing stages* – that occur once every t_{sha} generations and require the construction of an elite subset of individuals from $P \cup Q$ and the *attempt* to (re)insert the members of this elite subset in both subpopulations. We mark the size of this elite subset with e_{size}. The idea behind this process is to spread, from time to time, the best performing individuals across both coevolved subpopulations.

The word "attempt" is used in order to emphasize the fact that we are not enforcing the described operations in order to avoid the brutal altering of the inherent dynamics of the coevolved populations. Instead, we always rely on the implicit (highly elitist) selection for survival mechanism that is employed by both the SPEA2 and DEMO/GDE3 evolutionary models. This extensive use of the environmental selection operator throughout DECMO is also motivated by the fact that it inherently has two features that are extremely useful from a coevolutionary perspective:

1. a primary Pareto-based selection criterion – that can successfully act as a global (i.e., intra-population) fitness indicator;

2. a secondary crowding-based selection (filtering) criterion – that ensures diversity (niching) within the elite subsets that are a key part of the fitness sharing strategy.

N.B. Any MOO selection strategy based on the principles laid down by David Goldberg in [50] would also display the two features mentioned above. For example, instead of the environmental selection operator, one can also employ the non-dominated sorting strategy proposed by Deb et al. in [65]. The choice of using the former in DECMO is a matter of personal preference.

In Algorithm 3 we present the major steps of our proposed coevolutionary approach. We outline that in order to reduce parameterization requirements we adopted the setting $t_{sha} = e_{size} = p_{size}/10$.

The **DECMO**(*problem*, *popSize*, *maxGen*) function contains only three input parameters since we assume that the P and Q subpopulations are evolved either with the literature recommended paramterizations for SPEA2 and DEMO/GDE3 or with the parameterizations of these methods one would apply when running stand-alone versions of the MOEAs on the given *problem*. The *popSize* indicates the size of the population one would use when applying stand-alone runs of SPEA2 or DEMO/GDE3. Inside the main (generational loop), the instructions from lines 10-13 are responsible for implementing the weak sharing stage and the instructions from lines 16-23 are implementing the strong sharing stage. Like almost every MOEA, DECMO returns the *PN* obtained at the end of the run. When solving

Algorithm 3 The DECMO multi-objective evolutionary algorithm

```
1:  function DECMO(problem, popSize, maxGen)
2:      p_size, q_size ← popSize/2
3:      t_sha, e_size ← p_size/10
4:      t ← 1
5:      P ← INITIALIZEPOPULATION(p_size, problem)
6:      Q ← INITIALIZEPOPULATION(q_size, problem)
7:      while t ≤ maxGen do
8:          P' ← EVOLVEOFFSPRINGSPEA2(P)
9:          Q' ← EVOLVEOFFSPRINGDE(Q)
10:         x^p ← DUPLICATERANDOMINDIVIDUAL(P)
11:         x^q ← DUPLICATERANDOMINDIVIDUAL(Q)
12:         P' ← P' ∪ x^q
13:         Q' ← Q' ∪ x^p
14:         P ← E_sel(P ∪ P', p_size)
15:         Q ← E_sel(Q', q_size)
16:         if t_sha = 1 then
17:             E ← E_sel(P ∪ Q, e_size)
18:             P ← E_sel(P ∪ E, p_size)
19:             Q ← E_sel(Q ∪ E, p_size)
20:             t_sha ← p_size/10
21:         else
22:             t_sha ← t_sha − 1
23:         end if
24:         t ← t + 1
25:     end while
26:     PN ← E_sel(P ∪ Q, popSize)
27:     return PN
28: end function
```

CIMOOPs, one would normally modify this process and return the *PN* extracted from the set that contains every feasible individual generated during the run – as we have previously shown in the **NSGA-II-SEARCH**( ) function from Algorithm 2.

6.2.2 Comparative Performance on Benchmark MOOPs

In order to assess the performance of DECMO we performed several tests using the 25 artificial MOOPs mentioned in Section 3.1.1. At first we wanted to evaluate the comparative performance of DECMO with respect to stand-alone versions of SPEA2 and GDE3. The population (archive) size of each MOEA was set at 200 (i.e., $p_{size} = q_{size} = 100$ in the case of DECMO). All three algorithms were allowed to perform 50000 fitness evaluations per run. We repeated each experiment (i.e., MOEA-MOOP combination) 50 times. In the case of SPEA2 (with SBX and PM) we applied the standard parmeterization: 0.9 for

the crossover probability, $\alpha = 20$ (SBX crossover index), $\gamma = 20$ and $p_{mut} = 1/n$ (PM mutation distribution index and mutation rate). In the case of (DEMO/)GDE3, we apply the DE/rand/1/bin strategy with the parameterization $CR = 0.3$ and $F = 0.5$ (recommended in [72]). The exact same settings used in the stand-alone versions are also used to parameterize the evolutionary processes that steer the P and Q subpopulations of DECMO.

In Figure 6.1 we present the average convergence behavior of the three tested MOEAs on 10 MOOPs. The problems have been specially selected in order to illustrate the performance of DECMO in cases where:

- both (incorporated) evolutionary models perform equally well: DTLZ2 and DTLZ4;
- both (incorporated) evolutionary models perform well, but one converges slightly faster: DTLZ7, KSW10 and LZ09-F1;
- both (incorporated) evolutionary models perform well, but one converges much faster than the other: DTLZ1 and ZDT6;
- one (incorporated) evolutionary model performs well and the other does not: DTLZ6 and WFG1;
- both (incorporated) evolutionary models perform poorly, but one of them still displays a significantly better performance than the other: LZ09-F8.

The plots from Figure 6.1 clearly show that **DECMO is very successful** in its role as **it is generally able to replicate the behavior of the best incorporated evolutionary model**. Furthermore, it is important to remark the performance of DECMO on the LZ09-F1 problem. Initially, the coevolutionary approach demonstrates the same rapid convergence as SPEA2 but, after 25000 fitness evaluations, DECMO switches to a GDE3 convergence behavior as the DE-based MOEA generally performs better towards the end of the runs. On three problems (DTLZ1, DTLZ4 and WFG1), DECMO is even able to surpass the stand-alone performance of both SPEA2 and GDE3.

In Chapter 5 we have presented empirical evidence that, in the case of NSGA-II and SPEA2, for many MOOPs, after reaching *PN*s with $Ind_H > 0.85$, the algorithms enter a so-called late stage of convergence (in which improvements tend to come at a much higher *nfe* count). In order to clarify (and substantiate) our initial visual observations with regard to the average performance of DECMO, we present in Table 6.1 information (mean and standard deviation) regarding the *nfe* required by SPEA2, GDE3 and DECMO to reach $Ind_H > 0.85$ for each of the 10 MOOPs presented in Figure 6.1. For each MOEA-MOOP combination we also computed a successful run ratio (not: *srr*) that indicates which algorithms were able to reach a late stage of convergence during each of the 50 individual runs. In order to emphasize the importance of delivering a constantly good performance:

- although the *nfe*-related values from Table 6.1 were computed only taking into account the individual runs where the algorithms managed to discover a *PN* with $Ind_H > 0.85$;
- the best result for each MOOP is selected by considering only those MOEAs that were able to achieve a successful run ratio of 1.0 for that problem (e.g., DTLZ4).

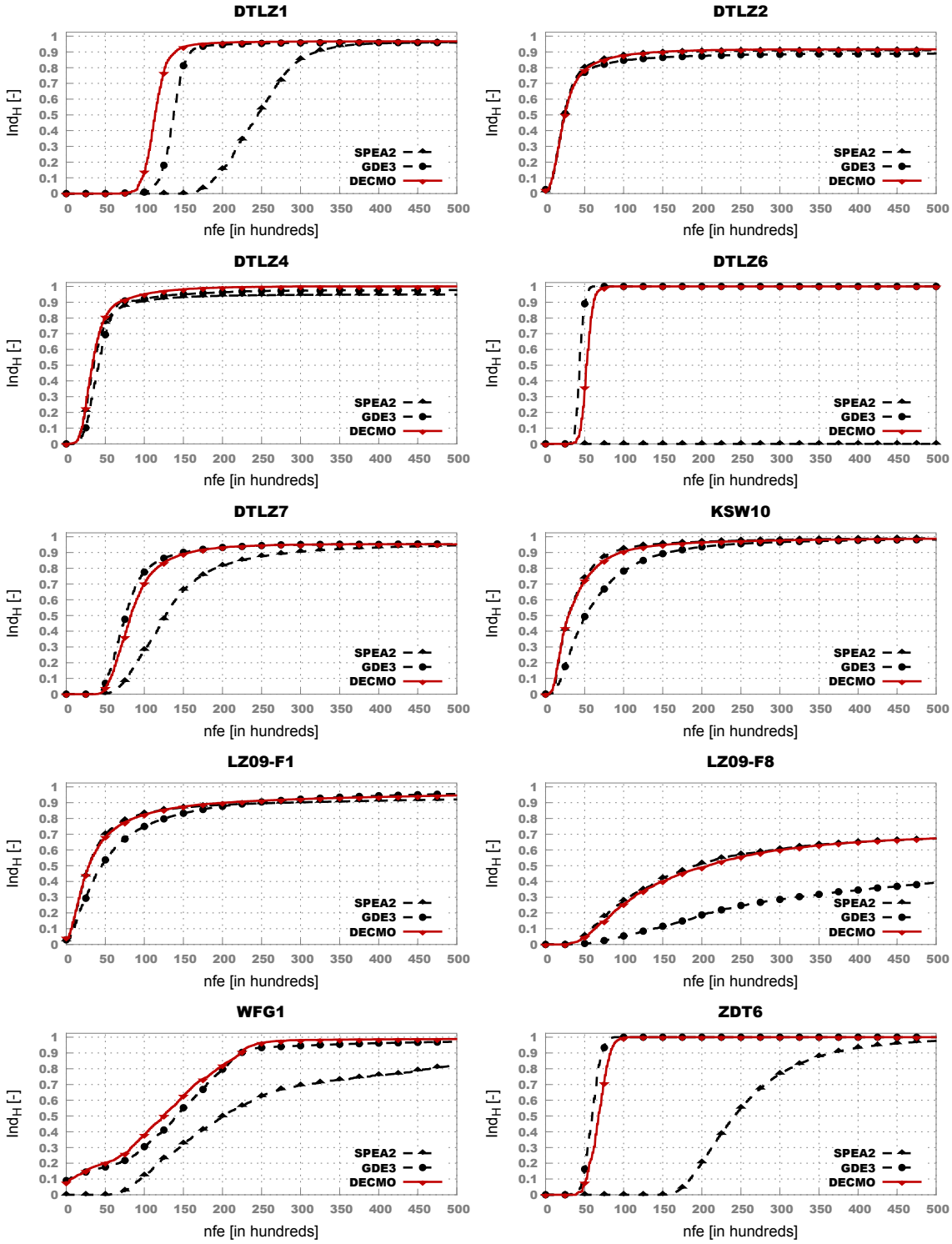


Figure 6.1: The convergence behavior of SPEA2, GDE3 and DECMO averaged over 50 runs.

Table 6.1: Mean and standard deviation information regarding the *nfe* required in order to reach average *PN*s with hypervolumes higher than 0.85. The best result for each MOOP is highlighted.

Problem	SPEA2			GDE3			DECMO		
	nfe		*srr*	*nfe*		*srr*	*nfe*		*srr*
	avg.	$\pm\sigma$		avg.	$\pm\sigma$		avg.	$\pm\sigma$	
DTLZ1	27908	3489.3	1.00	15068	810.8	1.00	12688	983.3	1.00
DTLZ2	7216	735.0	1.00	10348	923.9	1.00	7576	513.7	1.00
DTLZ4	5573	396.8	0.90	6164	337.3	1.00	5626	356.0	1.00
DTLZ6	69578	1712.0	0.94	4884	259.0	1.00	5980	271.8	1.00
DTLZ7	22364	1247.2	1.00	11940	622.4	1.00	13048	769.4	1.00
KSW10	6872	600.4	1.00	12564	735.9	1.00	7576	831.7	1.00
LZ09-F1	12416	2484.2	1.00	16728	930.9	1.00	12184	866.5	1.00
LZ09-F8	-	-	0.00	-	-	0.00	-	-	0.00
WFG1	39720	8289.1	0.40	21012	1131.7	0.98	21596	1669.2	1.00
ZFT6	33416	909.7	1.00	6884	433.5	1.00	7800	557.0	1.00

In the case of the MOOPs from Table 6.1 where at least two algorithms were able to achieve $srr = 1$, the differences between the average *nfe* required by the best performer and those required by the second-best performer are also statistically significant[8] with the exception of LZ09-F1. In this case, the better performance displayed by DECMO in front of SPEA2 is not statistically significant.

In order to better show the general performance of DECMO when compared to SPEA2 and GDE3, in Figure 6.2 we plot HRPCs obtained when considering all the 25 benchmark MOOPs (please refer back to Section 3.3.2). We present HRPCs plots obtained with:

- the basic ranking schema;
- a pessimistic ranking schema with $th = 0.01$ (i.e., that requires a minimum Ind_H-measured *PN* improvement of 1% in order to apply a rank improvement)
- a pessimistic ranking schema with $th = 0.05$ (i.e., 5% improvement required)
- a statistical ranking schema

All four DECMO vs SPEA2 vs GDE3 plots confirm the fact that the coevolutionary MOEA is indeed able to generally display the performance of it's best incorporated evolutionary model in the key-interest part from the beginning of the runs. Furthermore, towards the end of the runs, DECMO displays a better performance than both SPEA2 and GDE3.

In Figures 6.3 and Figures 6.4 we also present HRPC charts obtained when comparing DECMO with NSGA-II (same parameterization as SPEA2) and MOEA/D-DE (with a

[8]We applied a one-sided Mann-Whitney-Wilcoxon test with a considered significance level of 0.05.

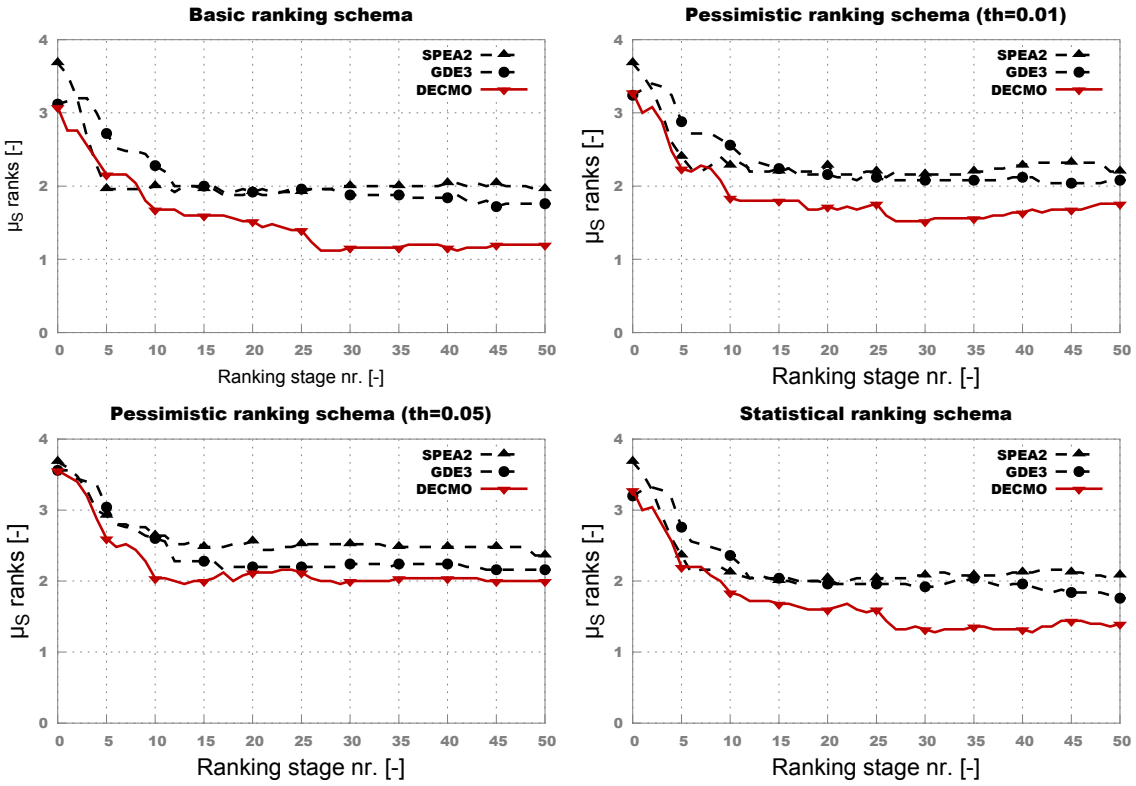


Figure 6.2: HRPCs obtained when comparing SPEA2, GDE3 and DECMO over all 25 benchmark MOOPs.

population size of 500 and all other parameters set according to the description in [78]).[9]. These HRPCs indicate that:

- NSGA-II is generally able to converge faster than DECMO in the initial stages of runs (i.e., for $nfe \leq 5000$) and, although statistically significant, this improvement does not seem very large as it is not visible when using the $th = 0.05$ pessimistic ranking schema;
- MOEA/D-DE matches the performance of DECMO for $nfe \leq 5000$ and outperforms the coevolutionary approach towards the end of the runs (i.e., $nfe \geq 40000$);
- DECMO seems a generally better choice when wanting (or being able) to run an optimization with 10000 to 40000 *nfe*.

The results of the comparisons with NSGA-II and MOEA/D-DE show that, although our coevolutionary approach can be considered a very successful proof of concept, **DECMO also provides room for improvement** – especially when considering its performance against NSGA-II in the beginning of the runs.

[9]In the case of SPEA2, GDE3, NSGA-II and MOEA/D-DE we relied on implementations largely based on those from the jMetal framework [157]

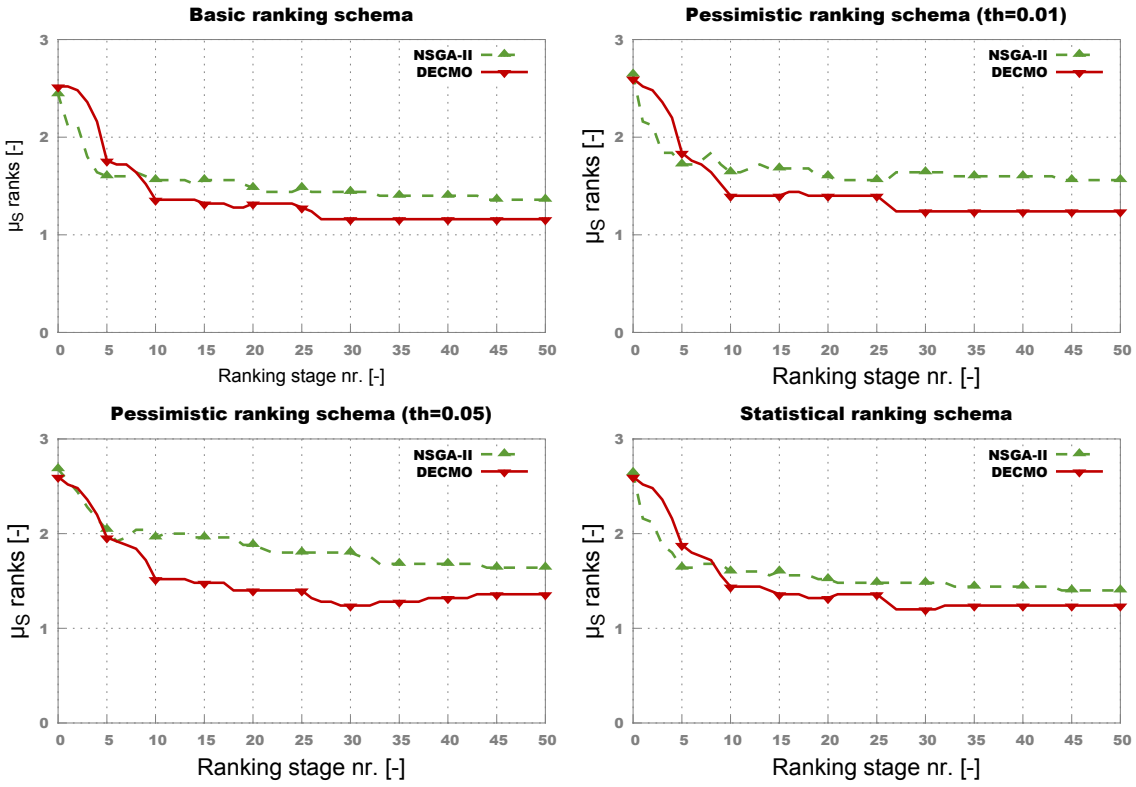


Figure 6.3: HRPCs of DECMO vs NSGA-II.

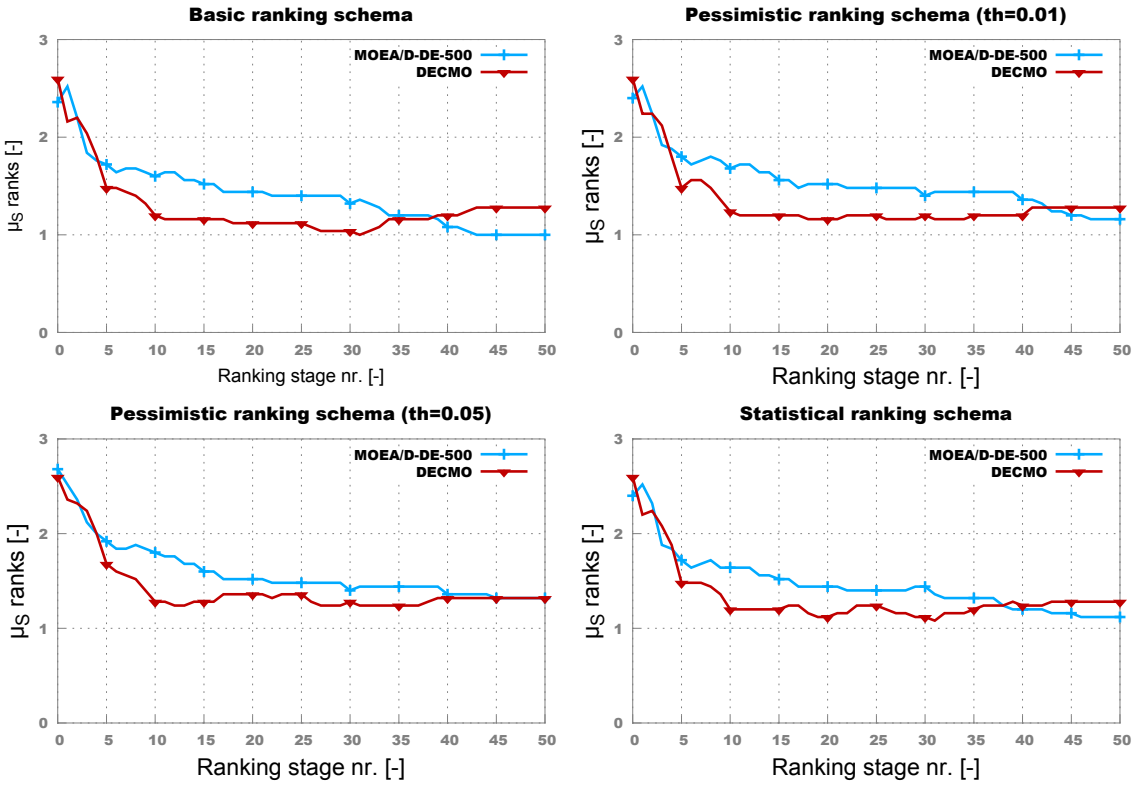


Figure 6.4: HRPCs of DECMO vs MOEA/D-DE-500.

6.3 The Refinement: DECMO2

6.3.1 Overview

In order to improve the efficiency of our initial coevolutionary approach, we considered two major enhancements: the integration of a decomposition strategy and the dynamic allocation of fitness evaluations during the run. We named this improved coevolutionary algorithm DECMO2. In Section 6.3.5 we present several results that indicate that DECMO2 is able to completely surpass its predecessor and, when considering all the 25 benchmark MOOPs described in Section 3.1.1, it displays a robustness and a *nfe*-measured efficiency that is unmatched by other state-of-the-art MOEAs.

DECMO2 is also a hybrid approach that mainly relies on coevolving two populations (P and Q) of equal sizes where P is evolved via the SPEA2 evolutionary models and Q is evolved using the DE mechanism described at the start of Section 6.2.1. The novelty in DECMO2 is represented by an external archive, marked with A, that is maintained according to a decomposition-based principle. Thus, DECMO2 was designed to integrate what we previously labeled as three major milestones in MOEA research: Pareto-based elitism, DE-based search and decomposition-based strategies.

In order to efficiently integrate all there concepts and thus obtain a MOEA that would be preferred by DMs in search of a black-box a posteriori MOO solver, the fitness sharing mechanism described in DECMO was redesigned / improved. While the environmental selection operator proposed by SPEA2, i.e., $E_{sel}(Set, count)$, still plays an important role, the version of it we use in DECMO2 is slightly modified as it includes a filter that first removes (objective-wise) duplicates from *Set* before starting the actual trimming process. As a direct result, very often applications of the operator are possible and the (strong) fitness sharing stage can be performed after every generation.

6.3.2 Decompostion-based Archive

With regard to its predecessor, DECMO2 proposes a major structural modification that comes in the form of the new archive (population) A with $|A| = a_{size}$. This archive is maintained according to a **decomposition-based principle that uses a modified weighted Tschebyscheff approach** (2.10). The working principles of this new (coevolved) component are inspired (and quite similar) to those initially proposed in [75] and re-popularized by MOEA/D [76].

Before proceeding with the description of the functionality of this archive, we must first make a few notations. Therefore, let us mark:

- the *current optimal reference point* of the DECMO2 run with: $\mathbf{p^{opt}} = (p_1^{opt}, \ldots, p_m^{opt})$. More formally, $p_i^{opt} = min\left\{o_i(x) \,|\, x \in D^{eval}\right\}$ for all $i \in \{1, \ldots, m\}$ where $D^E \subset D^n$ contains all the (feasible) individuals evaluated during the DECMO2 run (till the time $\mathbf{p^{opt}}$ is computed).

- an arbitrary *objective weighting vector* with $\mathbf{w}^i = (w_1^i, \ldots, w_m^i) \in [0,1]^m$ subject to

$w^i_j \geq 0$ and $\sum_{j=1}^{m} w^i_j = 1$ for all $j \in \{1,\ldots,m\}$ and for all $i \in \{1,\ldots,a_{size}\}$;

- the *Tschebyscheff distance* between a point $\mathbf{x} \in D^n$ and the current optimal reference point when considering an arbitrary weighting of the objectives $\mathbf{w}$ with:

$$dist_T(\mathbf{x},\mathbf{w}) = \max_{j\in\{1,\ldots,m\}} \left\{ w_j \left[o_i(\mathbf{x}) - p_i^{opt} \right] \right\}. \tag{6.4}$$

For any MOOP to be solved, the idea is to generate, prior to the start of the optimization, a_{size} uniformly spread objective weighting vectors $\mathbf{w}^1, \mathbf{w}^2, \ldots, \mathbf{w}^{a_{size}}$ and to subsequently use these vectors inside the $dist_T$ formulation in order to define a decomposition of the original MOOP into a_{size} single-objective optimization problems. After being generated, the weighting vectors remain constant throughout the optimization run. Archive A is organized as a set of pairs (i.e., 2-tuples) that we mark $\langle \mathbf{w}^i, \mathbf{x}^i \rangle$, with $\mathbf{w}^i \in [0,1]^m$ and $\mathbf{x}^i \in D^n$, $i \in \{1,\ldots,a_{size}\}$. Knowing that $\mathbf{w}^i$ is fixed, during the run of DECMO2, A is constantly updated in order to minimize $dist_T(\mathbf{x},\mathbf{w})$ for all $i \in \{1,\ldots,a_{size}\}$. Therefore, after a new individual $\mathbf{x}*$ is generated during the DECMO2 run, after performing the standard fitness evaluation:

1. we update the current optimal reference point $\mathbf{p^{opt}}$ as $\mathbf{x}^*$ might display improvements with regard to certain single-objectives;

2. we construct A' – the improvable subset of the current archive:

$$A' = \left\{ \langle \mathbf{w}', \mathbf{x}' \rangle \mid \langle \mathbf{w}', \mathbf{x}' \rangle \in A \text{ and } dist_T(\mathbf{x}^*, \mathbf{w}') < dist_T(\mathbf{x}', \mathbf{w}') \right\} \tag{6.5}$$

3. if $A' \neq \Phi$, we:

   (a) identify $\langle \mathbf{w^r}, \mathbf{x^r} \rangle \in A'$ that maximizes $\Delta_{dist_T} = dist_T(\mathbf{x^r}, \mathbf{w^r} - dist_T(\mathbf{x}^*, \mathbf{w^r})$, i.e., the single-objective optimization problem that $\mathbf{x}^*$ can improve the most;

   (b) update the archive by replacing the most improvable single-objective solution (i.e., $A = A \setminus \langle \mathbf{w^r}, \mathbf{x^r} \rangle$) with the current individual: $A = A \cup \langle \mathbf{w^r}, \mathbf{x}^* \rangle$;

Steps 3.(a) and 3.(b) can be easily redefined in order to incorporate replacement strategies that are more advanced than the presently proposed one. However, several comparative tests have shown that the simple greedy replacement offers a very good trade-off between complexity and general performance.

6.3.3 Search Adaptation and Fitness Sharing

In Section 2.3.1 we have argued that, in a broad sense, all EAs are adaptive by design as their mechanics are governed by a "survival of the fittest" paradigm that prompts the evolved population to progressively "adapt" – i.e., develop and / or retain the genetic makeup that is deemed successful. Furthermore, in Section 6.2.2 we have presented strong

empirical evidence that **a simple (symbiotic) cooperative-coevolutionary approach has the general ability to adapt on a meta level** and mimic the convergence behavior of its best incorporated evolutionary model.

In light of the latter observation, in DECMO2 we envisioned a mechanism that is aimed to slightly (but actively) bias the coevolutionary search towards the particular evolutionary model that seems to be more successful during the current part of the optimization. Concretely, at each odd-numbered generation $t, t \geq 3$, we reward the evolutionary model that was more successful in the previous, $t-1$, generation by allowing it to create an extra number of e_{size} bonus (offspring) individuals. The success of a particular evolutionary model is assessed by computing its *generational archive insertion ratio*. As such, after each even-numbered generation we compute:

- ϕ^P - the archive insertion ratio achieved by the p_{size} offspring generated in subpopulation P using SBX and PM;
- ϕ^Q - the archive insertion ratio achieved by the q_{size} offspring generated in subpopulation Q using the *DE/rand/1/bin* strategy;
- ϕ^A - the archive insertion ratio achieved when creating e_{size} individuals by applying *DE/rand/1/bin* on individuals selected directly from A. In this case, the e_{size} trial vectors are created for individuals that represent the solutions of the e_{size} single-objective optimization problems that have not been updated for the longest periods.

When considering the previous notations from Section 6.2.1 and 2.3.3, the DECMO2 explicit search adaptation enhancement works as follows:

- if at an arbitrary even-numbered generation t, $\phi^P > \phi^Q$ and $\phi^P > \phi^A$, at generation $t+1$ the size of the offspring population generated during the SPEA2 search (i.e., P')will be set at: $p_{size} + e_{size}$;
- if at an arbitrary even-numbered generation t, $\phi^Q > \phi^P$ and $\phi^Q > \phi^A$, at generation $t+1$, after the end of the Q'' processing cycle (i.e., when $Q'' = \Phi$), Q'' will be re-initialized with a set of e_{size} individuals randomly extracted from Q and the *DE/rand/1/bin* cycle will resume and continue until Q'' is fully processed again;

If neither success criterion is met, we shall create the e_{size} reward individuals of generation $t+1$ by applying *DE/rand/1/bin* on individuals selected directly from A.

DECMO2 generally achieves very good results when the sizes of its components satisfy the relation:

$$\begin{cases} e_{size} & = \frac{a_{size}}{10} \\ p_{size} & = q_{size} \\ a_{size} & = p_{size} + q_{size} + e_{size} \end{cases} \tag{6.6}$$

Like with its predecessor, the main channel through which fitness is shared among the coevolved components of DECMO2 is formed by a subset of elite individuals. We mark this subset with E and mention that its size is fixed to e_{size}. At a given generation $t, t \geq 1$, the very last steps are:

- the construction of the elite subset E via the environmental selection operator: $E = E_{sel}(P \cup Q \cup A, e_{size})$
- the attempt to spread these elite global solutions among the main coevolved components: $P = E_{sel}(P \cup E, p_{size})$ and $Q = E_{sel}(Q \cup E, q_{size})$;

6.3.4 Algorithmic Description

Algorithm 4 presents the main computational loop of DECMO2 as well as the initialization of its main components[10]: P, Q and A. Like its predecessor, DECMO2 also proposes only three input parameters as it is intended to run with more or less fixed (i.e., literature recommended) settings for SBX, PM and *DE/rand/1/bin*. In order to obtain results that are comparable with other methods, after the computation of the last generation, the **DECMO2**( ) function returns a *PN* extracted from the individuals stored in P, Q and A.

The auxiliary functions used across Algorithm 4 are:

- **COMPUTESIZES**(*archiveSize*) - given an *archiveSize*, this function solves (6.6) and computes p_{size}, q_{size}, and e_{size}.
- **INITIALIZEARCHIVE**(*problem*, *count*) - firstly, this function generates a number of *count* uniformly spread object weight vectors, $\mathbf{w}^1, \mathbf{w}^2, \ldots \mathbf{w}^{count}$, with a dimensionality that matches the number of single-objectives proposed by the given *problem*. Secondly it creates and returns an incomplete archive: $A = \{\langle \mathbf{w}^1, \rangle, \ldots, \langle \mathbf{w}^{count}, \rangle\}$.
- **CREATEINDIVIDUAL**(*problem*) - according to the definition (i.e., D^n) of the given *problem*, this function randomly initializes and returns a single individual (i.e., a real-coded variable vector).
- **INSERTINTOARCHIVE**(*ArchiveSet*, **x**) - this procedure locates a single incomplete pair of the form $\langle \mathbf{w}^i, \rangle$, with $i \in \{1, \ldots, |ArchiveSet|\}$ inside the *ArchiveSet*, performs an update (i.e., $Archive = Archive \setminus \langle \mathbf{w}^i, \rangle$ and $Archive = Archive \cup \langle \mathbf{w}^i, \mathbf{x} \rangle$), and returns the *ArchiveSet*.
- **EVOLVENEXTGENSPEA2**(*ParentSet*, *count*) - starting from the given *ParentSet*, this function generates a number of *count* offspring using the SPEA2 evolutionary model described in Section 2.3.3. The function returns two entities:
  1. a new set of size $|ParentSet|$ obtained by applying the environmental selection operator on the union of the initial *ParentSet* and the set containing the newly generated offspring;
  2. a real number representing the archive insertion ratio achieved by the *count* offspring that have been generated using SBX and PM.

[10]The presented pseudo-code and its corresponding description are primarily designed for readability and they should not be interpreted as instructions for an optimal implementation of DECMO2.

Algorithm 4 The DECMO2 multi-objective evolutionary algorithm

function DECMO2($problem$, a_{size}, $maxGen$)
  $P, Q \leftarrow \Phi$
  $\langle p_{size}, q_{size}, e_{size} \rangle \leftarrow$ **COMPUTESIZES**(a_{size})
  $A \leftarrow$ **INITIALIZEARCHIVE**($problem$, a_{size})
  $i \leftarrow 1$
  while $i \leq a_{size}$ **do**
    $\mathbf{x} \leftarrow$ **CREATEINDIVIDUAL**($problem$)
    $A \leftarrow$ **INSERTINTOARCHIVE**(A, $\mathbf{x}$)
    if $i \leq p_{size}$ **then**
      $P \leftarrow P \cup \{\mathbf{x}\}$
    else
      if $i \leq (p_{size} + q_{size})$ and $i > p_{size}$ **then**
        $Q \leftarrow Q \cup \{\mathbf{x}\}$
      end if
    end if
    $i \leftarrow i + 1$
  end while
  $\phi^P, \phi^Q, \phi^A, t \leftarrow 1$
  while $t \leq maxGen$ **do**
    $p_{bonus}, q_{bonus}, \leftarrow 0$
    $a_{bonus} \leftarrow e_{size}$
    if $t \in \{2k+1 : k \in \mathbb{Z}\}$ **then**
      $p_{bonus}, q_{bonus}, a_{bonus} \leftarrow 0$
      if $\phi^P > \phi^Q$ and $\phi^P > \phi^A$ **then**
        $p_{bonus} = e_{size}$
        $a_{bonus} \leftarrow 0$
      end if
      if $\phi^Q > \phi^P$ and $\phi^Q > \phi^A$ **then**
        $q_{bonus} = e_{size}$
        $a_{bonus} \leftarrow 0$
      end if
    end if
    $\langle P, \phi^P \rangle \leftarrow$ **EVOLVENEXTGENSPEA2**(P, $p_{size} + p_{bonus}$)
    $\langle Q, \phi^Q \rangle \leftarrow$ **EVOLVENEXTGENDE**(Q, $q_{size} + q_{bonus}$)
    $\phi^A \leftarrow$ **EVOLVEARCHIVEIND**(A, a_{bonus})
    $E \leftarrow E_{sel}(P \cup Q \cup A, e_{size})$
    $P \leftarrow E_{sel}(P \cup E, p_{size})$
    $Q \leftarrow E_{sel}(Q \cup Q, q_{size})$
    $t \leftarrow t + 1$
  end while
  $PN \leftarrow E_{sel}(P \cup Q \cup A, a_{size})$
  return PN
end function

- **EVOLVENEXTGENDE**(*ParentSet*, *count*) - starting from the given *ParentSet*, this function generates a number of *count* (offspring) trial vectors using the DE-based evolutionary model described in Section 6.2.1. The function returns two entities:
  1. a new set of size $|ParentSet|$ obtained after applying a DEMO/GDE3-like evolutionary model on the *ParentSet*;
  2. a real number representing the archive insertion ratio achieved by the *count* (offspring) trial vectors that have been generated using the *DE/rand/1/bin* strategy.
- **EVOLVEARCHIVEIND**(*ArchiveSet*, *count*) - this function uses the *DE/rand/1/bin* strategy to create a number of *count* (offspring) trial vectors using individuals from the pairs that make up the given *ArchiveSet*. The returned value is the archive insertion ratio achieved by the newly generated individuals. If $count = 0$, the function also returns 0.

The above descriptions also help to highlight one of the **major shortcomings of DECMO2**: **the high structural and computational complexity**[11]. For example, when also considering the incorporated SPEA2 and DEMO/GDE3-style evolutionary processes, the environmental selection operator is applied five times during a single generation. In the next two sections we shall provide strong empirical evidence (on bechmark MOOPs and real-life CIMOOPs) that **the structural complexity of DECMO2 is fully compensated by the general convergence performance** displayed by this hybrid and adaptive coevolutionary MOEA.

6.3.5 Comparative Performance on Benchmark MOOPs

In order to test the general convergence behavior of DECMO2, we performed comparative tests with SPEA2, GDE3, NSGA-II, MOEA/D-DE and DECMO on the 25 artificial (benchmark) MOOPs described in Section 3.1.1. As the **main intent of our tests is to estimate how robust these algorithms are** (when used as black-box a posteriori MOO solvers), for each MOEA we applied a **fixed parameterization across all test problems**. We performed 50000 fitness evaluations during each optimization run and we made 100 independent runs for each MOEA-MOOP combination. In the case of SPEA2, GDE3 and NSGA-II we used a population / archive size of 200 and all other parameters were set as previously described in Section 6.2.2 (i.e., we used literature recommended settings). In the case of MOEA/D-DE we used the same version as the one in the DECMO tests (i.e., fixed population size of 500) but we also compared DECMO2 to the literature recommended variant that uses a population size of 300 for problems with two objectives and a population size of 595 for MOOPs with three objective.

In the case of DECMO we used a subpopulation size of 100. For DECMO2 we set the archive size at 200. For both coevolutionary MOEAs we used the literature recommended settings for the SPEA2 evolutionary model (i.e., subpopulation *P*). For the DE-based

[11] That is somewhat expected when considering that we are trying to harmoniously integrate three different MOEA strategies.

subpopulation Q we applied the *DE/rand/1/bin* strategy with control parameters aimed to (I) ensure a good trade between exploration and intensification ($F = 0.5$) and (II) stimulate a minor increase in population diversity (i.e., $CR = 0.2$) as shown by Zaharie in [158].

First of all, in Figure 6.5 we present the average Ind_H measured performance of all six MOEAs over the entire benchmark set. These results indicate that both coevolutionary methods (with a plus for DECMO2) have the tendency to converge faster than their counterparts. Interestingly, this chart indicates that DECMO has an advantage over MOEA/D-DE towards the end of the runs (i.e., $nfe > 40000$) while the HRPC plot from Figure 6.4 indicates the opposite. This divergence is due to the fact that MOEA/D-DE performs slightly (but significantly) better than DECMO on many problems, while DECMO has larger advantages on a few MOOPs.

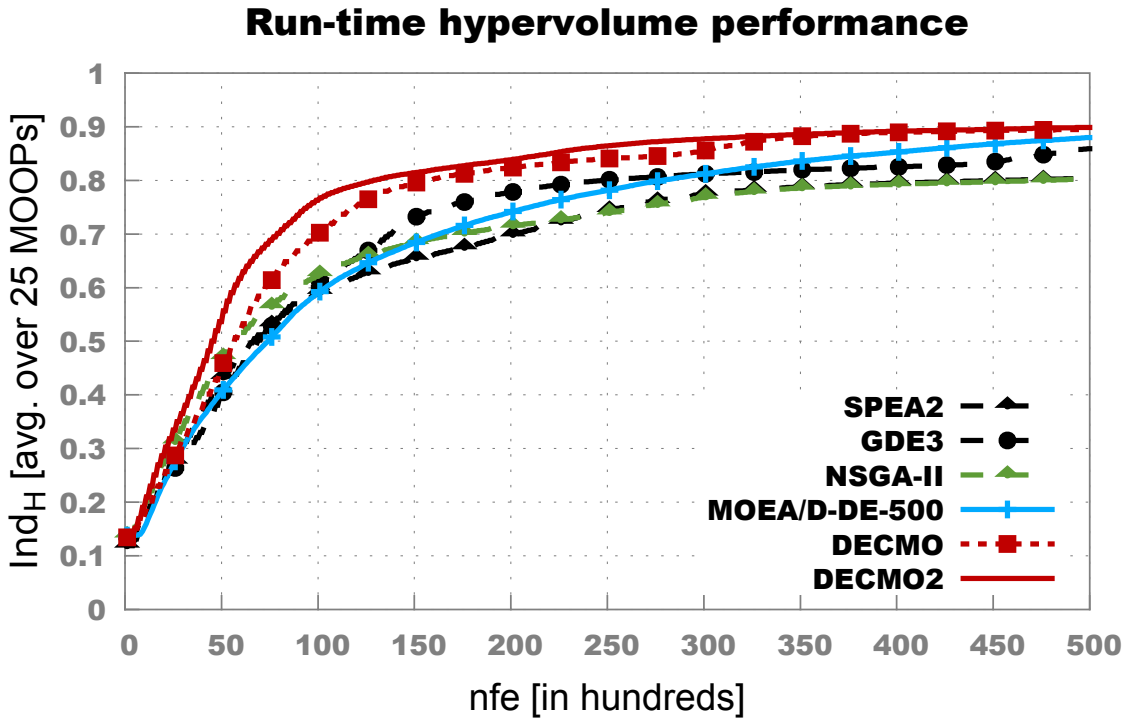


Figure 6.5: Averaged run-time $\mathscr{H}$-measured performance over 20 artificial benchmark MOOPs.

In order to offer more insight into the comparative performance of the six tested MOEAs, in Figure 6.6 we plot HRPCs obtained with four different ranking schemata: basic, pessimistic with $th = 0.01$, pessimistic with $th = 0.05$ and statistical. In Table 6.2 we present the associated *overall average ranks* (i.e., μ_A) and the *final stage average ranks* (i.e., μ_F) achieved by the tested algorithms when applying each ranking schemata. All these results confirm previous remarks regarding the comparative performance of DECMO with regard to the other four (non-coevolutionary) MOEAs. More importantly, all the HRPC plots from Figure 6.6 and the average rank data from Table 6.2 clearly show that **DECMO2 profits significantly from** the introduction of the **decomposition-based archive** and of the

dynamic search adaptation mechanism. As such, *on average*, the refined coevolutionary method surpasses all the other tested MOEAs.

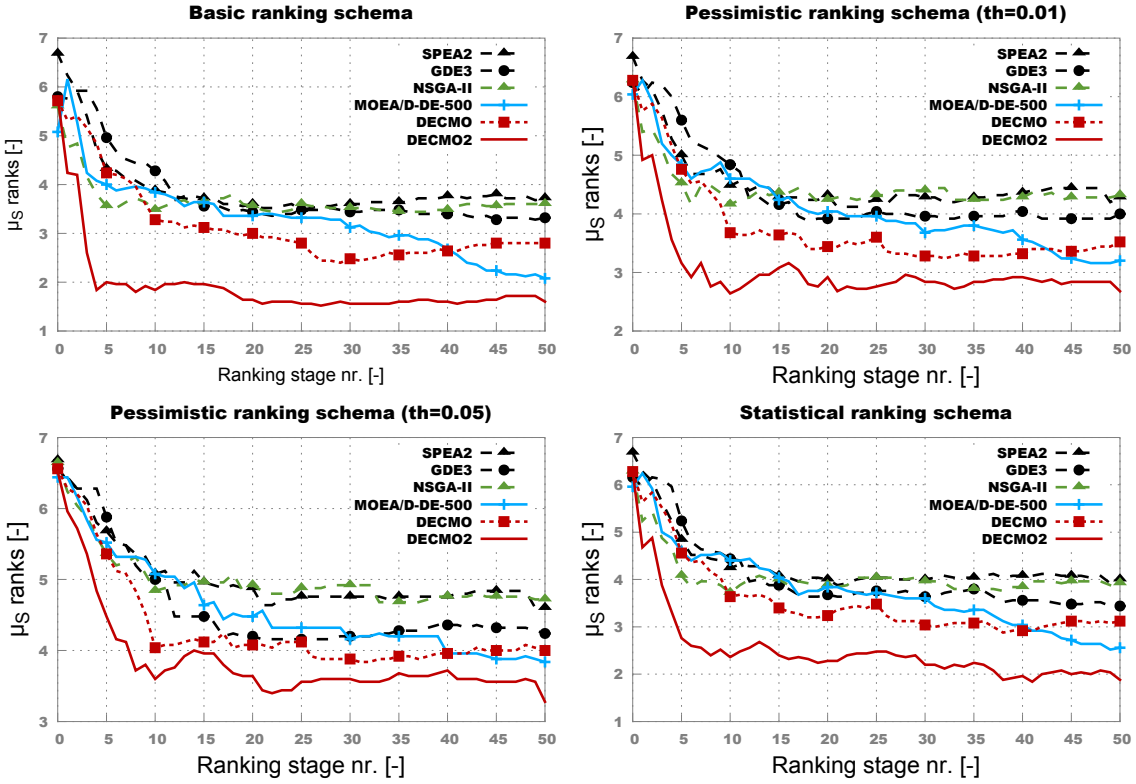


Figure 6.6: HRPCs obtained when comparing DECMO2 with five other MOEAs across 25 benchmark MOOPs.

The very good HRPC-quantified performance exhibited by DECMO2 (especially when comparing to DECMO and the average Ind_H-plots from Figure 6.5) can also be explained by taking a closer look at the number of MOOPs on which each algorithm was able to achieve "full convergence" (i.e., $Ind_H > 0.99$ for the*PN* of the final population). This is because (as explained in Section 3.3.2), at each ranking stage, the bonus / penalty system we applied for all four ranking schemata rewards "full convergence" with a special rank "0" that has the obvious effect of reducing all average ranks like μ_S, μ_A, μ_F. Across the 25 artificial MOOPs, SPEA2 achieved "full convergence" on 5 problems, GDE3 on 6 problems, NSGA-II on 4 problems, MOEA/D-DE on 7 problems, DECMO on 6 problems and DECMO2 on 9 problems. LZ09-F1 is the onlyMOOP on which DECMO2 was unable to "fully convergence", but another algorithm (MOEA/D-DE) managed to do so.

In Figures 6.7 and 6.8 we display the results obtained when performing individual comparisons between DECMO2 and NSGA-II and between DECMO2 and DECMO. All the HRPCs suggest that DECMO2 displays a superior performance.

In Figure 6.9 we present HRPC plots that highlight the comparative convergence behavior of DECMO2 and MOEA/D-DE. The curves indicate that our coevolutionary approach generally performs better, especially in the first 35 ranking stages (i.e., for $nfe \leq 35000$).

Table 6.2: The overall and final stage average ranks achieved by the six tested MOEAs over the benchmark problem set when considering four different ranking schemata. The best (i.e., lowest) values are highlighted.

MOEA	HRPC ranking schemata							
	Basic		Pes. $th = 0.01$		Pes. $th = 0.05$		Statistical	
	μ_A	μ_F	μ_A	μ_F	μ_A	μ_F	μ_A	μ_F
SPEA2	3.92	3.72	4.52	4.28	5.02	4.60	4.29	4.00
GDE3	3.82	3.32	4.34	4.00	4.62	4.24	4.04	3.44
NSGA-II	3.66	3.60	4.42	4.32	4.99	4.72	4.06	3.92
MOEA/D-DE	3.30	2.08	4.09	3.20	4.58	3.84	3.76	2.56
DECMO	3.15	2.80	3.75	3.52	4.30	4.00	3.55	3.12
DECMO2	1.90	1.60	3.05	2.68	3.87	3.28	2.50	1.88

Nevertheless, MOEA/D-DE become quite competitive towards the end of the runs and, the basic ranking schema indicates that it is able to surpass DECMO2 in the last 10 ranking stages (i.e., for $nfe \geq 40000$). Interestingly enough, the other three ranking schemata also show that MOEA/D-DE is able to match DECMO2 but not surpass it. This phenomenon is explained by the fact that on the MOOPs where the two algorithms perform equally well (but are not able to "fully converge"), MOEA/D-DE has the advantage that its *PN* is composed of up to 500 points while the *PN* of DECMO2 contains at most 200 points. This means that while the search strategies are equally successful (by the end of the runs), the *PN* retention strategy of MOEA/D-DE is slightly better (probably because of archive size alone). For a very interesting discussion related to the importance of the interplay between the search strategy and the *PN* retention strategy in MOOAs, please see [159].

In Figure Figure 6.10 we present HRPC plots that highlight the comparative convergence behavior of DECMO2 and the version of MOEA/D-DE that uses a problem-dependent population size (please refer back to Section 3.3.3 for details). In this case, the ability of MOEA/D-DE to match and slightly surpass the performance of DECMO2 towards the end of the runs is also confirmed by the statistical ranking schema. Nevertheless, the HRPCs obtained with the pessimistic $th = 0.01$ ranking schema indicates that the improvements displayed by MOEA/D-DE are not large at all. When considering the most pessimistic ranking schema (i.e., $th = 0.05$), DECMO2 also outperforms this version MOEA/D-DE across all ranking stages This means that when only $> 1\%$ and $> 5\%$ differences of *PS* quality are of interest, DECMO2 still displays a general advantage across the problem set.

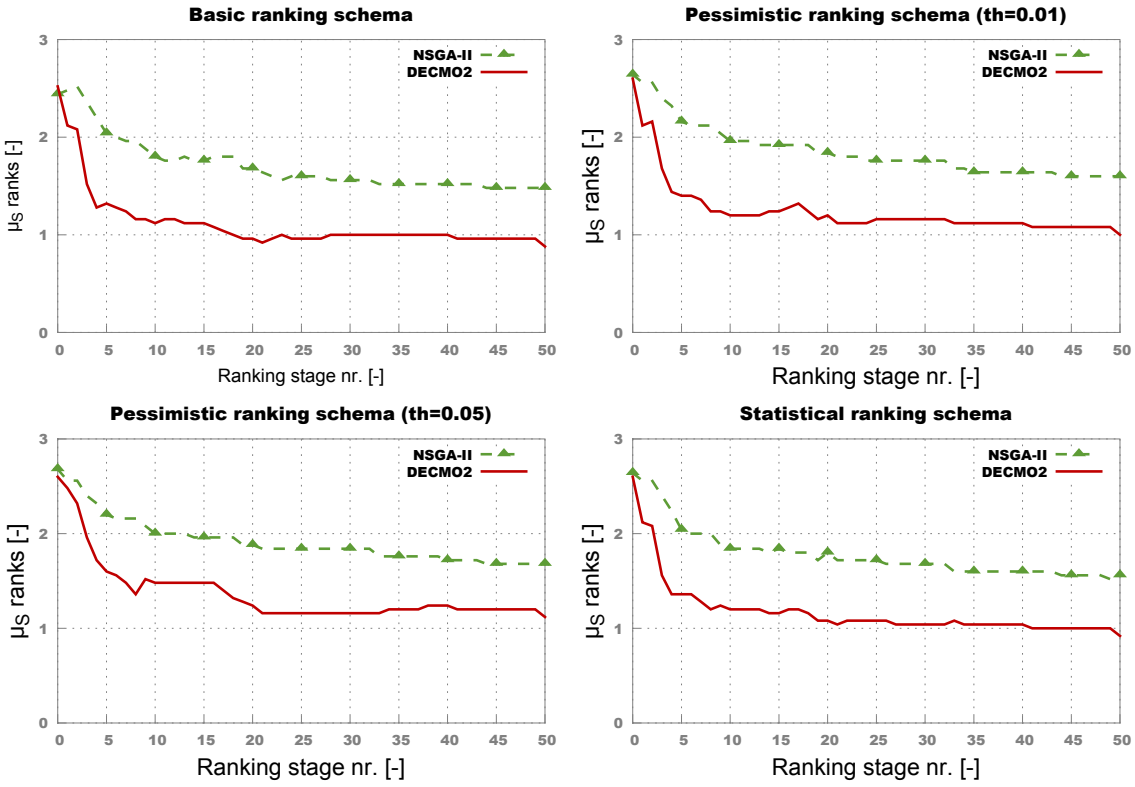


Figure 6.7: HRPCs of DECMO2 vs NSGA-II.

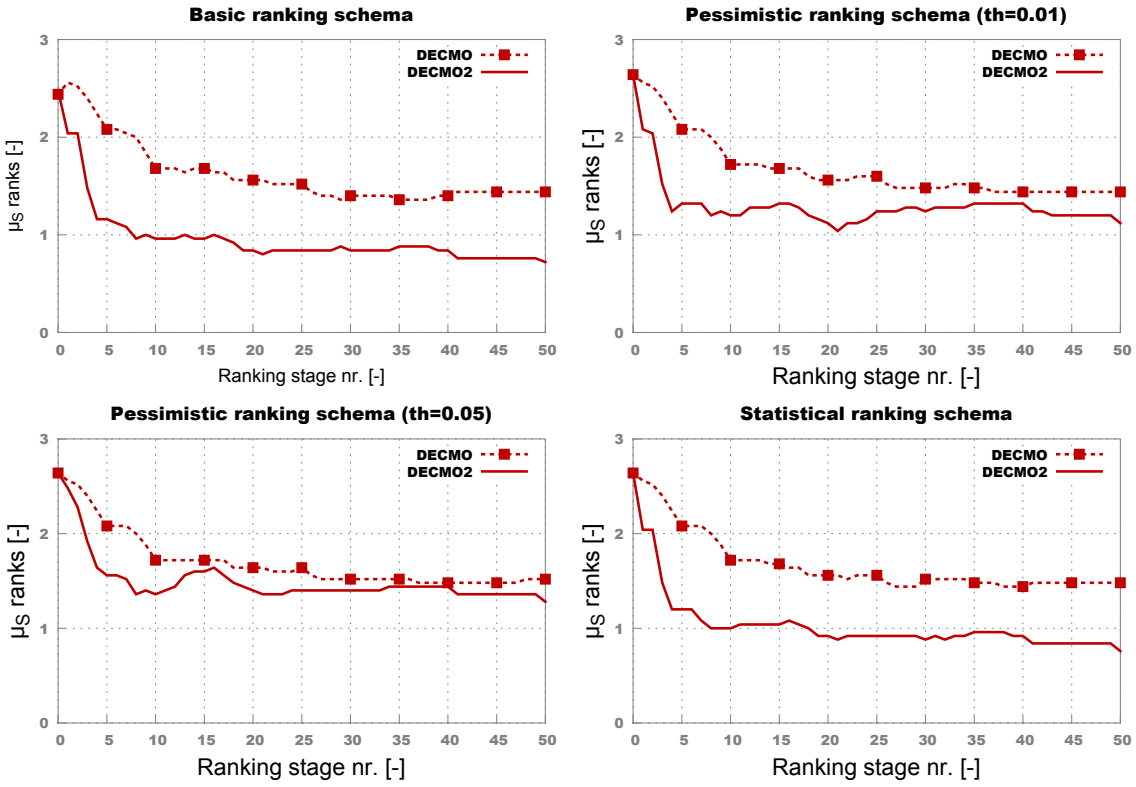


Figure 6.8: HRPCs of DECMO2 vs DECMO.

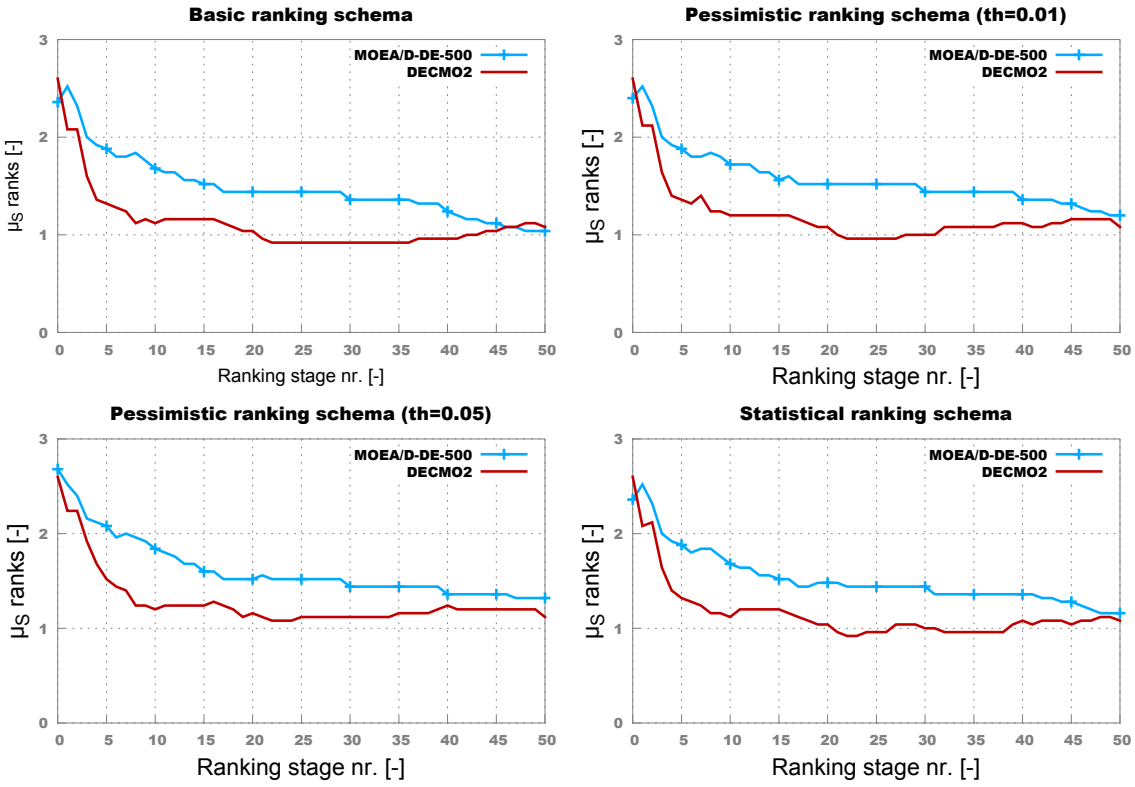


Figure 6.9: HRPCs of DECMO2 vs MOEA/D-DE.

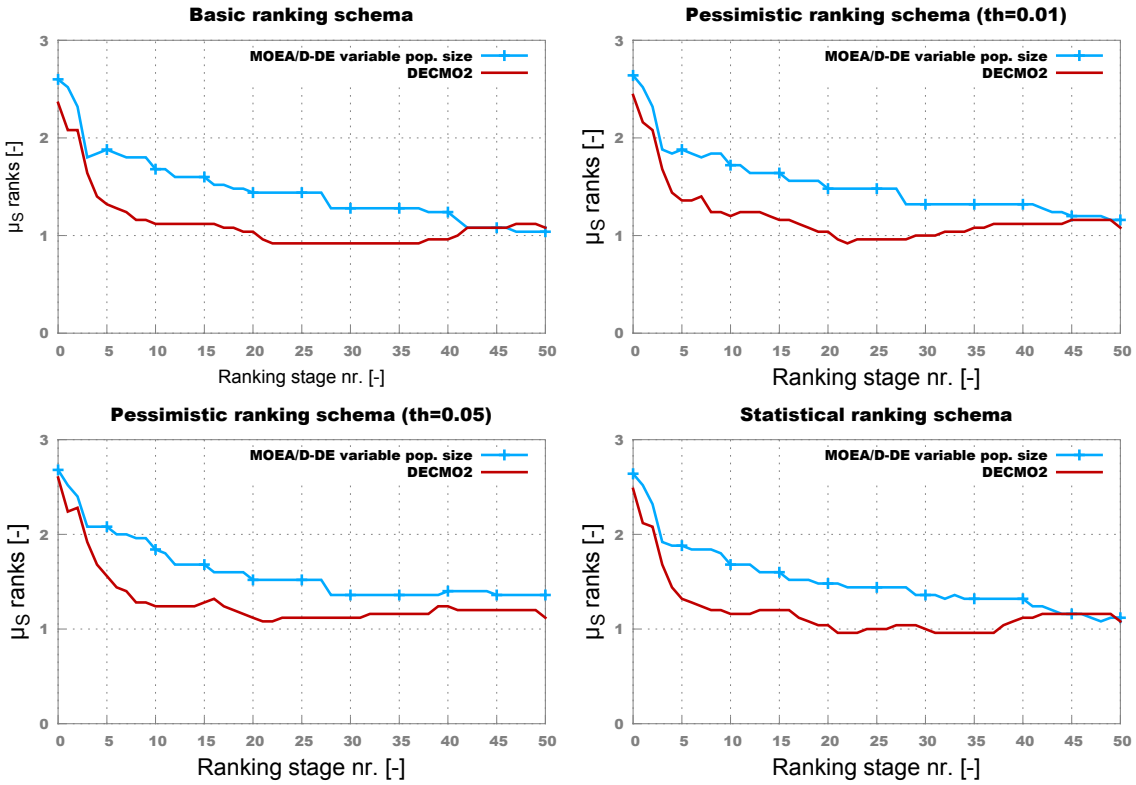


Figure 6.10: HRPCs of DECMO2 vs literature recommended MOEA/D-DE version with problem-dependent population size.

6.3.6 Performance on a CIMOOP

In order to test the performance of DECMO2 on real-life problems, we applied it with the same parameterizaton from the previous section on the (very complicated) IndMOOP4 electrical drive design problem described in Section 3.1.2. We allowed our coevolutionary algorithm to perform only 10000 fitness evaluations because, even when distributing the computations over the computer cluster architecture described in the previous chapters, performing all these evaluations required between 6 and 7 days.

Because of the extremely long run-times, we only managed to perform two independent runs with DECMO2. As a performance reference we used two (legacy) optimization runs performed with SPEA2. Using as reference for computing the Ind_H the best known solutions of this CIMOOP, in Figure 6.11 we present the comparative run-time convergence behavior of the two MOEAs. The chart indicates that DECMO2 is able to converge faster. In this particular setting, the performance improvement exhibited by DECMO2 means that the coevolutionary approach was usually able to find one day sooner *PN*s of similar Ind_H-estimated quality to those that would be obtained by SPEA2.

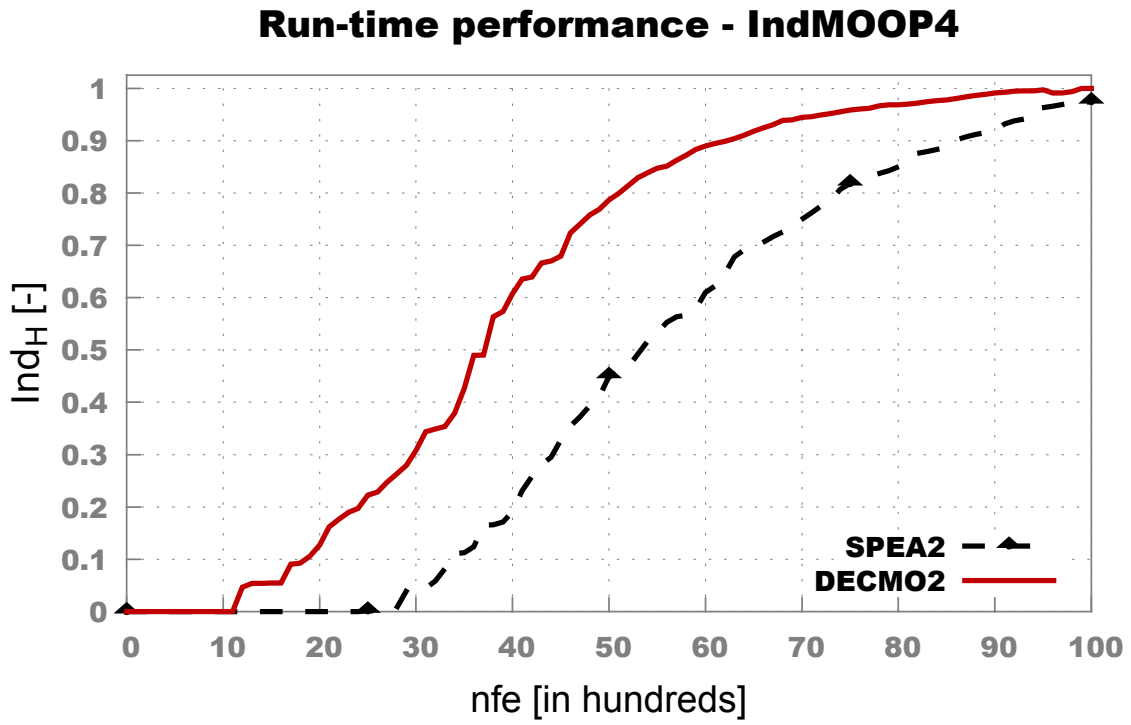


Figure 6.11: Run-time Ind_H-measured performance of DECMO2 and SPEA2 IndMOOP4.

6.3.7 Remarks Regarding the Proposed Algorithms

The two newly proposed coevolutionary algorithms described in this chapter, DECMO and DECMO2 are generally able to surpass the individual performance of their incorporated

evolutionary models when considering a quite wide (and diverse) set of benchmark MOOPs and fixed parameterizations.

The DECMO algorithm proves that even a basic coevolutionary approach is capable of enhancing the performance (robustness) of well-known and widely used MOEAs like SPEA2, DEMO and GDE3. Furthermore, the performed tests have shown that when considering optimization runs where one may perform 5000 to 35000 *nfe*, DECMO is a very attractive choice because of its structural simplicity.

The results of the reported individual (i.e., one-on-one) and batch comparative convergence comparisons strongly indicate that **DECMO2** is a much improved version of its predecessor and that this (somewhat structurally complex) coevolutionary approach **is able to successfully compete with** a very well known (**state-of-the-art**) MOEA like **MOEA/D-DE** across a wide range of artificial MOOPs. The highlighted construct is not a measure of false modesty but a reminder that all the presented results are based on the Ind_H quality indicator and that they should be regarded in combination with the appropriate interpretation of Ind_H monotonicity discussed in Section 3.2.2. According to this, the strongest definite statement that we can make based on the presented results is that, given the 25 benchmark problems: on average, DECMO2 is not a worse a posteriori MOO solver than any of the other tested algorithms during a large part[12] of the optimization run. Nevertheless, based on the same results, the previous definite statement can not be made for any of the other analyzed MOEAs. Therefore, one can *weakly argue* that, on average, DECMO2 is the best (*safest*) choice among all the considered algorithms. As a direct consequence, a DM (confronted with a CIMOOP) should be more likely to prefer DECMO2 as a the first option when searching for a robust and efficient black-box a posteriori MOO method[13].

[12] For example, when considering any $nfe \leq 40000$.

[13] If (s)he considers that the general optimization performance displayed on the 25 artificial MOOPs is relevant for the real-life MOOPs (s)he aims to solve.

Chapter 7

Cumulative Results

7.1 Combining the Enhancements

The results presented during the three previous chapters show that, taken separately, all the MOEA-intended enhancements we have proposed[1] are able to achieve their goal of generally improving run-time duration and final solution (i.e., *PN*) quality.

Nevertheless, in the introductory chapter of this thesis, we stated that our main idea when choosing to investigate three different (but complementary) MOEA improvements was to ensure that we could obtain powerful hybrid MOEAs that can effectively tackle complicated CIMOOPs from different angles. Thus, the motivation for combining the three proposed enhancements is very straightforward: if one particular enhancement (improvement strategy) cannot deliver good results on a certain CIMOOP, we hope that its poor performance can be compensated by (a combination of) other improvements.

In order to offer a basic example into this matter, let us consider the parallel / distributed implementation of the DECMO algorithm. Since the two coevolved subpopulations of DECMO are both using $(\mu + \lambda)$ MOO evolutionary models (implemented via the environmental selection operator), one can choose to parallelize them via a generational or a steady-state asynchronous MSP scheme. In light of the arguments from Section 5.4, given our optimization setup for CIMOOPs, the SSA-MSPS seems more attractive. The modifications required by DECMO[2] to enforce such a parallelization scheme are minimal as each subpopulation can independently manage its own asynchronous evolutionary process and:

- after every evaluation of at least p_{size} new individuals in subpopulation P and at least q_{size} new individuals in population Q, we apply a weak fitness sharing stage (i.e., a random copy of one individual from the complementary subpopulation);
- after every evaluation of at least $t_{sha} \times p_{size}$ individuals in P and at least $t_{sha} \times q_{size}$ individuals in Q, we apply the strong fitness sharing stage that requires the construction

[1] In the order of their introduction, these are: (i) global surrogate-based evolution, (II) steady-state parallelization and (III) symbiotic cooperative coevolution of multiple MOEAs.

[2] As it is described in Algorithm 3.

of the global elite subset.

In order to simplify referencing, we shall call this version of our basic coevolutionary MOEA: **SSA-DECMO**. It is worthy to mention that, when running SSA-DECMO on our computer cluster environment, we actually end up with a hybrid parallelization strategy that combines:

- an *island model paradigm* on the broader level – i.e., when considering the limited interactions between the two coevolved subpopulations P and Q;
- *SSA-MSPS* on the finer level – i.e., inside each of the coevolved components.

It is important to note that inside this basic island model topology inherently generated by the parallelization of DECMO, the dual fitness sharing mechanism completely defines the required *migration policy*.

SSA-DECMO can also be easily improved by integrating on-the-fly surrogate modeling. We shall refer to the obtained approach as **SSA-DECMO-SE**. Considering the most important findings from Chapter 4, SSA-DECMO-SE is obtained by integrating *surrogate-based optimization blocks* inside the main loop of SSA-DECMO. Each such block is represented by the successive application of a *surrogate model construction* stage a *surrogate-based MOEA execution* stage and a *FE-based re-evaluation* stage. In the re-evaluation stage of SSA-DECMO-SE blocks we apply the FE-based ranking on a preset number of the most promising[3] surrogate-discovered designs. At the end of a surrogate-based optimization block:

1. every discovered feasible individual is considered for a possible update of either the P or Q subpopulations (the choice is random);
2. the SSA-DECMO-SE may continue with a standard FE-based optimization or with another surrogate-based optimization block (depending on the DM's preference).

It is important to note that the ability to integrate multiple surrogate-based optimization blocks during the run is owed to the very time-efficient surrogate construction strategies introduced in Section 4.4. Furthermore, the approach of using the surrogate-based enhancement on-demand, according to the DM's preferences and observations, slightly pushes SSA-DECMO-SE towards the interactive MOO paradigm. If instead one chooses to employ very often (automatic / linked) short surrogate-based optimization blocks, SSA-DECMO-SE could also be seen as moving towards a (rudimentary) on-line learning strategy (aimed to permanently update the surrogates as new information becomes available).

7.2 Two Summarizing Plots

In order to show how all three enhancements can be combined in order to seriously improve the ability of MOEAs to tackle very complicated optimization scenarios, we present the

[3] We apply a standard Pareto-based ranking (i.e., selection for survival) strategy.

results of optimization runs on the most difficult industrial CIMOOPs we tested with: IndMOOP5. This problem is not only difficult because it has 22 variables and 4 objectives. When considering the performance evaluation process from Figure 3.2, IndMOOP5 requires 18 current-load simulations and the (non-parallel) post-processing phase is also computationally intensive and affects the achievable parallelization ratios. Furthermore, the problem also suffers from formalization constraints and, as shown by IndMOOP3 in Section 4.3.2, this can seriously impact the performance of global surrogate models. The combined effect of all these features is that, when using a population size of 100, the generational version of SPEA2(not: GEN-SPEA2) required, on average, 350 hours (wall-clock time) to evolve 100 generations[4]. In light of all these considerations, one would expect that no standalone enhancement is able to ensure a very large improvement. Nevertheless, a combination of two or all three enhancements might prove more successful.

In order to show the progressive impact of the various enhancements proposed throughout this thesis, we also applied a *steady-state asynchronous* version of SPEA2 (not: SSA-SPEA2) with the same population size of 100 and SSA-DECMO and SSA-DECMO-SE with a population size of 200. Thus, SSA-SPEA2 proposes the most basic "enhancement" – i.e., applying an asynchronous steady-state evolutionary model instead of a generational one. SSA-DECMO improves on this and adds a coevolutionary enhancement that introduces a DE-based paradigm aimed at improving the performance of the SSA-SPEA2 approach. SSA-DECMO-SE finally adds (on-demand) surrogate modeling to the mix.

All four algorithms were parameterized in a standard way and in the case of SSA-DECMO-SE we performed six *surrogate-based optimization blocks*: after 500, 1500, 2500, 3500, 4500 and 5500 fitness evaluations. We always used *Pareto-trimmed training sets* of at most 1000 individuals and we also constructed *surrogate feasibility models* (as described in Section 4.4.3) in order to alleviate the effect of the formalization constraints during the surrogate-based optimization runs. At the end of each surrogate-based optimization cycle we re-evaluated the best 1000 individuals discovered during the surrogate-based search. The surrogate-based optimizations were carried out using SSA-DECMO (with an identical parameterization) and were allowed to generate 10000 individuals. Two of the four objectives were very difficult to model, as the best global MLP surrogates we obtained only achieved a training R^2 of 0.80. The decision not to continue with surrogate-based optimization cycles after the *nfe*=6500 was influenced by the observation that the improvements delivered by this enhancement were becoming smaller and smaller. As such for the last 3500 allowed fitness evaluations SSA-DECMO-SE fell back to the standard SSA-DECMO evolutionary model on which it is based.

We performed three independent tests with each method and (in order to obtain more comparable results) we started each MOEA run with the same, LHS-generated population of 100 individuals. The comparative *nfe* and time-wise **MOEA convergence plots** are **presented in Figure 7.1**. While we do strongly feel that these performance plots **are self-explanatory and form a perfect showcase for the main ideas and concepts presented throughout this thesis**, two things must be pointed out explicitly:

1. all three enhanced methods are eventually able to discover final *PN*s that are at least

[4] All fitness evaluations were distributed over the HTCondor™-managed cluster computing environment

as good (i.e., Ind_H=0.9) as those obtained by the baseline MOEA (i.e., GEN-SPEA2);

2. the MOEA that incorporates all three enhancements (i.e., SSA-DECMO-SE) finds *PN*s with Ind_H=0.9 approximately six days (140 hours) faster than GEN-SPEA2 and is able to discover *PN*s with a significantly better (Ind_H-estimated) quality than all its counterparts.

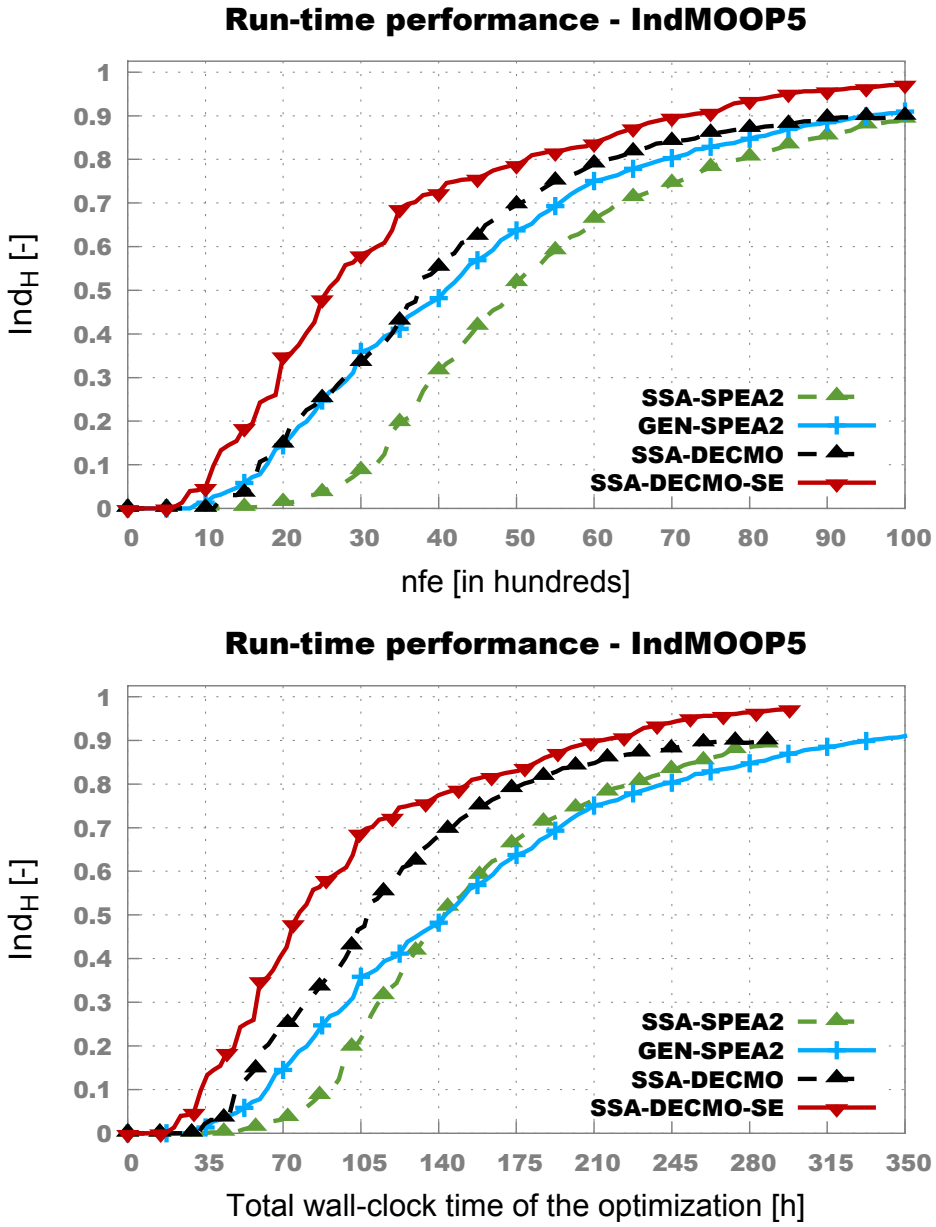


Figure 7.1: Ind_H-measured comparative convergence of a standard MOEA (i.e., GEN-SPEA2) and of three iteratively enhanced MOEAs: SSA-MSPS, SSA-DECMO and SSA-DECMO-SE.

"Perfection is a stick with which to beat the possible."

Rebecca Solnit

Chapter 8

Conclusions

8.1 Achievements

In the present thesis we have proposed several ideas aimed at enhancing the performance of (standard) multi-objective evolutionary algorithms (MOEAs) applied on computationally-intense problems. Our improvements are structured along three complementary research lines:

1. In Chapter 4 we systematically introduced a procedure for generating on-the-fly (global) surrogate models capable of alleviating the dependency of the MOEA on the computationally-intensive original fitness evaluation mechanism.

2. In Chapter 5 we investigated which master-slave parallelization schema is more likely to help a $(\mu + \lambda)$-styled MOEA obtain better performance given the particularities of the considered multi-objective optimization process. The results of this analysis are very helpful for practitioners of MOEA as they offer valuable insight with regard to a simple decision that can significantly reduce the total run-time of the optimizations without affecting final solution quality.

3. In Chapter 6 we proposed two new algorithms that apply a symbiotic cooperative-coevolutionary paradigm in order to efficiently integrate the search strategies of multiple MOEAs in a single run. Across a wide set of artificial test problems, the obtained coevolutionary multi-objective optimization solvers are shown to be both robust to parameterization and highly competitive when comparing with state-of-the-art approaches.

Across the thesis, the individual and combined effectiveness of our proposed enhancements is empirically tested and confirmed on five real-life computationally-intensive optimization problems (CIMOOPs) from the field of electrical drive engineering.

As a byproduct of our research endeavors, we have also developed a new racing-based methodology for easily measuring (and reporting) the comparative convergence performance (behavior) of MOEAs over large benckmark problem sets. This methodology is introduced in Section 3.3 and extensively used throughout Chapter 6.

All in all, we consider that through the present work we have introduced at least a few novelties that can benefit the field of MOEA research in general and the work of MOEA practitioners dealing with CIMOOPs in particular.

8.2 Outlook

Far from contesting the importance of visionary thinking inside the research world, we believe that the epigraph of this chapter describes quite well the complementary state that generally characterizes a scientists' attitude towards his own achievements and towards the results obtained by others.

With regard to the work discussed in this thesis, we would primarily like to extend the presented coevolutionary approaches by reshaping them as a general framework capable of integrating even more multi-objective optimization methods. The convergence behavior of the different MOEAs we have tested during our experiments indicates that there is a huge improvement potential in applying DECMO2 during the initial phase of the optimization runs and switching to MOEA/D-DE after that. The hybridization of the two search strategies should be facilitated by the fact that both maintain a decomposition-based population (archive). More advanced parallelization methods, better suited for the resulting hybrid MOEA, also warrant a close examination.

Concerning the surrogate-based enhancements, future plans are centered around the investigation of on-line learning paradigms suitable for global surrogate modeling and on the possible integration of pre-selection strategies and simpler local models.

Generally, we consider that significant improvements in the field of CIMOOP can be discovered by hybridizing state-of-the-art non-deterministic solvers either with proven classical strategies (like those presented in Section 2.2) or with (domain-adapted) machine learning and data mining surrogate construction methods.

Bibliography

[1] A.-C. Zăvoianu, E. Lughofer, G. Bramerdorfer, W. Amrhein, E. P. Klement, DECMO2: a robust hybrid and adaptive multi-objective evolutionary algorithm, Soft Computing [available online] (2014) 1–19.

[2] A.-C. Zăvoianu, G. Bramerdorfer, E. Lughofer, S. Silber, W. Amrhein, E. P. Klement, A hybrid soft computing approach for optimizing design parameters of electrical drives, in: V. Snášel, A. Abraham, E. S. Corchado (Eds.), Advances in Intelligent Systems and Computing, Vol. 188 of Advances in Intelligent Systems and Computing, Springer Berlin Heidelberg, 2013, pp. 347–358.

[3] A.-C. Zăvoianu, G. Bramerdorfer, E. Lughofer, S. Silber, W. Amrhein, E. P. Klement, Hybridization of multi-objective evolutionary algorithms and artificial neural networks for optimizing the performance of electrical drives, Engineering Applications of Artificial Intelligence 26 (8) (2013) 1781–1794.

[4] A.-C. Zăvoianu, E. Lughofer, G. Bramerdorfer, W. Amrhein, E. P. Klement, An effective ensemble-based method for creating on-the-fly surrogate fitness functions for multi-objective evolutionary algorithms, in: Proceedings of the 15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC 2013), IEEE Computer Society Conference Publishing Services (CPS), 2014, pp. 237–244.

[5] A.-C. Zăvoianu, E. Lughofer, W. Koppelstätter, G. Weidenholzer, W. Amrhein, E. P. Klement, On the performance of master-slave parallelization methods for multi-objective evolutionary algorithms, in: L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. A. Zadeh, J. M. Zurada (Eds.), Artificial Intelligence and Soft Computing, Vol. 7895 of Lecture Notes in Artificial Intelligence, Springer Berlin Heidelberg, 2013, pp. 122–134.

[6] A.-C. Zăvoianu, E. Lughofer, W. Amrhein, E. P. Klement, Efficient multi-objective optimization using 2-population cooperative coevolution, in: Computer Aided Systems Theory - EUROCAST 2013, Lecture Notes in Computer Science, Springer Berlin Heidelberg, 2013, pp. 251–258.

[7] K. Miettinen, Nonlinear Multiobjective Optimization, Kluwer Academic Publishers, 1999.

[8] V. Pareto, Manuale di Economia Politica, Societa Editrice Libraria, 1906, Translated into English by A.S. Schwier as Manual of Political Economy, Macmillan, New York, 1971.

[9] F. Y. A. Edgeworth, Mathematical psychics: An essay on the application of mathematics to the moral sciences, C. Keagann Paul, 1881.

[10] W. Stadler, Initiators of multicriteria optimization, in: Recent Advances and Historical Development of Vector Optimization, Springer, 1987, pp. 3–47.

[11] A. Smith, An Inquiry into the Nature and Causes of the Wealth of Nations, W. Strahan and T. Cadell, London, 1776.

[12] H. Khun, A. Tucker, Nonlinear programming, in: J. Neyman (Ed.), Proceedings of the Second Berkeley Symposium on Mathematical Statistics and Probability, University of California Press, Berkeley, California, 1951, pp. 481–492.

[13] C. Coello Coello, G. Lamont, D. Van Veldhuisen, Evolutionary Algorithms for Solving Multi-Objective Problems, Genetic and Evolutionary Computation Series, Springer, 2007.

[14] T. C. Koopmans, Analysis of production as an efficient combination of activities, in: T. C. Koopmans (Ed.), Activity Analysis of Production and Allocation, Cowles Commision Monograph No. 13, John Wiley & Sons, New York, 1951, pp. 33–97.

[15] S. A. Marglin, Public Investment Criteria, Allen & Unwin, London, 1967.

[16] L. Zadeh, Optimality and non-scalar-valued performance criteria, Automatic Control, IEEE Transactions on 8 (1) (1963) 59–60.

[17] W. Stadler, Preference optimality and applications of Pareto-optimality, in: G. Leitmann, A. Marzollo (Eds.), Multicriteria Decision Making, Springer, 1975, pp. 125–225.

[18] J. L. Cohon, Multiobjective Programming and Planning, Academic Press, 1978.

[19] M. Ehrgott, Multicriteria optimization, 2nd Edition, Vol. 2, Springer, Berlin, 2005.

[20] J. L. Cohon, D. H. Marks, A review and evaluation of multiobjective programing techniques, Water Resources Research 11 (2) (1975) 208–220.

[21] C.-L. Hwang, S. R. Paidy, K. Yoon, A. S. M. Masud, Mathematical programming with multiple objectives: A tutorial, Computers & Operations Research 7 (1) (1980) 5–31.

[22] J. Andersson, Multiobjective optimization in engineering design applications to fluid power systems, Ph.D. thesis, Linköping University (2001).

[23] R. T. Marler, J. S. Arora, Survey of multi-objective optimization methods for engineering, Structural and multidisciplinary optimization 26 (6) (2004) 369–395.

[24] A. Osyczka, An approach to multicriterion optimization problems for engineering design, Computer Methods in Applied Mechanics and Engineering 15 (3) (1978) 309–333.

[25] B. F. Hobbs, A comparison of weighting methods in power plant siting, Decision Sciences 11 (4) (1980) 725–737.

[26] J. Rao, N. Roy, Fuzzy set theoretic approach of assigning weights to objectives in multieriteria decision making, International Journal of Systems Science 20 (8) (1989) 1381–1386.

[27] A. Osyczka, Multicriterion optimization in engineering with FORTRAN programs, Ellis Horwood series in mechanical engineering, E. Horwood, 1984.

[28] D. Bestie, P. Eberhard, Dynamic system design via multicriteria optimization, in: Multiple Criteria Decision Making, Springer, 1997, pp. 467–478.

[29] P. C. Fishburn, Lexicographic orders, utilities and decision rules: A survey, Management science 20 (11) (1974) 1442–1471.

[30] A. Charnes, W. W. Cooper, R. O. Ferguson, Optimal estimation of executive compensation by linear programming, Management science 1 (2) (1955) 138–151.

[31] A. Charnes, W. W. Cooper, Management Models and Industrial Applications of Linear Programming, Vol. 1, John Wiley and Sons Inc., New York, 1961.

[32] D. R. Wallace, M. J. Jakiela, W. C. Flowers, Design search under probabilistic specifications using genetic algorithms, Computer-Aided Design 28 (5) (1996) 405–421.

[33] A. P. Wierzbicki, The use of reference objectives in multiobjective optimization, in: G. Fandel, T.Gal (Eds.), Multiple Criteria Decision Making Theory and Applications, Vol. 177 of Lecture Notes in Economics and Mathematical Systems, Springer-Verlag Berlin Heidelberg, 1980, pp. 468–486.

[34] R. Steuer, Multiple Criteria Optimization: Theory, Computation, and Applications, John Wiley & Sons, 1986.

[35] Y. Y. Haimes, L. Ladson, D. A. Wismer, Bicriterion formulation of problems of integrated system identification and system optimization, IEEE Transactions on Systems Man and Cybernetics 1 (1971) 296.

[36] I. Das, J. E. Dennis, Normal-boundary intersection: A new method for generating the Pareto surface in nonlinear multicriteria optimization problems, SIAM Journal on Optimization 8 (3) (1998) 631–657.

[37] A. Messac, A. Ismail-Yahaya, C. A. Mattson, The normalized normal constraint method for generating the Pareto frontier, Structural and Multidisciplinary Optimization 25 (2) (2003) 86–98.

[38] C. Darwin, The Origin of Species By Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life., Murray, London, 1859.

[39] R. Dawkins, The blind watchmaker: Why the evidence of evolution reveals a universe without design, WW Norton & Company, 1996.

[40] A. M. Turing, Computing machinery and intelligence, Mind 59 (1950) 433–460.

[41] J. H. Holland, Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence., University of Michigan Press, 1975.

[42] I. Rechenberg, The evolution strategy. A mathematical model of darwinian evolution, in: E. Frehland (Ed.), Synergetics - from Microscopic to Macroscopic Order, Springer, 1984, pp. 122–132.

[43] H.-P. Schwefel, Numerical optimization of computer models, John Wiley & Sons, Inc., 1981.

[44] J. R. Koza, Genetic programming: on the programming of computers by means of natural selection, MIT press, 1992.

[45] J. H. Holland, Genetic algorithms, Scientific American 267 (1) (1992) 66–72.

[46] D. Dasgupta, Z. Michalewicz, Evolutionary algorithms in engineering applications, Springer, 1997.

[47] C. A. Coello Coello, G. B. Lamont, Applications of multi-objective evolutionary algorithms, World Scientific, 2004.

[48] W. Paszkowicz, Genetic algorithms, a nature-inspired tool: Survey of applications in materials science and related fields, Materials and Manufacturing Processes 24 (2) (2009) 174–197.

[49] W. Paszkowicz, Genetic algorithms, a nature-inspired tool: A survey of applications in materials science and related fields: Part II, Materials and Manufacturing Processes 28 (7) (2013) 708–725.

[50] D. E. Goldberg, Genetic algorithms in search, optimization, and machine learning, Addison-Wesley, Reading, MA, 1989.

[51] M. Mitchell, An Introduction to Genetic Algorithms, 5th Edition, A Bradford Book The MIT Press, Cambridge, Massachusetts - London, England, 1999.

[52] V. K. Koumousis, C. P. Katsaras, A saw-tooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance, Evolutionary Computation, IEEE Transactions on 10 (1) (2006) 19–28.

[53] M. Affenzeller, S. Wagner, S. Winkler, Self-adaptive population size adjustment for genetic algorithms, in: Computer Aided Systems Theory–EUROCAST 2007, Springer, 2007, pp. 820–828.

[54] R. S. Rosenberg, Simulation of genetic populations with biochemical properties, Ph.D. thesis, University of Michigan, Ann Harbor, Michigan (1967).

[55] J. D. Schaffer, Multiple objective optimization with vector evaluated genetic algorithms, in: Proceedings of the 1st international Conference on Genetic Algorithms, L. Erlbaum Associates Inc., 1985, pp. 93–100.

[56] D. E. Goldberg, J. Richardson, Genetic algorithms with sharing for multimodal function optimization, in: Genetic algorithms and their applications: Proceedings of the Second International Conference on Genetic Algorithms, Hillsdale, NJ: Lawrence Erlbaum, 1987, pp. 41–49.

[57] D. A. Van Veldhuizen, Multiobjective evolutionary algorithms: Classifications, Analyses, and New Innovations, Ph.D. thesis, Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio (1999).

[58] C. M. Fonseca, P. J. Fleming, Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization., in: S. Forrest (Ed.), Proceedings of the Fifth International Conference on Genetic Algorithms, Morgan Kauffman Publishers, 1993, pp. 416–423.

[59] J. Horn, N. Nafpliotis, D. E. Goldberg, A niched pareto genetic algorithm for multiobjective optimization, in: Proceedings of the First IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence, IEEE Service Center, 1994, pp. 82–87.

[60] N. Srinivas, K. Deb, Muiltiobjective optimization using nondominated sorting in genetic algorithms, Evolutionary computation 2 (3) (1994) 221–248.

[61] E. Zitzler, L. Thiele, Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach, IEEE Transactions on Evolutionary Computation 3 (4) (1999) 257–271.

[62] G. Rudolph, A. Agapie, Convergence properties of some multi-objective evolutionary algorithms, in: IEEE Congress on Evolutionary Computation (CEC 2000), IEEE Press, 2000, pp. 1010–1016.

[63] J. D. Knowles, D. W. Corne, Approximating the nondominated front using the Pareto archived evolution strategy, Evolutionary Computation 8 (2) (2000) 149–172.

[64] C. A. Coello Coello, 20 years of evolutionary multiobjective optimization: What has been done and what remains to be done, in: Computational Intelligence: Principles and Practice, IEEE Computational Intelligence Society, 2006, pp. 73–88.

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

[66] E. Zitzler, M. Laumanns, L. Thiele, SPEA2: Improving the strength Pareto evolutionary algorithm for multiobjective optimization, in: Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), International Center for Numerical Methods in Engineering (CIMNE), 2002, pp. 95–100.

[67] K. Deb, R. B. Agrawal, Simulated binary crossover for continuous search space, Complex Systems 9 (1995) 115–148.

[68] K. Deb, M. Goyal, A combined genetic adaptive search (GeneAS) for engineering design, Computer Science and Informatics 26 (1996) 30–45.

[69] K. Deb, Multi-Objective Optimization using Evolutionary Algorithms, John Wiley & Sons, 2001.

[70] R. Storn, K. V. Price, Differential evolution - a simple and effcient heuristic for global optimization over continuous spaces, Journal of Global Optimization 11 (4) (1997) 341–359.

[71] J. Vesterstrom, R. Thomsen, A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems, in: IEEE Congress on Evolutionary Computation (CEC 2004), IEEE Press, 2004, pp. 1980–1987.

[72] T. Robič, B. Filipič, DEMO: Differential evolution for multiobjective optimization, in: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2005), Springer, Springer Berlin / Heidelberg, 2005, pp. 520–533.

[73] S. Kukkonen, J. Lampinen, GDE3: The third evolution step of generalized differential evolution, in: IEEE Congress on Evolutionary Computation (CEC 2005), IEEE Press, 2005, pp. 443–450.

[74] S. Kukkonen, J. Lampinen, Performance assessment of Generalized Differential Evolution 3 with a given set of constrained multi-objective test problems, in: IEEE Congress on Evolutionary Computation (CEC 2009), IEEE Press, 2009, pp. 1943–1950.

[75] A. Jaszkiewicz, On the performance of multiple-objective genetic local search on the 0/1 knapsack problem - A comparative experiment, IEEE Transactions on Evolutionary Computation 6 (4) (2002) 402–412.

[76] Q. Zhang, H. Li, MOEA/D: A multi-objective evolutionary algorithm based on decomposition, IEEE Transactions on Evolutionary Computation 11 (6) (2007) 712–731.

[77] H. Li, Q. Zhang, Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II, IEEE Transactions on Evolutionary Computation 13 (2) (2009) 284–302.

[78] Q. Zhang, W. Liu, H. Li, The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances, Tech. rep., School of CS & EE, University of Essex (February 2009).

[79] D. Angus, C. Woodward, Multiple objective ant colony optimisation, Swarm intelligence 3 (1) (2009) 69–85.

[80] M. Reyes-Sierra, C. A. Coello Coello, Multi-objective particle swarm optimizers: A survey of the state-of-the-art, International journal of computational intelligence research 2 (3) (2006) 287–308.

[81] K. C. Tan, C. K. Goh, A. Mamun, E. Ei, An evolutionary artificial immune system for multi-objective optimization, European Journal of Operational Research 187 (2) (2008) 371–392.

[82] S. Bandyopadhyay, S. Saha, U. Maulik, K. Deb, A simulated annealing-based multi-objective optimization algorithm: Amosa, Evolutionary Computation, IEEE Transactions on 12 (3) (2008) 269–283.

[83] F. Bolognini, A. A. Seshia, K. Shea, Exploring the application of multidomain simulation-based computational synthesis methods in MEMS design, in: ICED 07 International Conference on Engineering Design, 2007.

[84] J. H. Holland, Building blocks, cohort genetic algorithms, and hyperplane-defined functions, Evolutionary Computation 8 (4) (2000) 373–391.

[85] F. Kursawe, A variant of evolution strategies for vector optimization, in: Workshop on Parallel Problem Solving from Nature (PPSN I), Vol. 496 of Lecture Notes in Computer Science, Springer, 1991, pp. 193–197.

[86] E. Zitzler, K. Deb, L. Thiele, Comparison of multiobjective evolutionary algorithms: Empirical results, Evolutionary Computation 8 (2) (2000) 173–195.

[87] K. Deb, L. Thiele, M. Laumanns, E. Zitzler, Scalable multi-objective optimization test problems, in: IEEE Congress on Evolutionary Computation (CEC 2002), IEEE Press, 2002, pp. 825–830.

[88] S. Huband, L. Barone, L. While, P. Hingston, A scalable multi-objective test problem toolkit, in: Evolutionary Multi-Criterion Optimization (EMO 2005), Vol. 3410 of Lecture Notes in Computer Science, 2005, pp. 280–295.

[89] J. T. Richardson, M. R. Palmer, G. E. Liepins, M. Hilliard, Some guidelines for genetic algorithms with penalty functions, in: Proceedings of the third international conference on Genetic algorithms, Morgan Kaufmann Publishers Inc., 1989, pp. 191–197.

[90] Z. Michalewicz, M. Schoenauer, Evolutionary algorithms for constrained parameter optimization problems, Evolutionary Computation 4 (1) (1996) 1–32.

[91] D. Whitley, S. Rana, J. Dzubera, K. E. Mathias, Evaluating evolutionary algorithms, Artificial intelligence 85 (1) (1996) 245–276.

[92] T. Weise, R. Chiong, K. Tang, Evolutionary optimization: Pitfalls and booby traps, Journal of Computer Science and Technology 27 (5) (2012) 907–936.

[93] T. Okabe, Y. Jin, M. Olhofer, B. Sendhoff, On test functions for evolutionary multi-objective optimization, in: Parallel Problem Solving from Nature-PPSN VIII, Springer, 2004, pp. 792–802.

[94] FEMAG - The finite-element program for analysis and simulation of electrical machines and devices, Website http://people.ee.ethz.ch/∼ femag/englisch/index_en.htm.

[95] J. J. Durillo, A. J. Nebro, E. Alba, The jMetal framework for multi-objective optimization: Design and architecture, in: IEEE Congress on Evolutionary Computation (CEC 2010), 2010, pp. 1–8.

[96] J. Knowles, L. Thiele, E. Zitzler, A tutorial on the performance assessment of stochastic multiobjective optimizers, Tech. Rep. 214, Computer Engineering and Networks Laboratory (TIK), ETH Zürich (2006).

[97] D. Van Veldhuizen, G. Lamont, Multiobjective evolutionary algorithm research: A history and analysis, tech. rep. tr-98-03, Tech. rep., Dept. Elec. Comput. Eng., Graduate School of Eng., Air Force Inst. Technol., Wright-Patterson, AFB, OH (1998).

[98] A. Zhou, Y. Jin, Q. Zhang, B. Sendhoff, E. Tsang, Combining model-based and genetics-based offspring generation for multi-objective optimization using a convergence criterion, in: IEEE Congress on Evolutionary Computation (CEC 2006), 2006, pp. 892–899.

[99] E. Zitzler, Evolutionary algorithms for multiobjective optimization: Methods and applications, Ph.D. thesis, Swiss Federal Institute of Technology (1999).

[100] M. Fleischer, The measure of Pareto optima: Applications to multi-objective metaheuristics, in: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2003), Springer, 2003, pp. 519–533.

[101] E. Zitzler, L. Thiele, M. Laumanns, C. M. Fonseca, V. G. da Fonseca, Performance assessment of multiobjective optimizers: An analysis and review, IEEE Transactions on Evolutionary Computation 7 (2) (2003) 117–132.

[102] E. Zitzler, L. Thiele, Multiobjective optimization using evolutionary algorithms - a comparative case study, in: Parallel problem solving from nature - PPSN V, Springer, 1998, pp. 292–301.

[103] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, IEEE Transactions on Evolutionary Computation 1 (1) (1997) 67–82.

[104] H. B. Mann, D. R. Whitney, On a test of whether one of two random variables is stochastically larger than the other, The annals of mathematical statistics 18 (1) (1947) 50–60.

[105] L. V. Santana-Quintero, A. A. Montaño, C. A. C. Coello, A review of techniques for handling expensive functions in evolutionary multi-objective optimization, in: Y. Tenne, C.-K. Goh, L. M. Hiot, Y. S. Ong (Eds.), Computational Intelligence in Expensive Optimization Problems, Vol. 2 of Adaptation, Learning, and Optimization, Springer, 2010, pp. 29–59.

[106] N. Queipo, R. Haftka, W. Shyy, T. Goel, R. Vaidyanathan, P. Kevin Tucker, Surrogate-based analysis and optimization, Progress in Aerospace Sciences 41 (1) (2005) 1–28.

[107] A. Forrester, A. Sobester, A. Keane, Engineering Design via Surrogate Modelling: A Practical Guide, Wiley, 2008.

[108] Y. Tenne, C. Goh (Eds.), Computational Intelligence in Expensive Optimization Problems, Vol. 2 of Adaptation, Learning and Optimization, Springer, 2010.

[109] M. Pilát, Evolutionary algorithms for multiobjective optimization, Ph.D. thesis, Charles University in Prague, Institute of Computer Science, AS CR (2013).

[110] I. Loshchilov, M. Schoenauer, M. Sebag, A mono surrogate for multiobjective optimization, in: Proceedings of the 12th annual Conference on Genetic and Evolutionary Computation (GECCO), ACM, 2010, pp. 471–478.

[111] M. Pilát, R. Neruda, ASM-MOMA: Multiobjective memetic algorithm with aggregate surrogate model, in: IEEE Congress on Evolutionary Computation (CEC 2011), 2011, pp. 1202–1208.

[112] I. Loshchilov, M. Schoenauer, M. Sebag, Dominance-based Pareto-surrogate for multi-objective optimization, in: Simulated Evolution and Learning, Springer, 2010, pp. 230–239.

[113] M. Pilát, R. Neruda, Local meta-models for asm-moma, in: Bio-Inspired Computing and Applications, Springer, 2012, pp. 79–84.

[114] D. Stephens, D. Gorissen, K. Crombecq, T. Dhaene, Surrogate based sensitivity analysis of process equipment, Applied Mathematical Modelling 35 (4) (2011) 1676–1687.

[115] M. D. McKay, R. J. Beckman, W. J. Conover, Comparison of three methods for selecting values of input variables in the analysis of output from a computer code, Technometrics 21 (2) (1979) 239–245.

[116] I. Voutchkov, A. Keane, Multi-objective optimization using surrogates, in: Computational Intelligence in Optimization, Springer, 2010, pp. 155–175.

[117] S. Haykin, Neural Networks: A Comprehensive Foundation, 2nd Edition, Prentice Hall Inc., 1999.

[118] R. Collobert, S. Bengio, SVMTorch: Support vector machines for large-scale regression problems, Journal of Machine Learning Research 1 (2001) 143–160.

[119] M. Buhmann, Radial Basis Functions: Theory and Implementations, Cambridge University Press, 2003.

[120] C. Bishop, Pattern Recognition and Machine Learning, Information Science and Statistics, Springer, 2006.

[121] J. Park, I. W. Sandberg, Universal approximation using radial-basis-function networks, Neural computation 3 (2) (1991) 246–257.

[122] D. Aha, D. Kibler, M. Albert, Instance-based learning algorithms, Machine learning 6 (1) (1991) 37–66.

[123] Y. Jin, M. Hüsken, M. Olhofer, B. Sendhoff, Neural networks for fitness approximation in evolutionary optimization, in: Y. Jin (Ed.), Knowledge Incorporation in Evolutionary Computation, Studies in Fuzziness and Soft Computing, Springer, 2004, pp. 281–305.

[124] Y.-S. Hong, H. Lee, M.-J. Tahk, Acceleration of the convergence speed of evolutionary algorithms using multi-layer neural networks, Engineering Optimization 35 (1) (2003) 91–102.

[125] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks 2 (5) (1989) 359–366.

[126] M. Paliwal, U. A. Kumar, Neural networks and statistical techniques: A review of applications, Expert Systems with Applications 36 (1) (2009) 2–17.

[127] C. N. Gupta, R. Palaniappan, S. Swaminathan, S. M. Krishnan, Neural network classification of homomorphic segmented heart sounds, Applied Soft Computing 7 (1) (2007) 286–297.

[128] A. Wefky, F. Espinosa, A. Prieto, J. Garcia, C. Barrios, Comparison of neural classifiers for vehicles gear estimation, Applied Soft Computing 11 (4) (2011) 3580–3599.

[129] P. J. Werbos, Beyond regression: New tools for prediction and analysis in the behavioral sciences, Ph.D. thesis, Harvard University (1974).

[130] T. Hastie, R. Tibshirani, J. Friedman, The Elements of Statistical Learning: Data Mining, Inference and Prediction, 2nd Edition, Springer, 2009.

[131] P. S. Churchland, T. J. Sejnowski, The Computational Brain, MIT Press, 1992.

[132] G. Cybenko, Approximation by superpositions of a sigmoidal function, Mathematics of Control, Signals, and Systems 2 (4) (1989) 303–314.

[133] L. Breiman, J. Friedman, R. Olshen, C. Stone, Classification and Regression Trees, Chapman and Hall / CRC, 1993.

[134] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I. H. Witten, The WEKA data mining software: an update, SIGKDD Explorations 11 (1) (2009) 10–18.

[135] D. Thain, T. Tannenbaum, M. Livny, Distributed computing in practice: the condor experience., Concurrency - Practice and Experience 17 (2-4) (2005) 323–356.

[136] F. Bittner, I. Hahn, Kriging-assisted multi-objective particle swarm optimization of permanent magnet synchronous machine for hybrid and electric cars, in: IEEE International Electric Machines & Drives Conference (IEMDC 2013), IEEE, 2013, pp. 15–22.

[137] M. L. Stein, Interpolation of spatial data: some theory for Kriging, Springer, 1999.

[138] P. K. Nain, K. Deb, A multi-objective optimization procedure with successive approximate models, Tech. Rep. 2005002, KanGAL (2005).

[139] D. E. Goldberg, K. Deb, A comparative analysis of selection schemes used in genetic algorithms, in: Foundations of Genetic Algorithms, Morgan Kaufmann, 1991, pp. 69–93.

[140] J. Durillo, A. Nebro, F. Luna, E. Alba, A study of master-slave approaches to parallelize NSGA-II, in: IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2008), 2008, pp. 1–8.

[141] M. Yagoubi, L. Thobois, M. Schoenauert, Asynchronous evolutionary multi-objective algorithms with heterogeneous evaluation costs, in: IEEE Congress on Evolutionary Computation (CEC 2011), 2011, pp. 21–28.

[142] J. Durillo, A. Nebro, F. Luna, E. Alba, On the effect of the steady-state selection scheme in multi-objective genetic algorithms, in: International Conference on Evolutionary Multi-Criterion Optimization (EMO 2009), Springer, 2009, pp. 183–197.

[143] D. H. Wolpert, W. G. Macready, Coevolutionary free lunches, Evolutionary Computation, IEEE Transactions on 9 (6) (2005) 721–735.

[144] D. H. Wolpert, What the fo free lunch theorems really mean; how to improve search algorithms, Tech. rep., Santa Fe Institute and Information Sciences Division (Los Alamos National Laboratory) (2012).

[145] M. Oltean, Searching for a practical evidence of the no free lunch theorems, in: Biologically Inspired Approaches to Advanced Information Technology, Springer, 2004, pp. 472–483.

[146] S. Christensen, F. Oppacher, What can we learn from no free lunch? a first attempt to characterize the concept of a searchable function, in: Proceedings of the Genetic and Evolutionary Computation Conference, Vol. 2001, 2001, pp. 1219–1226.

[147] R. Mallipeddi, P. N. Suganthan, Q.-K. Pan, M. F. Tasgetiren, Differential evolution algorithm with ensemble of parameters and mutation strategies, Applied Soft Computing 11 (2) (2011) 1679–1696.

[148] T.-P. Hong, H.-S. Wang, W.-C. Chen, Simultaneously applying multiple mutation operators in genetic algorithms, Journal of heuristics 6 (4) (2000) 439–455.

[149] S. Bandaru, R. Tulshyan, K. Deb, Modified sbx and adaptive mutation for real world single objective optimization, in: IEEE Congress on Evolutionary Computation (CEC 2011), 2011, pp. 1335–1342.

[150] J. N. Thompson, The coevolutionary process, University of Chicago Press, 1994.

[151] J. Paredis, Coevolutionary computation, Artificial life 2 (4) (1995) 355–375.

[152] C. D. Rosin, R. K. Belew, New methods for competitive coevolution, Evolutionary Computation 5 (1) (1997) 1–29.

[153] M. A. Potter, K. A. De Jong, Cooperative coevolution: An architecture for evolving coadapted subcomponents, Evolutionary computation 8 (1) (2000) 1–29.

[154] S. Luke, Essentials of Metaheuristics, 2nd Edition, Lulu, 2013, available for free at http://cs.gmu.edu/∼sean/book/metaheuristics/.

[155] K. Chellapilla, D. B. Fogel, Evolution, neural networks, games, and intelligence, Proceedings of the IEEE 87 (9) (1999) 1471–1496.

[156] D. B. Fogel, T. J. Hays, S. Hahn, J. Quon, A self-learning evolutionary chess program, Proceedings of the IEEE 92 (12) (2004) 1947–1954.

[157] J. J. Durillo, A. J. Nebro, JMETAL: A Java framework for multi-objective optimization, Advances in Engineering Software 42 (2011) 760–771.

[158] D. Zaharie, Influence of crossover on the behavior of differential evolution algorithms, Applied Soft Computing 9 (3) (2009) 1126–1138.

[159] D. Corne, J. Knowles, Some multiobjective optimizers are better than others, in: IEEE Congress on Evolutionary Computation (CEC 2003), Vol. 4, 2003, pp. 2506–2512.