

Shadow Sort – A Hybrid Non-dominated Sorting Algorithm

Nicholas Albert Tränkle



Thesis presented in partial fulfilment of the requirements for the
degree of Master of Engineering (Industrial Engineering) in the
Faculty of Engineering at Stellenbosch University

Supervisor: Prof JF Bekker

April 2022

Declaration

By submitting this thesis electronically, I declare that the entirety of the work contained therein is my own, original work, that I am the sole author thereof (save to the extent explicitly otherwise stated), that reproduction and publication thereof by Stellenbosch University will not infringe any third party rights and that I have not previously in its entirety or in part submitted it for obtaining any qualification.

Date: 1 April 2022

Copyright © 2022 Stellenbosch University

All rights reserved

Abstract

In this study a novel, hybrid, non-dominated sorting algorithm called Shadow Sort is presented. The algorithm was developed bearing in mind real-world, practical requirements of non-dominated sorting algorithms. The majority of non-dominated sorting algorithms are employed in conjunction with a multi-objective optimisation algorithm, many of which make use of population-based meta-heuristic techniques. The outputs of each population-based meta-heuristic iteration typically include hundreds, if not thousands of solutions, which need to be sorted in order to prime the next evolutionary iteration. However, of all the solutions proposed by any given meta-heuristic iteration, a relatively low number of those solutions are of any use. Therefore, Shadow Sort approaches all non-dominated sorting by eliminating dominated solutions as early as possible, rather than processing all solutions in order to find their respective Pareto set. Shadow Sort was developed in Matlab, and is firstly compared to other non-dominated sorting algorithms, namely Deductive Sort, Efficient Non-dominated Sort and Best Order Sort. Next, a use-case test is performed where Shadow Sort is compared to the best-performing competitor non-dominated sorting algorithm by implementing both into state-of-the-art multi-objective evolutionary algorithms. Several cases of Shadow Sort are proposed based on the number of objectives in the population. It was found that Shadow Sort is competitive when optimising two and three objectives, for various population sizes.

Opsomming

In hierdie studie word 'n nuwe, hibriede, nie-gedomineerde sorteer-algoritme, genoem Skadusortering (“Shadow Sort”), voorgestel. Die algoritme is ontwikkel met inagneming van die werklike, praktiese vereistes van nie-gedomineerde sorteer-algoritmes. Die meeste nie-gedomineerde sorteer-algoritmes, waarvan baie populasie-gebaseerde meta-heuristiese tegnieke gebruik, word saam met 'n veeldoelige optimeringsalgoritme aangewend. Die uitsette van elke populasiegebaseerde meta-heuristiese iterasie bevat gewoonlik honderde, indien nie duisende nie, oplossings wat gesorteer moet word om die volgende evolusionêre iterasie te genereer. Dit is egter net 'n relatief klein aantal oplossings wat deur 'n bepaalde meta-heuristiese iterasie voorgestel word, wat bruikbaar is. Daarom hanteer Skadusortering alle nie-gedomineerde sorterings deur gedomineerde oplossings so vroeg moontlik uit te skakel, eerder as om alle oplossings te verwerk ten einde hul onderskeie Pareto-subversamelings te identifiseer. Skadusortering is in Matlab ontwikkel, en word eers vergelyk met ander nie-gedomineerde sorteer-algoritmes, naamlik Deduktiewe Sortering (“Deductive Sort”), Effektiewe Nie-gedomineerde Sortering (“Efficient Non-dominated Sort”) en Besgeordende Sortering (“Best Order Sort”). Daarna word 'n toepassingsgevalle-toets uitgevoer waarin Skadusortering vergelyk word met die nie-gedomineerde mededingingsorteer-algoritme wat die beste presteer deur albei algoritmes in die jongste veeldoelige evolusionêre algoritmes te implementeer. Gebaseer op die aantal doelstellings in die populasie, word verskeie gevalle van Skadusortering voorgestel. Daar is bevind dat Skadusortering vir verskillende populasiegroottes mededingend optree wanneer twee en drie doelwitte geoptimeer word.

Acknowledgements

I would like to thank the following people for their support in compiling this thesis:

- My wife, Veronique Tränkle, for unwavering love, support, motivation and proofreading.
- My supervisor, Professor James Bekker, for his guidance and expertise.
- My parents and parents-in-law for their support and proofreading efforts.

Contents

Declaration	ii
Abstract	iii
Opsomming	iv
Acknowledgements	v
List of Figures	ix
List of Tables	xii
Nomenclature	xvi
1 Introduction	1
1.1 Background	1
1.2 Problem description	2
1.3 Thesis scope	2
1.4 Research objectives	3
1.5 Problem-solving methodology	4
1.6 Thesis organisation	5
1.7 Summary: Chapter 1	6
2 Literature Review	7
2.1 Introduction to multi-objective optimisation	7
2.1.1 Basic definitions	8
2.1.2 Fitness landscape	8

CONTENTS

2.1.3	Pareto geometries	11
2.2	Computational complexity	12
2.3	Statistical testing	13
2.4	Optimisation algorithms	16
2.5	Multi-objective evolutionary algorithms	16
2.5.1	Multi-objective Particle Swarm Optimisation (MOPSO)	17
2.5.2	NSGA II	19
2.5.3	NSGA III	22
2.5.4	AGE-MOEA	26
2.5.5	A-NSGA III	26
2.6	Non-dominated sorting	29
2.6.1	Naïve and slow sorting method	30
2.6.2	Fast non-dominated sort	31
2.6.3	Deductive Sort	31
2.6.4	Efficient Non-dominated Sort (ENS)	35
2.6.5	Best Order Sort	42
2.6.6	Other non-dominated sorting algorithms	45
2.6.6.1	Corner Sort	45
2.6.6.2	Merge Non-dominated Sort	46
2.7	Standardised benchmark test suites	47
2.7.1	The DTLZ Benchmark MOPs	47
2.7.2	The Walking Fish Group test suite	54
2.7.3	The ZDT Benchmark MOPs	60
2.8	Summary: Chapter 2	63
3	Shadow Sort: a novel non-dominated sorting algorithm	65
3.1	Proposed Method	65
3.2	Shadow Sort	66
3.2.1	Case I: Two objectives	66
3.2.2	Case II: Three to 14 objectives	69
3.2.3	Case III: 15 or more objectives	73
3.2.4	Analysis of Computational Complexity	77

CONTENTS

3.3	Summary: Chapter 3	78
4	Evaluation against existing non-dominated sorting algorithms	79
4.1	Test Procedure	79
4.2	Test 1: Entire population ranking	80
4.3	Test 2: Partial population ranking	87
4.4	Summary: Chapter 4	96
5	Shadow Sort tested on multi-objective evolutionary algorithms	98
5.1	Test procedure	99
5.2	Testing with the MOPSO algorithm	101
5.3	Testing with NSGA II	104
5.4	Testing with NSGA III	107
5.5	Testing with the AGE-MOEA	109
5.6	Testing with the A-NSGA III	112
5.7	Discussion of results	114
5.8	Summary: Chapter 5	116
6	Research summary and conclusions	117
6.1	Project summary and conclusions	117
6.2	Future research	119
6.3	Appraisal of research	120
6.4	Concluding remarks	121
	Bibliography	122
	References	126
A	Algorithm hypervolume-per-volume results: ENS vs Shadow Sort	127
B	Non-dominated Sorting Algorithm MATLAB[®] Code	133
B.1	Code for Deductive Sort	134
B.2	Code for the Efficient Non-dominated Sort	136
B.3	Code for the Best Order Sort	137
B.4	Code for the Shadow Sort	139

List of Figures

1.1	The engineering design process	5
2.1	Mapping from the search space to the fitness space	9
2.2	Examples of deceptive problems	10
2.3	Examples of Pareto optimal geometries	11
2.4	Global optimisation approaches	17
2.5	Main NSGA II procedure	21
2.6	Calculation of the crowding distance operator	23
2.7	A hyperplane and supporting reference points	24
2.8	NSGA III reference points and vectors	25
2.9	A-NSGA III structured reference points	28
2.10	A-NSGA III reference points and associated solutions	28
2.11	Illustration of Deductive Sort	32
2.12	ENS: Common approach to non-dominated sorting	35
2.13	ENS: Dominance comparison strategy of ENS	36
2.14	The relationships between p_n and the solutions having been assigned to a set	37
2.15	Categorising dominance comparisons	37
2.16	ENS illustration population	39
2.17	BOS algorithm concept	42
2.18	Corner Sort – using corner solutions to ignore dominated solutions	46
2.19	Pareto front of DTLZ1	50
2.20	Pareto front of DTLZ2	50
2.21	Pareto front of DTLZ3	51
2.22	Pareto front of DTLZ4	51

LIST OF FIGURES

2.23	Pareto front of DTLZ5	52
2.24	Pareto front of DTLZ6	52
2.25	Pareto front of DTLZ7	53
2.26	Pareto fronts of the nine WFG problems	59
2.27	Pareto fronts of the six ZDT problems	62
3.1	Shadow Sort Case I example with one front	67
3.2	Shadow Sort Case I example with 2 fronts	69
3.3	Illustrative example of the Case II <i>Bitmat</i> operation	73
3.4	An example of matrices <i>Key</i> and <i>Standing</i>	75
4.1	Entire population sorting of a randomly-generated dataset with two dimensions	81
4.2	Entire population sorting of a randomly-generated dataset with three dimensions	82
4.3	Entire population sorting of a randomly-generated dataset with four dimensions	82
4.4	Entire population sorting of a randomly-generated dataset with five dimensions	83
4.5	Entire population sorting of a randomly-generated dataset with 10 dimensions	83
4.6	Entire population sorting of a randomly-generated dataset with 15 dimensions	84
4.7	Entire population sorting of a randomly-generated dataset with 20 dimensions	84
4.8	Partial population sorting of a randomly-generated dataset with two dimensions	89
4.9	Partial population sorting of a randomly-generated dataset with three dimensions	89
4.10	Partial population sorting of a randomly-generated dataset with four dimensions	90
4.11	Partial population sorting of a randomly-generated dataset with five dimensions	90

LIST OF FIGURES

4.12	Partial population sorting of a randomly-generated dataset with 10 dimensions	91
4.13	Partial population sorting of a randomly-generated dataset with 15 dimensions	91
4.14	Partial population sorting of a randomly-generated dataset with 20 dimensions	92
4.15	The number of Rank 1 solutions found in a randomly-generated cloud population with 10 000 solutions	96
5.1	Runtimes of the MOPSO algorithm using SS and ENS as non-dominated sorting algorithms	102
5.2	Runtimes of NSGA II using SS and ENS as non-dominated sorting algorithms	105
5.3	Runtimes of NSGA III using SS and ENS as non-dominated sorting algorithms	107
5.4	Runtimes of AGE-MOEA using SS and ENS as non-dominated sorting algorithms	110
5.5	Runtimes of A-NSGA III using SS and ENS as non-dominated sorting algorithms	112

List of Tables

2.1	A summary of the most common complexity classifications	13
2.2	Comparisons performed by Deductive Sort	40
2.3	Comparisons performed by ENS	40
2.4	An example of dominance sets used in MNDS	47
2.5	Standard DTLZ test problems	48
2.6	Standard DTLZ test problems continued	49
2.7	Characteristics of the DTLZ test suite	53
2.8	WFG shape functions	56
2.9	WFG transformation functions	57
2.10	The WFG test suite	58
2.11	Characteristics of the WFG test suite	60
2.12	Standard ZDT test problems	61
2.13	Characteristics of the ZDT test suite	63
4.1	Results of the entire population ranking test	85
4.1	Results of the entire population ranking test (continued)	86
4.2	Results of the partial population ranking test	93
4.2	Results of the partial population ranking test (continued)	94
4.3	The number of Rank 1 solutions found in a randomly-generated cloud population with 10 000 solutions	95
5.1	Runtime results of the MOPSO algorithm using SS and ENS as non-dominated sorting algorithms	103
5.2	Runtime results of NSGA II using SS and ENS as non-dominated sorting algorithms	106

LIST OF TABLES

5.3	Runtime results of NSGA III using SS and ENS as non-dominated sorting algorithms	108
5.4	Runtime results of AGE-MOEA using SS and ENS as non-dominated sorting algorithms	111
5.5	Runtime results of A-NSGA III using SS and ENS as non-dominated sorting algorithms	113
A.1	Hypervolume results of MOPSO using ENS and SS as non-dominated sorting algorithms	128
A.2	Hypervolume results of NSGA II using ENS and SS as non-dominated sorting algorithms	129
A.3	Hypervolume results of NSGA III using ENS and SS as non-dominated sorting algorithms	130
A.4	Hypervolume results of AGE-MOEA using ENS and SS as non-dominated sorting algorithms	131
A.5	Hypervolume results of A-NSGA III using ENS and SS as non-dominated sorting algorithms	132

Nomenclature

Acronyms

AGE-MOEA	Adaptive Geometry Based Multi-objective Evolutionary Algorithm
A-NSGA III	Adaptive-reference-point-based Non-dominated Sorting Genetic Algorithm III
BOS	Best Order Sort
CS	Corner Sort
DS	Deductive Sort
DTLZ	Deb-Thiele-Laumanns-Zitzler Test Suite
EA	Evolutionary Algorithms
ENS	Efficient Non-dominated Sort
FNDS	Fast Non-dominated Sort
HV	Hypervolume
MNDS	Merge Non-dominated Sorting Algorithm
MOO	Multi-objective Optimisation
MOP	Multi-objective Optimisation Problem
MOPSO	Multi-objective Particle Swarm Optimisation

NOMENCLATURE

NSGA II	Non-dominated Sorting Genetic Algorithm II
NSGA III	Non-dominated Sorting Genetic Algorithm III
PAES	Pareto Archived Evolution Strategy
SPEA	Strength Pareto Evolutionary Algorithm
SS	Shadow Sort
WFG	Walking Fish Group Test Suite
ZDT	Zitzler-Deb-Thiele Test Suite

NOMENCLATURE

Greek Symbols

α	Probability of making a Type I error
μ	Mean of a distribution
σ	Standard deviation of a distribution

Roman Symbols

D	Number of decision variables
P_t	Parent population of generation t
Q_t	Offspring population of generation t
R_t	Combined parent and offspring population of generation t
F_i	i^{th} Pareto set
f_i	Objective function i
M	Number of objectives
N	Number of solutions
H_0	Null hypothesis
H_1	Alternative hypothesis
$\mathcal{O}()$	Computational complexity

Chapter 1

Introduction

This chapter introduces the research presented in this thesis. The context of non-dominated sorting as a subfunction of multi-objective evolutionary algorithms is introduced through an analogy about travel, followed by a problem description and formalised research statement. Next, the scope of the research is defined, followed by the research objectives, the problem-solving methodology employed to achieve them, and lastly the structure of this document.

1.1 Background

Imagine someone is embarking on a journey from Cape Town, South Africa to Maputo, Mozambique. There are numerous travel options that could be considered to achieve this goal. One may, for example, decide to board a cruise ship and journey over the sea, or drive a vehicle along the highway. If the goal was to reach the destination in the shortest period of time, one would probably consider boarding an aeroplane to fly directly between the two cities. This mode can further be explored by different airlines and routes, or even a private chartered flight. However, no matter which mode of transport is chosen, a common element which will lengthen the journey will be passport control. Therefore, although significant effort should be focused on optimising the mode of transport, the required burden of standing in a line at the whims of a government official are inevitable. If the process of passport control itself could be optimised, then all modes would benefit. In the field of multi-objective optimisation, evolutionary algorithms employ a

1.2 Problem description

strategy which is to produce a population of potential solutions and then perform some operation to create a subsequent generation which, hopefully, yields better solutions than the previous generation. In order to assess whether the new generation is indeed better than their predecessors, the task of non-dominated sorting is required. Much like the analogy of passport control, this is a necessary step in many population-based evolutionary algorithms. Therefore, while research and development of multi-objective evolutionary algorithms is necessary to improve the performance of optimisation in general, the enhancement of the method used to perform non-dominated sorting would allow for a reduced total runtime for any optimisation algorithm that uses it.

1.2 Problem description

In this thesis, the necessary task of non-dominated sorting as required in evolutionary algorithms is explored. More specifically, the research question asks if this computationally expensive exercise could be redesigned to improve the overall runtime of multi-objective evolutionary algorithms. The research assignment can thus be summarised as

Improve the overall runtime performance of population-based multi-objective algorithms by designing and implementing a faster non-dominated sorting technique.

1.3 Thesis scope

The scope of this thesis will be limited to focus on a subfunction of multi-objective evolutionary algorithms, namely non-dominated sorting. This important phase of evolutionary algorithms is known to be a computationally expensive step, therefore necessarily contributing quite substantially to the overall runtime of the greater evolutionary algorithm. In order for any evolutionary algorithm to proceed with whichever strategy it employs, every generation needs to have its population assessed to identify which solutions should be used for the evolution of the following generation. Therefore, a non-dominated sorting algorithm needs to be called each time a new generation is created, which is generally hundreds, if

1.4 Research objectives

not thousands, of times during any given optimisation run. A selection of evolutionary multi-objective optimisation (MOO) algorithms will be used for testing. In this study, the term “multi-objective” refers to optimisation problems with two or more objectives of which at least two are conflicting.

This research will be conducted in the MATLAB environment. This includes the development of a new non-dominated sorting algorithm and, in order to ensure consistent reporting, the testing and evaluation of the new algorithm against existing non-dominated sorting algorithms. Therefore, wherever possible, existing algorithm MATLAB code will be used. In the event that no MATLAB code can be found for existing non-dominated sorting algorithms, a MATLAB function will be created by using the original author’s algorithm pseudocode. The MATLAB function will then be tested and a comparison made against the original author’s reported algorithm performance to ensure quality coding. If the MATLAB function’s performance cannot be brought to an acceptable performance level, then that algorithm will be excluded from the study.

1.4 Research objectives

The primary objective of this research is to develop a more efficient non-dominated sorting algorithm to extract superior solutions for use in multi-objective evolutionary algorithms. In order to achieve this, the following objectives are set:

1. *Conduct* a literature review on multi-objective optimisation, evolutionary algorithms, and existing non-dominated sorting algorithms and the premise of their search strategies.
2. *Develop* a novel non-dominated sorting algorithm that is able to efficiently rank solutions provided by a multi-objective evolutionary algorithm.
3. *Investigate* accepted test problems and procedures to report on the performance of the new algorithm.
4. *Test* the developed algorithm against recently developed non-dominated sorting algorithms and analyse the results.

1.5 Problem-solving methodology

5. *Improve* the performance of existing multi-objective evolutionary algorithms by implementing a more efficient non-dominated sorting algorithm.

1.5 Problem-solving methodology

The engineering design process presented in Figure 1.1 will be employed as a framework to direct efforts to achieve the research objectives. The first step in the engineering design process is to ask questions, thereby identifying the needs and constraints of what is to be designed. In view of multi-objective optimisation algorithms, the question to ask is which elements of the procedure require significant processing time. In order to answer this question, Step 2 is performed, which involves researching multi-objective algorithms- how they work and which strategies are followed to achieve the desired outcome. Once the problem is broken up into small enough elements by the iterative process of asking and researching, one can begin to imagine possible solutions to the problem, thereby completing Step 3. Step 4 involves investigating and selecting a possible solution proposed during the imagine phase before continuing to Step 5, which is to create a prototype of the solution. Having designed and built a prototype, Step 6 calls for it to be tested and evaluated which allows problem components to be identified in order to continue with the final step of the cycle, which is *improvement*.

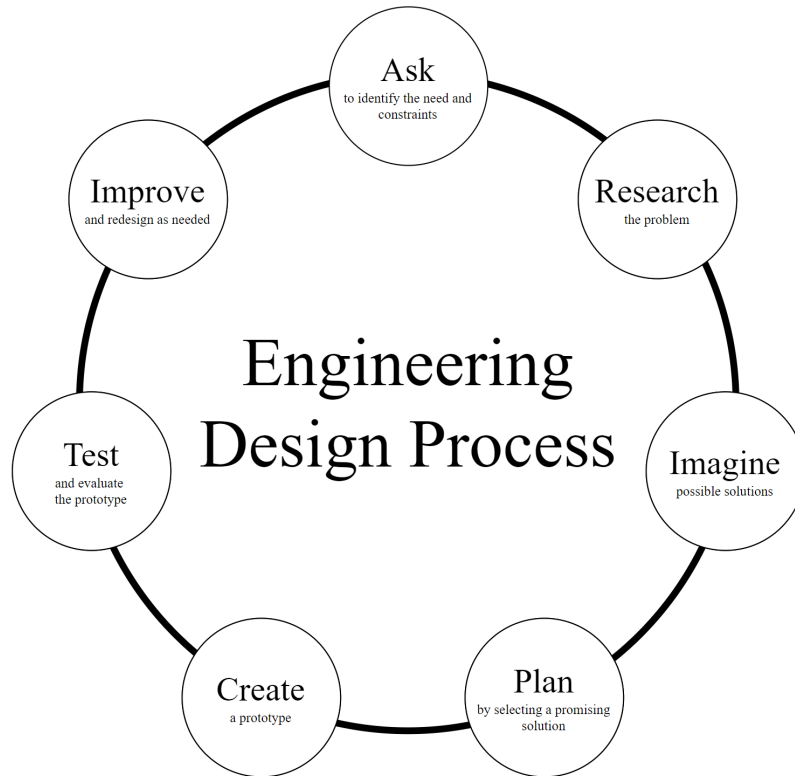


Figure 1.1: The engineering design process ([University of Colorado Boulder, 2016](#))

1.6 Thesis organisation

Following this introductory chapter, this thesis contains another five chapters, a conclusion, a bibliography and appendices. Chapter 2 provides a literature review on evolutionary algorithms, with a specific focus on the theory of multi-objective evolutionary algorithms, as well as some popular examples that will be used in subsequent chapters for testing. Following this is a study on non-dominated sorting, its history, and a review of a few of the state-of-the-art non-dominated sorting algorithms today. To meet the requirement of testing multi-objective evolutionary algorithms and non-dominated sorting algorithms, an overview of standardised test problems is provided. Chapter 3 introduces Shadow Sort (SS) as a novel non-dominated sorting algorithm. The premise of its sorting strategy is provided, along with pseudocode and an analysis of its computational complexity. In Chap-

1.7 Summary: Chapter 1

ter 4, a battery of tests are performed to compare Shadow Sort against some of the most recent non-dominated sorting algorithms found in literature. The results of these tests are tabulated and reported in figures. A comparison between an actual implementation of Shadow Sort and one of the best non-dominated sorting algorithms (Efficient Non-dominated Sort, or ENS) within selected well-known multi-objective evolutionary algorithms and on known test problems is presented in Chapter 5, followed by a discussion about which problems and multi-objective evolutionary algorithms should make use of SS for non-dominated sorting. The conclusion is provided in Chapter 6, followed finally by references and applicable appendices.

1.7 Summary: Chapter 1

This chapter presented the outline to the research problem, and the objectives set in order to achieve the overall goal of the project: to develop a more efficient non-dominated sorting algorithm to extract superior solutions for use in multi-objective population-based algorithms. Achieving this goal could add value to research communities, specifically those involved in the field of multi-objective optimisation.

Chapter 2

Literature Review

In the previous chapter, the concept of non-dominated sorting was introduced. This chapter elaborates not only on the field of non-dominated sorting, but also on where and why it is needed in the greater subject of multi-objective optimisation. This is accomplished by introducing some of the basic definitions and facets of multi-objective optimisation, as well as industry-accepted methodologies to test and quantify properties related to it. Next, numerous multi-objective evolutionary algorithms are introduced and explained, followed by a focus on non-dominated sorting algorithms – a subfunction required by the majority of multi-objective evolutionary algorithms and the focus of this research. Lastly, three standardised benchmark test suites are introduced and discussed in order to define a rigorous means to evaluate the aforementioned multi-objective evolutionary algorithms and the non-dominated sorting algorithms that they use.

2.1 Introduction to multi-objective optimisation

Before proceeding with the literature review, it is important to understand some of the basic terms, definitions and characteristics of multi-objective optimisation. The following sections provide an overview in order to assist understanding when the terms are used in later chapters.

2.1 Introduction to multi-objective optimisation

2.1.1 Basic definitions

The aim of multi-objective optimisation is to locate the set of optimal trade-off solutions referred to as the Pareto optimal set (E in Figure 2.1). In situations where there are problems of large cardinality (high number of elements within the set), a subset of solutions representing all the possible outcomes is usually accepted.

A solution (also known as a parameter vector) \mathbf{a} is said to dominate another solution \mathbf{b} if \mathbf{a} is better than \mathbf{b} on at least one objective, and is not worse than \mathbf{b} on any of its other objectives. This is written as $\mathbf{a} \prec \mathbf{b}$. Furthermore, \mathbf{a} is *equivalent to* \mathbf{b} if and only if \mathbf{a} and \mathbf{b} are identical in all m objectives. If \mathbf{a} is either equivalent to or dominates \mathbf{b} , then \mathbf{a} *covers* \mathbf{b} . If \mathbf{a} and \mathbf{b} are not equivalent and neither dominates the other, then \mathbf{a} and \mathbf{b} are said to be *incomparable* (Huband *et al.*, 2006). The properties of dominance are transitive, therefore if $\mathbf{a} \prec \mathbf{b}$ and $\mathbf{b} \prec \mathbf{c}$, then $\mathbf{a} \prec \mathbf{c}$. A parameter vector \mathbf{a} is considered to be *non-dominated* with respect to the set of vectors X if and only if there exists no vectors in X which dominate \mathbf{a} . As an extension, X is a *non-dominated set* if and only if each vector in X is mutually non-dominating. Finally, a *non-dominated set* (also known as a *Pareto set*) is the set of objective vectors of a non-dominated set. Non-dominated sorting entails assigning solutions in a population to different *ranks*. Keeping in mind the concept of dominance, solutions of the same rank are non-dominated by each other, and are dominated by at least one solution in the preceding rank (Wang & Yao, 2014). The subset of solutions which are non-dominated by any other subset are called Rank 1 solutions. Removing Rank 1 solutions from a population and then performing non-dominated sorting again will yield Rank 2 solutions.

2.1.2 Fitness landscape

Consider a multi-objective optimisation problem which is defined in terms of a search space consisting of n parameters x_1, \dots, x_n , with a vector of M objective functions $f_1(x_1, \dots, x_n), \dots, f_M(x_1, \dots, x_n)$ that map the parameter vectors into the fitness space. The fitness landscape is defined as the mapping from the search space to the fitness space and is illustrated in Figure 2.1.

2.1 Introduction to multi-objective optimisation

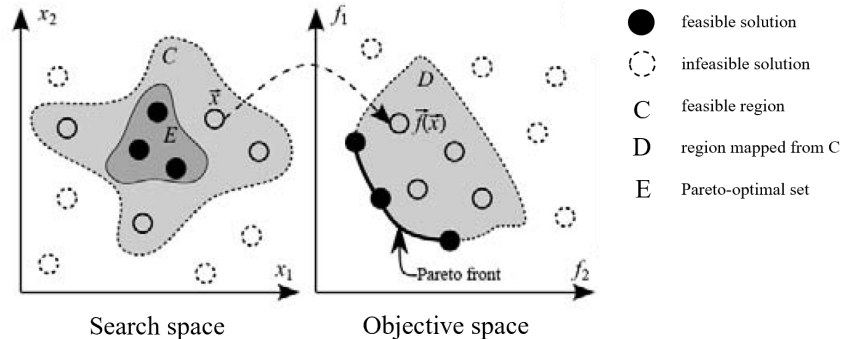


Figure 2.1: An illustration depicting the mapping from the search space to the fitness space (Sheng *et al.*, 2014)

Both the fitness landscape and the relationship between the Pareto optimal set and the Pareto optimal front may be described by certain characteristics.

The fitness landscape may be described as being either one-to-one or many-to-one. Many-to-one cases are generally more difficult for multi-objective optimisation algorithms as decisions have to be made between multiple parameter vectors that evaluate to the same objective vectors. The term *flat region* is used to describe problems where a connected subset of parameter space maps to a singleton in the fitness space. This means that small adjustments to the parameters do not change the objective values. More often than not, this is problematic for optimisers as the “feedback” between the search space and the fitness space is lost. In situations where the majority of the fitness space is flat, the Pareto optima are referred to as *isolated optima*.

Modality refers to the number of local optima a fitness landscape may have. Unimodal objective functions contain only one optimum, whilst a multi-modal objective function will have multiple local optima. An objective function may also be referred to as being *deceptive*, meaning that it contains at least two optima—a true optimum and a deceptive optimum. In a deceptive objective function, the majority of the search space will favour the deceptive optimum. Problems containing deceptive objectives are known as *deceptive problems*. These problems exacerbate the difficulty of multi-modal problems because the true optimum is usually found in an unlikely place. Huband *et al.* (2006) provides two exam-

2.1 Introduction to multi-objective optimisation

ples illustrating the difference between deceptive and non-deceptive multi-modal objectives, shown in Figure 2.2.

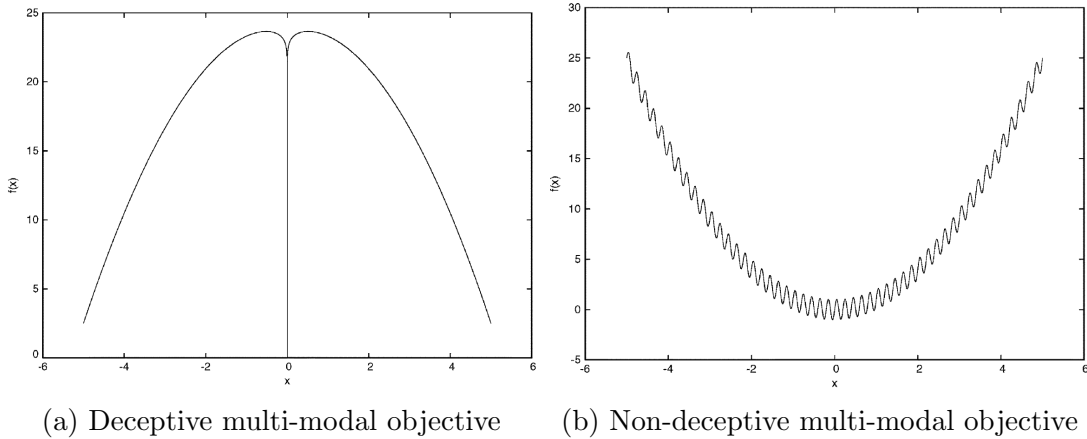


Figure 2.2: Examples illustrating the concept of deceptive problems (Huband *et al.*, 2006)

Another important aspect of a problem is *parameter dependencies*. Consider a single objective m , a parameter vector s , and an index i . The problem of optimising M by varying only s_i would therefore be defined as $P_{m,s,i}$. This is a single objective problem with a single parameter. Furthermore, $P_{m,s,i}^*$ may be defined as the set of global optima for every subproblem. If $P_{m,s,i}^*$ is identical for all values of s , then s_i is *separable* on m . Otherwise, s_i is *non-separable* on m . If every parameter of m is separable, then m is said to be a *separable objective*. Likewise, if every objective of a problem P is separable, then P is said to be a *separable problem*. Separable problems are typically simpler to optimise, as each separable objective can be optimised by independently considering each parameter in turn. Once this is achieved, the resulting set of globally optimal parameters may be determined by taking the cross-product of the optimal sets for each individual parameter.

Bias is a term used to describe the distribution of parameter vectors in the search space, and it has an impact on the search process when mapping from the Pareto optimal set to the Pareto optimal front. The degree to which a problem is considered as biased depends on the density variation of solutions in the

2.1 Introduction to multi-objective optimisation

fitness space, given an even distribution of solutions in the parameter space. Currently, no mathematical definition of bias exists, and so it is a more qualitative characteristic that is best investigated by plotting solutions in the fitness space.

2.1.3 Pareto geometries

The Pareto optimal set of a multi-objective problem differs fundamentally from that of a single objective problem in that it may possess a variety of geometries versus a single point of a single objective problem. These geometries may be described as being *convex* or *concave* (or *non-convex*). If a geometry is referred to as being *strictly* convex or concave, it means that it does not contain any segments which could be described by the opposing geometry (concave or convex). If the geometry has both convex and concave segments, it is a *linear* set. A set which is made up of connected subsets that are each strictly convex, strictly concave or linear is a *mixed* front. A *degenerate* set has a lower dimension than the objective space in which it is rooted, less one. For example, if a three-objective problem has a two-dimensional Pareto front, it is not considered degenerate. However, if the Pareto set of the three-objective problem were a line segment, then it would be degenerate. A *disconnected* set is self-explanatory — the sets making up the front are not connected.

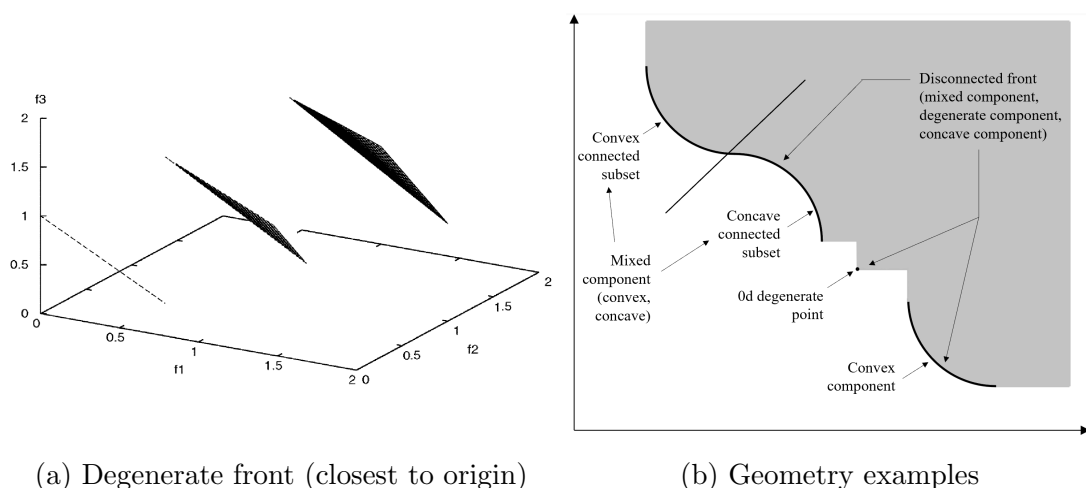


Figure 2.3: Examples of Pareto optimal geometries (Huband *et al.*, 2006)

2.2 Computational complexity

When describing computer functions or algorithms specifically for the sake of comparison against others, the first question should relate to the function's ability to perform the task it was meant to perform. For instance, if a function was designed to add the numbers from one to a specified input N , two functions that would both be able to perform the task required would be:

$$y = 1 + 2 + 3 + \dots + (N - 1) + N \quad (2.1)$$

or

$$y = \frac{N^2}{2} + \frac{N}{2}. \quad (2.2)$$

Satisfied that two different functions will offer the same result, the next feature used to distinguish the performance between two functions or algorithms is how fast they are, or in other words, what their runtime is. This may be a very complicated question to answer with a definitive value as there are many variables to consider. Some of these variables may relate to the processing performance of the computer on which the function or algorithm is being run, while others may relate to the specific programming language in which they are coded. Therefore, it is much more useful to describe the manner in which the runtime of a function or algorithm scales against the input given to it. There are various options one could use to describe such a property, such as having constant time (*i.e.* a function will require x amount of time, no matter the input provided), or having linear time (*i.e.* the increase in processing time is linearly proportional to the size of the input provided). Another option would be to say that the function or algorithm has quadratic time, or even exponential time. Suffice to say, many different potential descriptions exist to describe such a property.

To formalise a mathematical manner in which to describe computational complexity, the notion of “*Big-O*” is used and is represented as $\mathcal{O}()$. The size of the input argument is represented by N , and the operation performed on this input is written mathematically. Table 2.1 summarises the most common complexities.

2.3 Statistical testing

Table 2.1: A summary of the most common complexity classifications

Complexity	Description
$\mathcal{O}(1)$	Constant time
$\mathcal{O}(\log N)$	Logarithmic time
$\mathcal{O}(N)$	Linear time
$\mathcal{O}(N \log N)$	Linearithmic time
$\mathcal{O}(N^k)$	Polynomial time
$\mathcal{O}(k^N)$	Exponential time
$\mathcal{O}(N!)$	Factorial time

Based on the computational complexity reported, it is easy to understand the effect on runtime an increase in the input will have. For example, if one was to say that an algorithm had complexity $\mathcal{O}(N^2)$, then the runtime increases by the square of the input — an input b that is ten times larger than an input a will take 100 times longer to run.

In order to report the computational complexity of a calculated formula describing how a function scales, the following steps are used:

1. Find the fastest growing term.
2. Remove the coefficient from the term found in Step 1.

For example, if the actual runtime of an algorithm was calculated to be

$$T(N) = 3N^3 + 2N + 7 \tag{2.3}$$

then the fastest growing term will be $3N^3$, which has a coefficient value of 3. Therefore, the computational complexity would be reported as $\mathcal{O}(N^3)$.

2.3 Statistical testing

Multi-objective evolutionary algorithms fall under the category of stochastic optimisation techniques. As such, it becomes important to ensure that multiple tests are conducted when comparing outcomes, because the results of a stochastic process will be a specific instance of a set of possible outcomes. Outcomes (which

2.3 Statistical testing

may refer to an output answer from a multi-objective evolutionary algorithm or the algorithm's runtime) of stochastic events are best described with a mean and standard deviation.

In Chapters 4 and 5, repeated tests between algorithms are conducted to ensure fairness and accurate reporting. In order to claim that a statistically significant difference exists between the algorithm candidates, it is necessary to perform a statistical test that is suitable.

Many statistical tests make use of hypothesis testing to make a decision regarding a given test output. A hypothesis test allows for one of two decisions to be made per test. Firstly, there is the null hypothesis which is indicated by H_0 . The null hypothesis specifies the conditions that are being tested and is assumed to be true unless there is sufficient evidence generated to reject it. The second decision outcome is described by the alternative hypothesis which is indicated by H_1 . It defines all other conditions which are feasible, but not being tested (Montgomery & Runger, 1994).

The Wilcoxon rank-sum test can be used to test the hypothesis $H_0 : \mu_1 = \mu_2$. It is most suitable when testing two independent and continuous populations with the same shape and spread, but not necessarily with the same locations. This test is suitable for analysing the effect on runtime caused by exchanging a multi-objective algorithm's non-dominated sorting technique with another, as the underlying evolutionary algorithm remains the same. As mentioned earlier, the runtime results of an evolutionary algorithm are considered samples from a true runtime distribution with parameters mean μ and standard distribution σ . The test is performed to assess whether the means of two distributions (differentiated by the non-dominated sorting algorithm they employ) are the same or not, thus the hypotheses are formulated as follows:

$$H_0 : \mu_1 = \mu_2 \quad H_1 : \mu_1 \neq \mu_2 \quad (2.4)$$

If the result of the Wilcoxon rank-sum test allows us to reject H_0 , then the mean results of the algorithms are assessed to determine which non-dominated sorting technique is superior.

2.3 Statistical testing

The Wilcoxon rank-sum test is performed as follows (assuming a level of significance of $\alpha = 0.05$)

1. Sort all observations in ascending order and assign ranks to them. If there is a tie, use the mean of the ranks that would have been assigned to the tied observations.
2. Calculate the sum of the ranks of population one and two (U_1, U_2), and assign U_1 to be the smaller of these values. If the population sizes are the same, then the sums of the ranks will be the same.
3. Calculate the mean of the rankings. When the number of observations from population 1 (n_1) and population 2 (n_2) both exceed eight, the sampling distribution of U_1 or U_2 approaches the normal distribution (Walpole & Myers, 1993). The mean and variance are calculated by

$$\mu_{U_1} = \frac{n_1 n_2}{2} \quad (2.5)$$

$$\sigma_{U_1}^2 = \frac{n_1 n_2 (n_1 + n_2 + 1)}{12} \quad (2.6)$$

4. Calculate the z -statistic with:

$$z = \frac{U_1 - \mu_{U_1}}{\sigma_{U_1}} \quad (2.7)$$

If the z -statistic is smaller than -1.96 or larger than 1.96, then reject H_0 .

In summary, the Wilcoxon rank-sum test is a non-parametric test for the equality of means of two populations. Therefore, it is a suitable choice when comparing test results of a newly developed algorithm to an existing algorithm, as these tests are structured to compare the runtimes of two different implementations of non-dominated sorting within a given multi-objective optimisation algorithm.

2.4 Optimisation algorithms

In general, the primary motivation behind any engineering undertaking is to create something that will be of benefit to a stakeholder (animate or inanimate) within the system of said undertaking. More often than not, the design and creation of the product will involve making decisions to optimise a set of objectives. This task becomes even more difficult when optimising one objective necessarily worsens the result of another. The task of finding the most suitable compromise for conflicting objectives is described by multi-objective optimisation. To formalise concepts used in the field, one may refer to potential solutions to a problem as *decision variables*, and to the quantification of a particular requirement of the problem itself as an *objective*. Therefore, in multi-objective optimisation, one aims to find the best set of decision variable values which result in the most desirable objective outcomes. The word *outcomes* is plural because in multi-objective problems, there is invariably no one optimal solution, but rather a set of solutions. Various optimisation techniques exist, and can be categorised into three categories, namely enumerative, deterministic and stochastic (Coello *et al.*, 2007).

This study will focus on a technique found along the stochastic branch, namely evolutionary computation, which is a technique inspired by the natural process of evolution. Evolutionary computation techniques consist of a population of (initially randomly generated) solutions which are processed in such a manner that they evolve into a better generation to meet the needs of the objective functions. This process is repeated until some stopping criteria is met, resulting in a final population of solutions.

2.5 Multi-objective evolutionary algorithms

This section is devoted to introducing the multi-objective optimisation algorithms that are used later in this study. These algorithms were selected as they represent some of the most popular and well-regarded multi-objective evolutionary algorithms in the field, or are recently developed variations of these well-known algorithms.

2.5 Multi-objective evolutionary algorithms

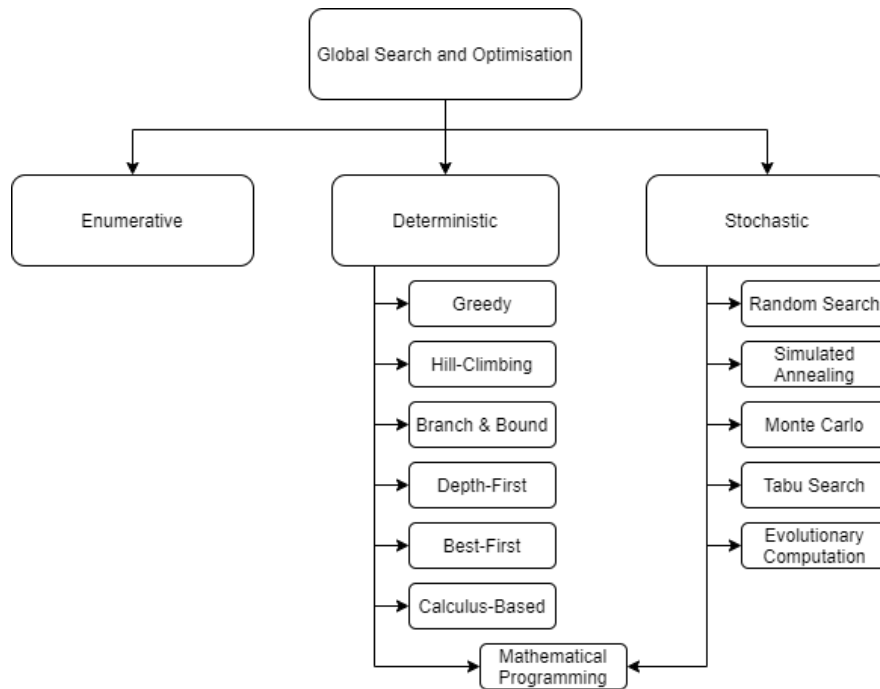


Figure 2.4: Global optimisation approaches (Coello *et al.*, 2007)

2.5.1 Multi-objective Particle Swarm Optimisation (MOPSO)

The movement of flocks of birds and schools of fish have long captivated scientists, who have strived to understand and replicate the movement of such organisms and is the basis of MOPSO. Two notable simulations were presented by Reynolds (1987) and Heppner & Grenander (1990) who attempted to graphically simulate the unpredictable movements of flocks of birds. In order to achieve such a simulation, two properties were established and used to dictate the movement of each agent within the simulation; the matching of velocity to the nearest neighbour, and a randomness value. The velocity-matching property is assigned through each iteration loop, where, for a specific agent in focus, its nearest neighbour is found, and that neighbour's X and Y velocities are assigned to the agent in focus. As can be expected, it does not take long before all agents settle on a unanimous direction without any intervention to change it. This is where the random property is required. At each iteration, stochastic changes are made to selected X and

2.5 Multi-objective evolutionary algorithms

Y velocities, thereby introducing enough change into the population of agents to suitably mimic the natural world. Later, Heppner’s bird simulation incorporated a feature which acted like a dynamic force on the flock. A fixed “roost” position was defined which attracted the agents, thereby eliminating the need for the randomness property. This in turn led to yet another variation of the simulation which introduced what became known as the Cornfield Vector. The idea was that the agents do not necessarily know exactly where the roost point (or any other desired location such as a food source) was. Instead, the agents would use their own memory, along with the best flock memory of good position to direct their next move. In nature, birds are able to capitalise on one another’s knowledge (think about how quickly birds find a bird-feeder even if it has only recently been put there) (Sear *et al.*, 2007). The two-dimensional Cornfield Vector stores specific X_1Y_1 coordinates, and each agent evaluates its present position in terms of the equation

$$Eval = \sqrt{(presentX - X_1)^2} + \sqrt{(presentY - Y_1)^2} \quad (2.8)$$

Each agent remembers which position resulted in their best (lowest) *Eval* value. This position is called $p_best[]$, and is comprised of position coordinates $p_bestX[]$ and $p_bestY[]$ ¹. An agent’s movement is guided by its current XY position relative to two elite positions, namely the agent’s best position $p_best[]$, and the flock’s best global position $g_best[]$. In a two-stage manner, the agent first evaluates its position relative to $p_best[]$, and then tries to move back towards it. For example, if the agent finds itself left of $p_bestX[]$, then its X -velocity (vx) will be adjusted positively by a random weighted amount as

$$vx = vx + rand() * p_increment, \quad (2.9)$$

where $p_increment$ is a defined algorithm parameter weight. If the agent is right of its $p_bestX[]$, then the $rand() * p_increment$ will be subtracted from its vx . The same principle applies to the Y component. The second stage of the agent’s move would then compare this newly calculated velocity against $g_best[]$ in the

¹The square brackets indicate vectors with number of elements equals to the number of agents

2.5 Multi-objective evolutionary algorithms

same manner. Assuming the correct selection of $p_increment$ weights, the result of this addition to the simulation was that the agents circled around the goal position with movements very similar to those witnessed in nature before ultimately coming to rest on the exact Cornfield Vector position. Based on the success of the simulation, it becomes clear that the technique could be modified for simple, two-dimensional optimisation. Some changes to the original simulation were made to enhance the goal-finding requirement, including the removal of nearest-neighbour velocity matching. Although this did indeed improve the performance of the algorithm as an optimisation technique, it also meant that the aesthetically pleasing flock-like behaviour was lost, and the iterative evolution of the algorithm started to resemble something more like a swarm. In 1995, [Kennedy & Eberhart \(1995\)](#) introduced particle swarm optimisation (PSO). The algorithm boasts both computationally inexpensive memory requirements and speed.

Seven years later, [Coello Coello & Lechuga \(2002\)](#) introduced an extension of PSO to deal with multi-objective optimisation problems. In this version, Pareto dominance is employed to determine the direction of a particle while maintaining previously discovered non-dominated vectors in a repository which is accessible to other particles to guide their own direction. The global attraction mechanisms of PSO coupled with the proposed historical archive of previously identified non-dominated vectors facilitate convergence towards globally non-dominated solutions.

In their paper introducing the algorithm, [Coello Coello & Lechuga \(2002\)](#) compared MOPSO against two other state-of-the-art multi-objective evolutionary algorithms, namely the non-dominated sorting genetic algorithm, NSGA II (see Subsection 2.5.2) and the Pareto Archived Evolution Strategy, PAES ([Knowles & Corne, 2000](#)). They concluded that MOPSO performed “relatively well,” further elaborating that it remained a competitive algorithm, although it did not necessarily outperform the other two algorithms in all cases.

2.5.2 NSGA II

In the same year that [Coello Coello & Lechuga \(2002\)](#) introduced MOPSO, [Deb *et al.* \(2002\)](#) introduced the Non-dominated sorting genetic algorithm II

2.5 Multi-objective evolutionary algorithms

(NSGA II), an algorithm which is still viewed today as one of the most popular multi-objective evolutionary algorithms (Garcia & Trinh, 2019). Before the introduction of NSGA II, multi-objective evolutionary algorithms making use of non-dominated sorting were criticised because of the following:

1. their high computational complexity of $\mathcal{O}(MN^3)$ (where N is the size of the population and M is the number of objectives),
2. their lack of elitism in their approach, and
3. their requirement for a sharing parameter to be specified.

The introduction of NSGA II addressed all three of the above issues. During initial algorithm testing on standard benchmark multi-objective problems, NSGA II boasted a fast non-dominated sorting time with $\mathcal{O}(MN^2)$ computational complexity, while being able to find a much better solution spread as well as enhanced convergence near the optimal Pareto set when compared to Pareto-archived evolution strategy, PAES (Knowles & Corne, 2000) and strength-Pareto evolutionary algorithm, SPEA (Zitzler & Thiele, 1999). NSGA II is described next, together with the original figures as presented by Deb *et al.* (2002). Input parameters required from the user are population size N , number of population generations g , and mutation m and cross-over c probabilities.

The algorithm begins by generating a random initial population of size N and then calculates the objective values. This is followed by non-dominated sorting, after which these solutions are used to generate the offspring population. The parent and offspring populations are then combined to form a larger population R_t of size $2N$ which also undergoes non-dominated sorting. The lowest (best) rank solutions are then selected for inclusion into the next generation of solutions P_{t+1} .

In Figure 2.5, the segment labelled P_t represents the parent population in the algorithm repository. Each solution contains its decision variables, as well as its objective values which need to be optimised. Members of the parent population are selected for breeding by means of tournament selection. Q_t represents the

2.5 Multi-objective evolutionary algorithms

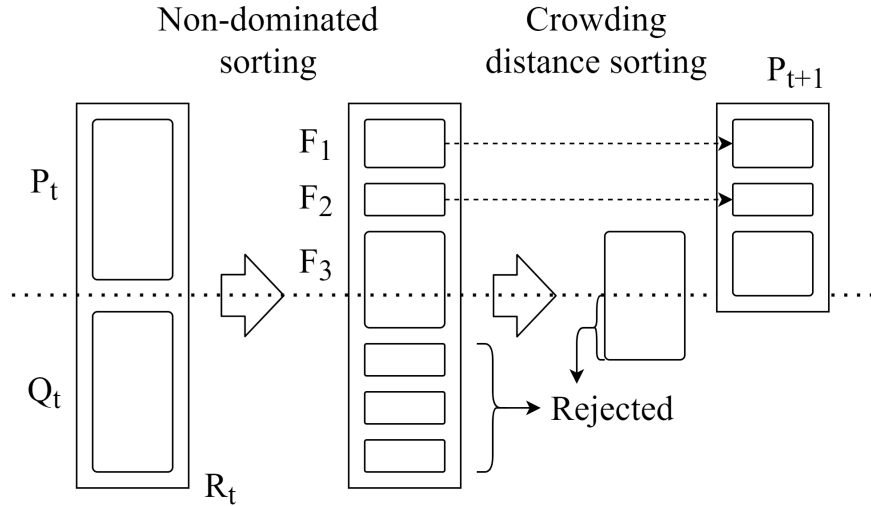


Figure 2.5: Main NSGA II procedure (Deb *et al.*, 2002)

offspring population which was generated by crossing over the genes of the tournament winners from parent population P_t , and then mutating some of the genes with a specified parameter.

The combination of P_t and Q_t results in R_t , a population twice the size of P_t . Non-dominated sorting is then applied to the population R_t . When possible (depending on the non-dominated sorting algorithm employed), only enough solutions to form complete rank sets summing to at least the size of P_t need to be ranked. As indicated in Figure 2.5, the full sets of ranks F_1 , F_2 and F_3 have undergone non-dominated sorting. The new population P_{t+1} is created by selecting the best subsets from the ranked solutions until the size of the original population is reached.

If a subset cannot be selected for the new population P_{t+1} in its entirety, then a crowding distance operator (\prec_n) is calculated and used to determine which members of that subset will be included in the next generation of the population. This avoids convergence on local optima instead of global optima and ensures a good solution spread. Solutions with a larger crowding distance are selected first.

Computing the crowding distance operator is accomplished by sorting the solutions in ascending order for each objective. Then, the boundary solutions of

2.5 Multi-objective evolutionary algorithms

each objective (solutions 1 and l) are assigned a crowding distance value of infinity to ensure that they will always be included in the next population generation. The remaining solutions are assigned a value calculated by normalising each objective m , and then calculating $d_m(i)$, the absolute difference in objective values of the two neighbouring solutions $|f_m(i-1) - f_m(i+1)|$, where $m \in [1 : M]$. This procedure is repeated for all objectives M in the population. The total crowding distance for each individual is the sum of all distances $d_m(i)$ for every objective $m \in [1 : M]$. Figure 2.6 provides an illustration which explains the crowding distance operator.

After having undergone non-dominated sorting and crowding distance assignment, each solution i in the population now has two attributes, namely:

- non-domination rank (i_R)
- crowding distance (i_d)

Between two solutions of different ranks, the solution with the lower (better) rank will be selected. If the solutions share the same rank, then the solution with the larger crowding distance will be selected.

Once the size of population P_{t+1} is the same size as P_t , the algorithm is ready to repeat the procedure for the next generation, starting by creating offspring population Q_{t+1} . The entire procedure is repeated until the specified number of generations g has been created.

2.5.3 NSGA III

The non-dominated sorting genetic algorithm (NSGA) III was introduced by [Deb & Jain \(2014\)](#) 12 years after its popular predecessor NSGA II. It is a reference-point-based evolutionary multi-objective optimisation algorithm which focuses on solutions that are non-dominated, but are also close to a set of given reference points. When considering many-objective problems, one of the main issues facing existing multi-objective optimisation algorithms is the high proportion of non-dominated solutions in a randomly chosen set of objective values. Since the non-dominated solutions account for most of the population, any elite-preserving algorithm experiences difficulty in trying to include a sufficient number of new

2.5 Multi-objective evolutionary algorithms

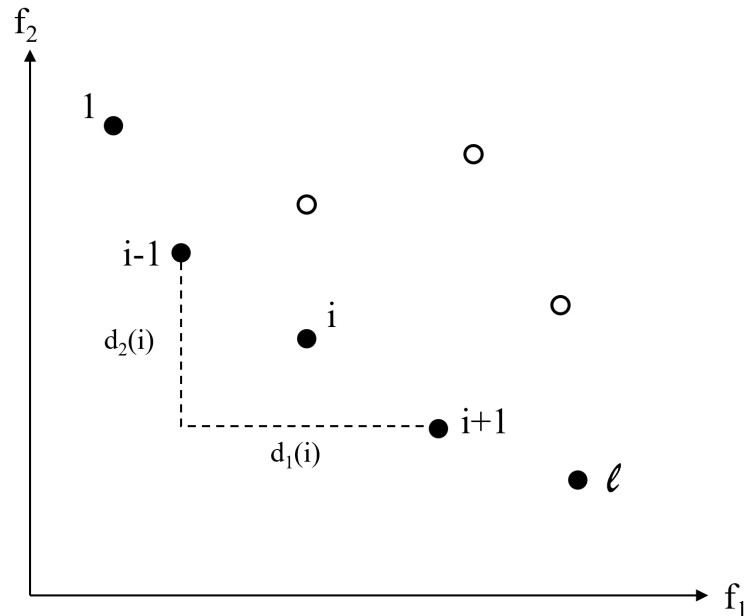


Figure 2.6: An illustration explaining the calculation of the crowding distance operator

solutions in the population. This hinders evolution and delays the search process considerably (Reddy & Kumar, 2006). Another challenge experienced by many current optimisation algorithms is the need to calculate a diversity-preserving operator such as a clustering operator (Zitzler & Thiele, 1999) or a crowding-distance operator (Deb *et al.*, 2002). The calculation of such operators becomes computationally expensive as the number of objectives increases.

As such, the main adaptation to NSGA II that is presented in NSGA III is the replacement of the crowding-distance operator with suitable reference points. The NSGA III allows for these reference points to either be supplied preferentially by the user, or be predefined in a structured manner. Should the latter be chosen and no preferential information provided, then a systematic approach proposed by Dennis (1998) can be used¹. In this method, reference points are placed on a normalised hyper-plane which is equally inclined to each objective axis and intersects each objective at the one-intercept. Since the hyper-plane is normalised,

¹Other predefined structures of reference points may also be used, but this is the method employed by Deb & Jain (2014) in their original paper.

2.5 Multi-objective evolutionary algorithms

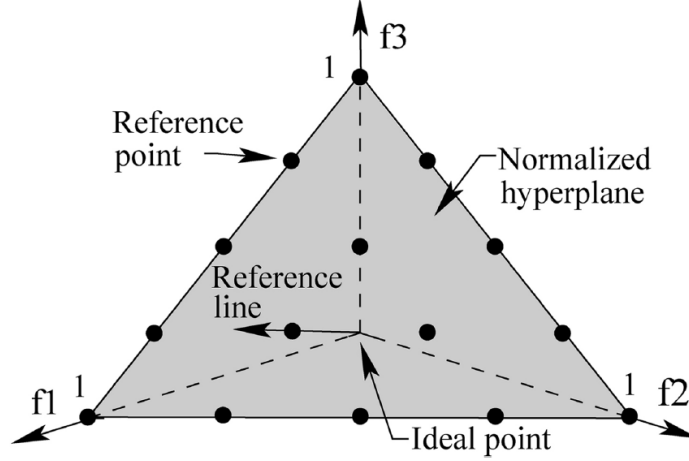


Figure 2.7: A hyperplane intersecting each objective axis at one, and supporting H reference points (Deb & Jain, 2014)

scaled objectives are sufficiently handled without complicating the reference point methodology. Once a hyper-plane is established, it is divided into H reference points as illustrated in Figure 2.7.

The number of reference points is calculated based on the number of objective M , and the number of divisions created along each objective p according to

$$H = \frac{M + p - 1}{p} \quad (2.10)$$

The number of reference points used by NSGA III is not a user-specified parameter, but rather related to the number of trade-off points (solutions) requested by the user. Thus, the number of reference points is roughly equal to the size of the population, $H \approx N$. As in NSGA II, the next generation P_{t+1} is created by using entire sets of non-dominated subsets after non-dominated sorting has taken place (see Figure 2.5). However, if an entire non-dominated subset cannot fit into the next generation (as a result of the requirement to keep the population size N of P_{t+1} the same size as P_t), then the reference points are used to select which solutions from this non-dominated subset are selected for inclusion in the next generation P_{t+1} . This is accomplished by calculating the perpendicular distance between the solutions in R_t and a vector extending from the origin through each

2.5 Multi-objective evolutionary algorithms

of the reference points as depicted in Figure 2.8.

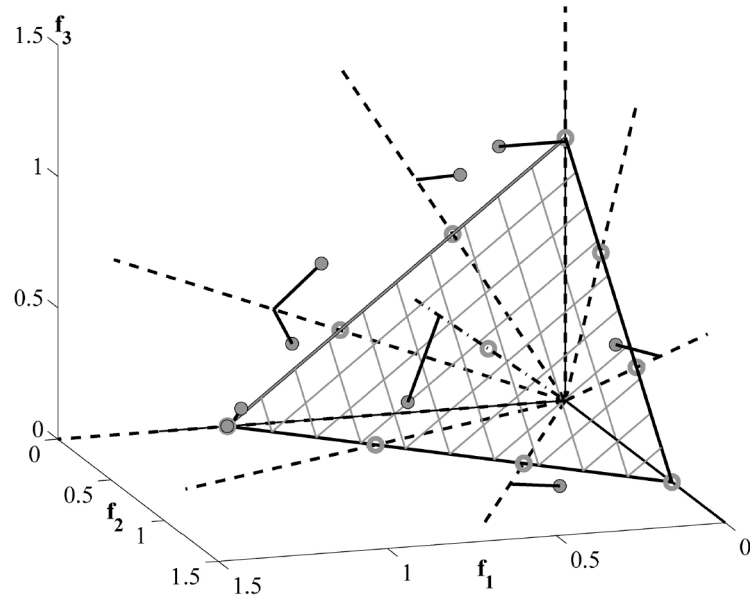


Figure 2.8: Vectors extending from the origin through the H reference points used for determining which solutions in R_t are to be included in the next generation P_{t+1} (Deb & Jain, 2014)

The solutions which are nearest to the provided reference vectors are included in the next generation.

2.5 Multi-objective evolutionary algorithms

2.5.4 AGE-MOEA

NSGA III, and specifically its use of the reference point methodology for solution selection for a subsequent generation, is based on the implicit assumption that the Pareto set has Euclidean geometry (*i.e.* a flat plane as can be seen in Figure 2.7). However, many multi-objective problems have Pareto sets whose geometry is convex (and therefore hyperbolic geometry), or concave (and therefore spherical geometry) (Martínez *et al.*, 2014). In response to this, Panichella (2019) introduced Adaptive Geometry Based Multi-Objective Evolutionary Algorithm (AGE-MOEA), which provides a framework based on NSGA II, but creates no assumptions of the geometry of a multi-objective problem's Pareto set. AGE-MOEA differs from NSGA II in the replacement of the crowding-distance factor with a survival score, which is calculated based on the diversity and proximity of non-dominated sets. To facilitate such a computation, AGE-MOEA estimates the geometry of the first non-dominated subset in each generation and then assumes this to be the geometry of the real Pareto set. As the algorithm matures towards better solutions, this assumption becomes more accurate.

Having estimated the geometry of the Pareto set based on the first non-dominated set, each solution is assessed and receives a *proximity* score and a *diversity* score. The proximity score is computed by calculating the distance between the solution and the ideal point based on the calculated geometry, while the diversity score is simply the minimum distance between the current solution and other solutions in that non-dominated set. These two scores are then used to calculate a *survival* score for each solution.

Then, much like NSGA II and NSGA III, the solutions are selected for the next generation based on their non-dominated rank. If an entire non-dominated set does not entirely fit into the next generation, the solutions are selected based on their survival scores. Based on their paper, Panichella (2019) claims that AGE-MOEA indiscriminately outperforms NSGA III.

2.5.5 A-NSGA III

Adaptive-reference-point-based non-dominated sorting genetic algorithm (A-NSGA III) is another multi-objective evolutionary algorithm which is based on the frame-

2.5 Multi-objective evolutionary algorithms

work of NSGA II. The reason for its development was to address a processing issue in a very specific environment, namely endmember extraction in hyperspectral imaging. Geological mapping, precision agriculture, resource surveying, target detection and even geological analysis of Mars are just a few of the applications of hyperspectral imaging, a technique that is considered invaluable in the field of Earth observation (Tong *et al.*, 2014). Hyperspectral images typically contain two important types of data; spatial information and spectral signature. The latter is made up of hundreds of thin, adjacent spectral ranges per pixel, which allow for easy identification and interpretation of ground objects. However, due to the homogenous mixture of different objects on the ground, as well as limited resolution, many hyperspectral images contain pixels that contain multiple types of materials, negatively affecting ground object recognition and classification accuracy. Therefore, it is necessary to perform what is known as spectral unmixing, a task which aims to decompose the mixed pixels into various heterogeneous ground elements known as endmembers. The extraction of these endmembers significantly enhances the ground object recognition and classification accuracy mentioned above (Xu *et al.*, 2015).

In recent years there have been numerous proposals for endmember extraction. However, most of these methods fail to represent all the properties of each endmember. Research then turned towards multi-objective optimisation techniques with the aim to generate a set of Pareto-optimal solutions. In response, Cheng *et al.* (2019) introduced A-NSGA III which dynamically creates and updates reference points (instead of the crowding-distance factor found in NSGA II). Much of the framework of NSGA II was retained in A-NSGA III, however an important distinction can be found for the selection of solutions from the non-dominated set which does not entirely fit into the following generation's parent population P_{t+1} . A-NSGA III calculates equidistant reference points, whose number and positions depend on the number of objectives in the problem. An example of the reference points used in a two-dimensional problem is depicted in Figure 2.9.

Once offspring solutions have been generated, each solution is associated with its nearest reference point. An example of reference points and their associated solutions is provided in Figure 2.10. It may be that reference points exist which

2.5 Multi-objective evolutionary algorithms

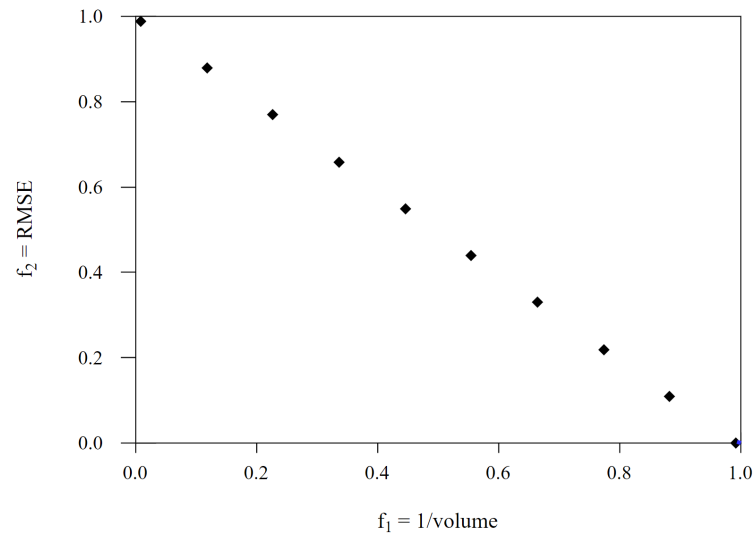


Figure 2.9: Structured (equidistant) reference points for a two-dimensional problem (Cheng *et al.*, 2019)

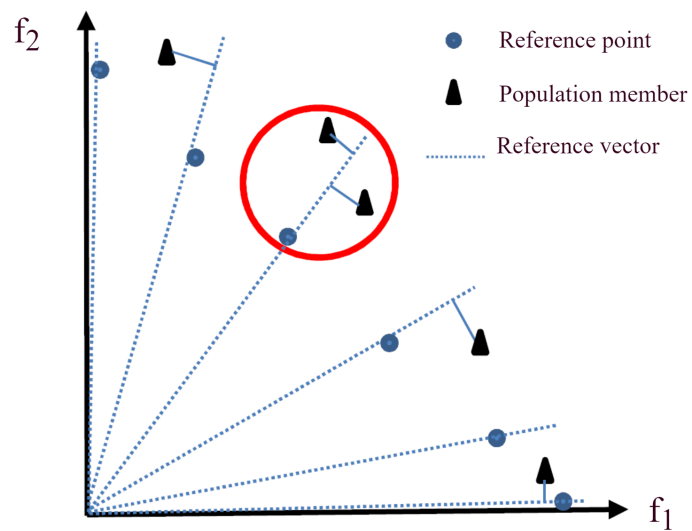


Figure 2.10: Reference points and their associated solutions (Cheng *et al.*, 2019)

2.6 Non-dominated sorting

have multiple solutions associated with it (see circled region in Figure 2.10), whilst others may have none.

Then, as in NSGA II, solutions are selected for inclusion in the next generation based on their non-dominated rank. If a non-dominated set (symbolised by s) does not fit into the next generation of size N in its entirety, then the reference points are used to dictate which members of that non-dominated set should be included.

The reference points are assessed in increasing order of the number of solutions already included in the next generation (*i.e.* the solutions which dominate those in s) associated with them. This number is recorded as p_j . Two scenarios are possible:

1. If $p_j = 0$, then no solution that is currently included in the next generation is associated with this reference point. If there are any solutions in s associated with such a reference point, then that which is closest to the reference point is included in the next generation, and that reference point's citation is incremented to $p_j \leftarrow p_j + 1$. If there are no solutions in s associated with this reference point, then it is removed from the current population.
2. If $p_j \geq 1$, then a random solution in s associated with that reference point is selected for inclusion in the next generation and its citation is also incremented to $p_j \leftarrow p_j + 1$.

In their paper, [Cheng *et al.* \(2019\)](#) claim A-NSGA III to be a robust variation of NSGA II with good search capabilities, although further development of the algorithm is intended.

2.6 Non-dominated sorting

Non-dominated sorting (also known as non-dominated ranking)¹ is a fundamental activity in many multi-objective evolutionary algorithms. In single-objective optimisation, it is very easy to rank the solutions based on their objective values — if it is a minimisation problem, then the solution with the lowest objective

¹Non-dominated sorting and non-dominated ranking are used interchangeably.

2.6 Non-dominated sorting

value is the best. However, for multi-objective optimisation, ranking of solutions becomes more complicated, as a solution which fares well in one objective may perform poorly in another. On the other hand, one may have a solution which is not the best in any one objective, but overall is very competitive when taking all the objectives into account.

Numerous non-dominated sorting algorithms have been proposed over the years, and three of these will be used as benchmark algorithms. Their searching strategies are discussed next.

2.6.1 Naïve and slow sorting method

The naïve and slow non-dominated sorting approach is the easiest to understand and typically used for explanation purposes, especially when introducing the concept of non-domination. In this approach, to identify the first non-dominated set of solutions, every solution in the population needs to be compared to every other solution (Mishra & Harit, 2010). This computation requires $O(MN)$ comparisons for each solution in the population with M being the number of objectives. Continuing this procedure for all solutions of the first non-dominated front requires complexity $O(MN^2)$. At this point, the above procedure would have allowed for the all Rank 1 solutions to be identified. To continue non-dominated sorting on the remaining solutions in the population, all Rank 1 solutions already identified need to be temporarily removed from further consideration and the same procedure repeated. A worst case complexity scenario exists when $O(N)$ number of solutions form part of the second or higher-order non-dominated sets. The same worst case scenario holds for the remaining non-dominated sets. Therefore, in the event that there are N solutions and N non-dominated sets (*i.e.* each non-dominated set holds only one solution), then the complexity for the algorithm is $O(MN^3)$. The storage requirement for the naïve and slow algorithm is $O(N)$.

Given the advances in the field of non-dominated sorting, the naïve and slow technique is not competitive in practical applications. Its inclusion in this study is intended as a simple, introductory technique to facilitate a fundamental understanding of the task required of non-dominated sorting.

2.6 Non-dominated sorting

2.6.2 Fast non-dominated sort

Fast non-dominated sort (FNDS) is a variation of the naïve and slow algorithm, but requires far fewer comparisons. This is achieved by calculating two entities for each solution in the population, namely a domination count n_p which is the number of solutions which dominate solution p , and s_p which is the set of solutions which solution p dominates. Since all solutions that form part of the first non-dominated set are not dominated by any other solutions, their domination count n_p will be zero. Then, the set s_p of every solution whose n_p is zero is addressed, and its n_p reduced by one. Of these solutions, those whose n_p is now equal to zero forms part of the second non-dominated set. The procedure is then repeated until all solutions have been assigned to their respective non-dominated sets. The addition of these entities allows for the algorithm complexity to be reduced to $O(MN^2)$, however the storage requirements are increased to $O(MN^2)$.

FNDS is the non-dominated sorting algorithm originally introduced in [Deb *et al.*'s NSGA II](#), and is employed in many other multi-objective optimisation algorithms, including NSGA III, AGE-MEOA and A-NSGA III. However, with the advent of more recent non-dominated sorting algorithms, its performance is regarded as suboptimal ([McClymont & Keedwell, 2012](#); [Roy *et al.*, 2016](#); [Zhang *et al.*, 2015](#)), and thus is not included in the tests conducted in this research.

2.6.3 Deductive Sort

Deductive sort (DS) was proposed by [McClymont & Keedwell \(2012\)](#) and takes advantage of the transitive properties of dominance. DS attempts to reduce the number of comparisons required to find the Pareto sets by assessing each solution based on the population's fixed natural order. The algorithm compares each solution to all following solutions in the population, not preceding ones. The concept is illustrated in [Figure 2.11](#).

If a preceding solution is found to be dominated by the current solution, it is flagged as dominated and will not be assessed later for the current set in question. Conversely, if the current solution is found to be dominated by a preceding solution, the current solution will be flagged as dominated and will not undergo further assessment. When the algorithm reaches a solution already

2.6 Non-dominated sorting

marked as dominated by one of the preceding solutions, it is skipped. Once the entire population is traversed and all correct solutions are added to the set in question, the process repeats itself while ignoring all solutions already assigned to a previous set until all solutions are assigned. Figure 2.11 is provided to explain the DS operation.

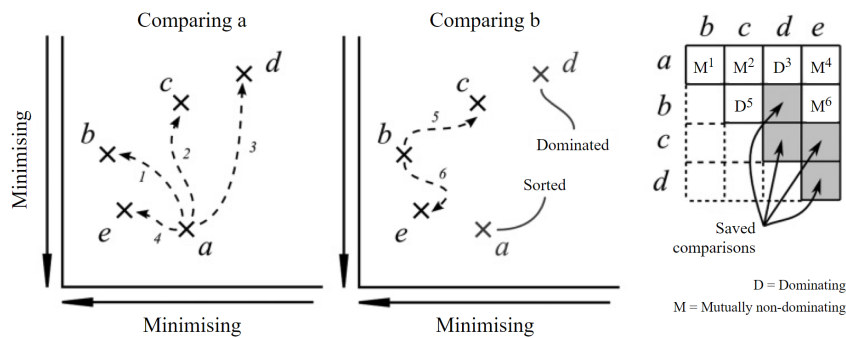


Figure 2.11: Illustration of Deductive Sort, [McClymont & Keedwell \(2012\)](#)

The algorithm approaches the population to be sorted in its original order. For ease of explanation, the solutions in Figure 2.11 are labelled a to e , the alphabetical order indicating the population's original order. Firstly, a is compared to the next solution in the population, b , which is found to be mutually non-dominating. Next, a is compared to c , which is also found to be mutually non-dominating. a is then compared to d , which it dominates. Therefore, d is marked as dominated and will not undergo assessment for this first non-dominated set in question. Lastly, a is compared to e which is also assessed to be mutually non-dominating. After having been compared to all solutions in the population without being dominated, a is inserted into the first non-dominated set.

Having compared a to all solutions in the population (because none of the solutions b to e dominated a), the next solution to undergo assessment is b since it was not marked as dominated by a . Since the comparison between b and a has already taken place, the first comparison will be between solutions b and c . Since c is dominated by b , c will be marked as such and not undergo assessment for this first non-dominated set. d was already marked as dominated in a previous comparison against a , and will therefore not require comparison against b .

2.6 Non-dominated sorting

Therefore, the last solution to which b needs to be compared is e , which is found to be mutually non-dominating. Again, like a , solution b has been compared to all necessary solutions without being dominated and is therefore inserted into the first non-dominated set.

Solution c was marked as dominated by b , and therefore does not undergo assessment for this first non-dominated set. Similarly, d was marked by a as dominated, and also does not undergo assessment for the first non-dominated set. Lastly, e , which has not been marked as dominated, is reached and has no other solutions to which it must be compared. Therefore it is inserted into the first non-dominated set. This concludes assigning solutions to the first non-dominated set. Should additional non-dominated sets be required, the solutions forming the first non-dominated set would be removed from the population and the process repeated.

The pseudocode for DS is provided in Algorithm 1, and its MATLAB code provided in Appendix B. It is worth noting that in line 12, the $dominates(Pop(j), Pop(i))$ function which computes the dominance relation between input solutions x and y has three possible outcomes; 1 if y dominates x ; 2 if x is mutually non-dominated with respect to y ; and 3 if x dominates y .

DS has storage complexity $\mathcal{O}(N)$. Best case time complexity for deductive sort is where the first set consists of only one dominating solution and only one set is desired, requiring $\mathcal{O}(MN \log N)$. The worst case time complexity is realised if all sets are desired, or when all solutions form part of the first front, requiring $\mathcal{O}(MN^2)$.

2.6 Non-dominated sorting

Algorithm 1 Deductive Sort

```

1: procedure DEDUCTIVE SORT( $Pop$ )
2:    $x \leftarrow 0$  ▷ Set first set
3:    $f \leftarrow 0$  ▷ Set number sorted to 0
4:    $\mathcal{F} = \emptyset$  ▷ Initialise sets
5:   while  $f < |Pop|$  do ▷ While not all are sorted
6:      $\mathcal{F}_+ =$  ▷ Append a new (empty) set
7:      $D = false(|Pop|)$  ▷ Clear dominated flag array
8:     for  $i = 0$  to  $(|Pop| - 1)$  do ▷ Iterate through solutions
9:       if  $D(i) = false$  and  $Pop(i) \notin \mathcal{F}$  then ▷ If  $Pop(i)$  not dominated
or sorted
10:        for  $j = i + 1$  to  $|Pop| - 1$  do ▷ From next solution to end
11:          if  $D(j) = false$  and  $Pop(j) \notin \mathcal{F}$  then ▷ If  $Pop(i)$  not
dominated or sorted
12:             $d = dominates(Pop(j), Pop(i))$  ▷ Compute relation
13:            if  $d = 1$  then ▷ If  $Pop(j)$  dominated
14:               $D(j) = true$  ▷ flag  $Pop(j)$  as dominated
15:            else if  $d = 3$  then ▷ else if  $Pop(i)$  dominated
16:               $D(i) = true$  ▷ flag  $Pop(i)$  as dominated
17:              break
18:            end if
19:          end if
20:        end for
21:        if  $D(i) = false$  then ▷ if  $Pop(i)$  is not dominated
22:           $\mathcal{F}(x)_+ = Pop(i)$  ▷ insert  $Pop(i)$  into  $\mathcal{F}(x)$ 
23:           $f = f + 1$  ▷ increment number sorted
24:        end if
25:      end if
26:    end for
27:     $x = x + 1$  ▷ increment current set
28:  end while
29: end procedure

```

2.6 Non-dominated sorting

2.6.4 Efficient Non-dominated Sort (ENS)

Zhang *et al.* (2015) proposed the Efficient Non-dominated Sorting algorithm (ENS) which employs a conceptually different approach to most other non-dominated sorting algorithms. In most non-dominated sorting algorithms, processing takes place on a set by set basis. In other words, all solutions are evaluated and if they belong to the set in question, then they are assigned accordingly. Once the whole population has been checked, the solutions belonging to the first non-dominated set are removed from the population, and the process is repeated for the solutions remaining in the population. For example, a population Pop which contains L non-dominated sets $F_i, i \in [1, L]$ would first have all solutions belonging to the first non-dominated set assigned. Next, the non-dominated sorting procedure is typically performed on the remaining $Pop - F_1$ solutions (those not assigned to the first non-dominated set). This concept is illustrated in Figure 2.12.

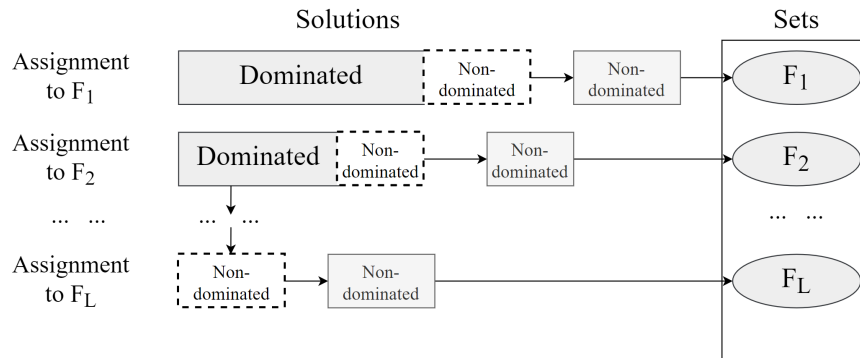


Figure 2.12: Assigning solutions per non-dominated set, Zhang *et al.* (2015)

The way that ENS differs in its approach is by determining the relevant non-dominated set to which the current solution in question belongs. This approach is illustrated in Figure 2.13.

2.6 Non-dominated sorting

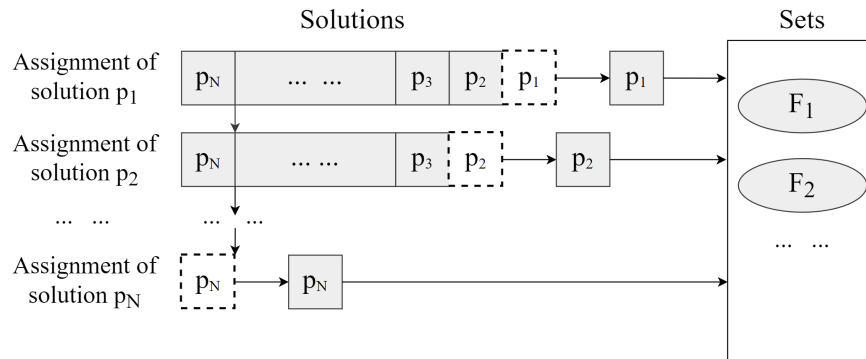


Figure 2.13: Assigning solutions to its relevant non-dominated set as it is processed , [Zhang *et al.* \(2015\)](#)

By processing solutions one by one and assigning them to their applicable non-dominated set, ENS avoids duplicate comparisons as a solution which needs to be assigned only needs to be compared against those solutions which have already been assigned. In order to ensure that this statement is true, ENS begins by sorting the population by its first objective in ascending order (assuming a minimisation problem). Therefore, no solution can dominate a preceding solution, as any preceding solution is superior in the first objective. The possible relationships between a solution undergoing processing and those that have already been assigned are illustrated in Figure 2.14.

2.6 Non-dominated sorting

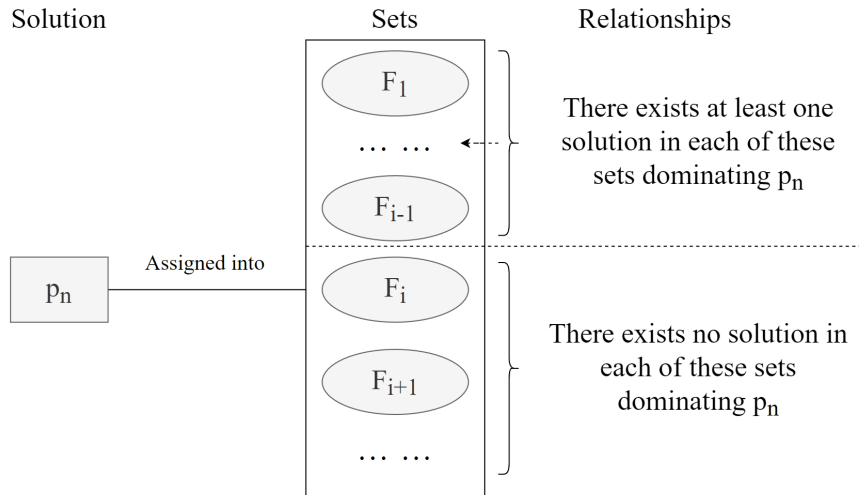


Figure 2.14: The relationships between p_n and the solutions having been assigned to a set, Zhang *et al.* (2015)

Zhang *et al.* (2015) also investigated the type of comparisons between different solutions in a population. They found that there are four different cases of comparisons that exist which are summarised in Figure 2.15.

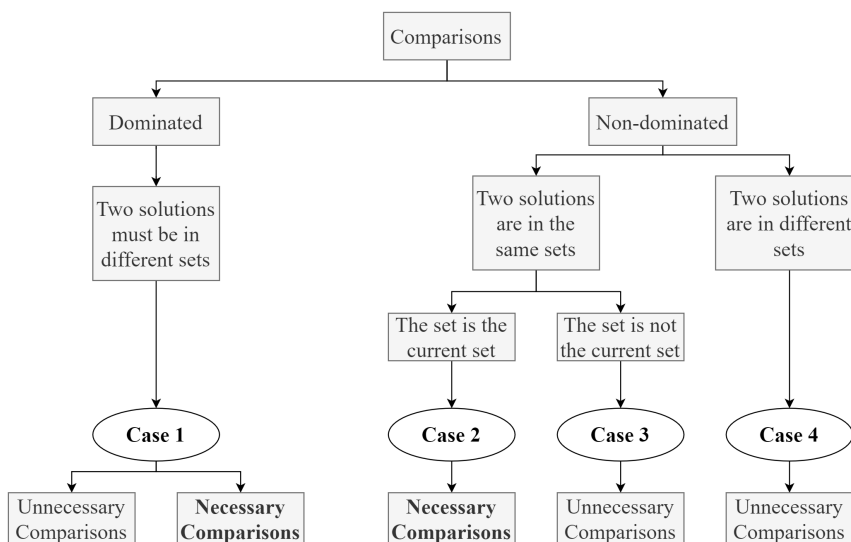


Figure 2.15: Categorising dominance comparisons, Zhang *et al.* (2015)

2.6 Non-dominated sorting

As depicted by the tree-diagram, *Zhang et al.* suggest that dominance ranking can be categorised into four cases as follows:

- Case 1: x is dominated by y , or y is dominated by x .
- Case 2: x and y are non-dominated and both belong to the same current non-dominated set F_i .
- Case 3: x and y are non-dominated and both belong to the same non-current non-dominated set F_i .
- Case 4: x and y are non-dominated, but belong to different sets.

Furthermore, *Zhang et al.* suggest that the only comparisons necessary to ascertain dominance are some comparisons in Case 1, and all comparisons in Case 2. Cases 3 and 4 are deemed to be unnecessary comparisons, and as such, are ignored in the Efficient Non-dominated Sorting algorithm. Accordingly, the authors report that the sum of the necessary comparisons in Case 1 and Case 2 is the theoretical minimum number of required dominance comparisons for any non-dominated sorting algorithm.

To illustrate the efficiency provided by ENS, an example population is provided in Figure [2.16](#).

2.6 Non-dominated sorting

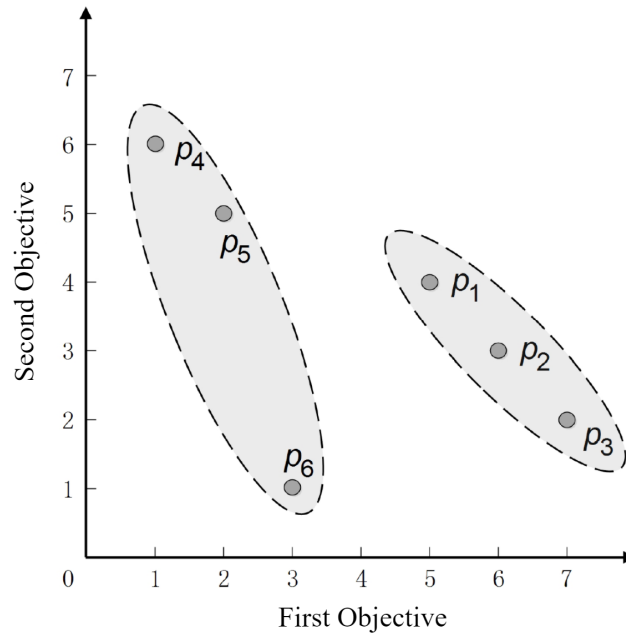


Figure 2.16: A population containing six solutions for a bi-objective minimisation problem, [Zhang *et al.* \(2015\)](#)

Table 2.2 summarises the comparisons performed by [McClymont & Keedwell's](#) Deductive Sort on the population illustrated in Figure 2.16. As indicated, DS requires 18 comparisons to complete the non-dominated sorting procedure. In comparison, Table 2.3 summarises the comparisons required by ENS to perform non-dominated sorting on the same population. Compared to DS, ENS required half of the number of comparisons.

The pseudocode for ENS is provided in Algorithm 2, and its `MATLAB` code provided in Appendix B. ENS has space complexity $\mathcal{O}(1)$. [Zhang *et al.* \(2015\)](#) propose two methods for a specific step in the algorithm, either a sequential search strategy or a binary search strategy. The method employed in the pseudocode of Algorithm 2 and the `MATLAB` code in Appendix B is the sequential sort version. Best case time complexity for ENS (sequential search) is $\mathcal{O}(MN\sqrt{N})$ and best case for ENS (binary search) is $\mathcal{O}(MN \log N)$ while the worst case time complexity for both ENS methods is $\mathcal{O}(MN^2)$.

2.6 Non-dominated sorting

Table 2.2: Comparisons performed by Deductive Sort for the population shown in Figure 2.16, (Zhang *et al.*, 2015)

Front	Comparison	Operation	Comparison Result	
F_1	(p_1, p_2)		Case 3	
	(p_1, p_3)		Case 3	
	(p_1, p_4)		Case 4	
	(p_1, p_5)		Case 4	
	(p_1, p_6)	p_1 is ignored	Case 1	
	(p_2, p_3)		Case 3	
	(p_2, p_4)		Case 4	
	(p_2, p_5)		Case 4	
	(p_2, p_6)	p_2 is ignored	Case 1	
	(p_3, p_4)		Case 4	
	(p_3, p_5)		Case 4	
	(p_3, p_6)	p_3 is ignored	Case 1	
	(p_4, p_5)		Case 2	
	(p_4, p_6)	p_4 is assigned into F_1	Case 2	
	(p_5, p_6)	p_5 and p_6 are assigned into F_1	Case 2	
F_2	(p_1, p_2)		Case 2	
	(p_1, p_3)	p_1 is assigned into F_2	Case 2	
	(p_2, p_3)	p_2 and p_3 are assigned into F_2	Case 2	
Case 1: 3		Case 2: 6	Case 3: 3	Case 4: 6
Total: 18				

Table 2.3: Comparisons performed by ENS for the population shown in Figure 2.16, (Zhang *et al.*, 2015)

Assigned Solution	Comparison	Operation	Comparison Result	
p_4		p_4 is assigned into F_1		
p_5	(p_5, p_4)	p_5 is assigned into F_1	Case 2	
p_6	(p_6, p_5)		Case 2	
	(p_6, p_4)	p_6 is assigned into F_1	Case 2	
p_1	(p_1, p_6)	p_1 is assigned into F_2	Case 1	
p_2	(p_2, p_6)		Case 1	
	(p_2, p_1)	p_2 is assigned into F_2	Case 2	
p_3	(p_3, p_6)		Case 1	
	(p_3, p_2)		Case 2	
	(p_3, p_1)	p_3 is assigned into F_2	Case 2	
Case 1: 3		Case 2: 6	Case 3: 0	Case 4: 0
Total: 9				

2.6 Non-dominated sorting

Algorithm 2 Efficient Non-dominated Sort (Sequential Sort)

```

1: procedure ENS( $Pop$ ) ▷ The population
2:    $F \leftarrow \emptyset$ 
3:   Sort  $Pop$  in ascending order of objective 1
4:   for all  $Pop[n] \in Pop$  do
5:      $x \leftarrow size(F)$  ▷ number of sets found so far
6:      $k \leftarrow 1$  ▷ Set now being checked
7:     while true do
8:       Compare  $Pop(n)$  with the solutions in  $F(k)$  starting from the last
9:       one and ending with the first one
10:      if  $F(k)$  contains no solutions dominating  $Pop(n)$  then
11:        move  $Pop(n)$  to  $F(k)$ 
12:        break
13:      else
14:         $k++$ 
15:        if  $k > x$  then
16:          move  $Pop(n)$  to a new set
17:          break
18:        end if
19:      end if
20:    end while
21:  end for
22:  return  $F$ 
23: end procedure

```

2.6.5 Best Order Sort

Roy *et al.* (2016) proposed the Best Order Sort (BOS) algorithm which reduces the number of solution comparisons required in a worst case scenario. The basic premise of their algorithm is that for each solution p_i , $i \in [1, N]$, one can find a set T_j^i for each objective $j \in [1, M]$ which denotes the solutions that are not worse than p_i in that objective. Therefore, in order to find the rank of p_i , it would be sufficient to only consider one of the sets T_j^i . The concept is illustrated in Figure 2.17.

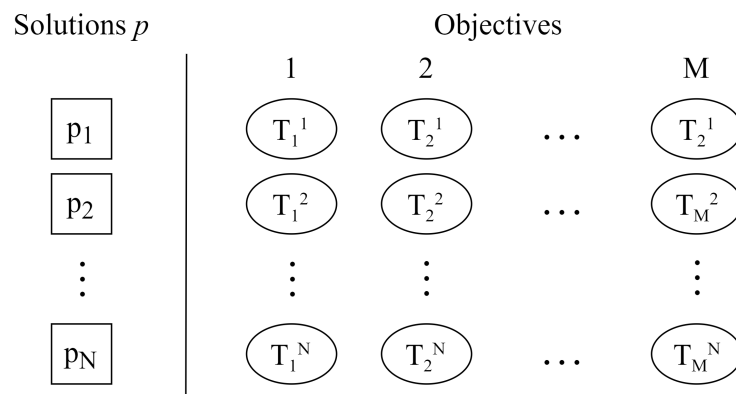


Figure 2.17: Illustration of the concept behind BOS

Some elements $t \in T$ dominate p , whilst others are non-dominated with p . M objective sets exist, each of which could be considered in isolation to evaluate the rank of p . However, in order to reduce the number of comparisons to a minimum, BOS will choose the smallest set T to evaluate the rank of p . Once p has been compared against all the solutions occupying this smallest set, the solutions which dominate p will be used to calculate what the rank of p should be. The rank of p will be the highest rank k of these solutions (solutions with higher ranks are dominated by solutions with lower ranks) plus 1.

Mishra *et al.* (2018) analysed BOS and found that the algorithm did not perform well when duplicate solutions were present in the population. In 2018, these authors published Modified Best Order Sort (MBOS) which included an additional step to facilitate the handling of duplicate solutions.

2.6 Non-dominated sorting

The pseudocode for BOS is provided in Algorithm 3.

Algorithm 3 Best Order Sort

BOS 3.1: INITIALISATION(Pop)

```

1:  $N \leftarrow$  number of solutions in  $Pop$ 
2:  $M \leftarrow$  number of objectives in  $Pop$ 
3:  $L_j^i \leftarrow \emptyset, \forall j \in [1, M], \forall i \in [1, N]$ 
4:  $C_i \leftarrow \{1, 2, \dots, M\} \forall i \in [1, N]$  ▷ Comparison set
5:  $isRanked(P) \leftarrow false$  ▷ Solution ranked or not
6:  $SC \leftarrow 0$ 
7:  $RC \leftarrow 1$ 
8:  $R(P) \leftarrow 0$ 
9: for  $j = 1 : M$  do
10:    $Q_j \leftarrow$  Sort  $P$  by objective  $j$ 
11: end for

```

BOS 3.2: MAIN LOOP(sorted population Q)

```

12: for  $i = 1 : N$  do
13:   for  $j = 1 : M$  do
14:      $s \leftarrow Q_j(i)$  ▷ Take  $i^{th}$  element from  $Q$ 
15:      $C_s \leftarrow C_s - \{j\}$  ▷ Reduce comparison set
16:     if  $isRanked(s) = True$  then
17:        $L_j^{R(s)} = L_j^{R(s)} \cup \{s\}$  ▷ Include  $s$  in  $L_j^{R(s)}$ 
18:     else
19:        $FINDRANK(s, j)$  ▷ Find  $R(s)$ 
20:        $isRanked(s) \leftarrow True$  ▷ Non-dominated sorting done
21:        $SC \leftarrow SC + 1$ 
22:     end if
23:   end for
24:   if  $SC = N$  then
25:      $break$ 
26:   end if
27: end for

```

2.6 Non-dominated sorting

BOS 3.3: FINDRANK(s, j)

```

28: done = False                                ▷ Done bit
29: for  $k=1:RC$  do                                ▷ For all discovered ranks
30:   check = False                                ▷ Check bit
31:   for  $t \in L_j^k$  do
32:     check  $\leftarrow$  DOMINATIONCHECK( $s, t$ )
33:     if check = True then                        ▷ If dominated
34:       break                                       ▷ Break the loop
35:     end if
36:   end for
37:   if check = False then                        ▷ Rank found
38:     R( $S$ ) gets  $k$                                 ▷ Update rank
39:     done = True                                    ▷ Update done bit
40:      $L_j^{R(s)} = L_j^{R(s)} \cup \{s\}$            ▷ Include  $s$  to  $L_j^{R(s)}$ 
41:     break                                       ▷ Break the loop
42:   end if
43: end for
44: if done = False then                            ▷ If not done
45:    $RC \leftarrow RC + 1$                             ▷ Update fronts count
46:    $R(s) \leftarrow RC$                                 ▷ Update rank
47:    $L_j^{R(s)} = L_j^{R(s)} \cup \{s\}$            ▷ Include  $s$  to  $L_j^{R(s)}$ 
48: end if

```

BOS 3.4: DOMINATIONCHECK(s, t)

```

49: for  $j \in C_t$  do
50:   if  $s_j \prec t_j$  then
51:     return false
52:   end if
53: end for

```

BOS has space complexity $\mathcal{O}(MN)$. Best case time complexity for BOS is $\mathcal{O}(MN \log N)$ and worst case time complexity is $\mathcal{O}(MN^2)$.

2.6 Non-dominated sorting

2.6.6 Other non-dominated sorting algorithms

There are numerous other non-dominated algorithms that exist, but these will not be discussed in detail. The primary reason for this is due to the scoping boundaries set which state that this study will be conducted in a `MATLAB` environment.

For the non-dominated sorting algorithms mentioned in this subsection, no `MATLAB` code was available. Efforts were made to create a `MATLAB` function using the algorithm's original pseudocode, but during testing and debugging, it was found that the `MATLAB` implementations did not reflect the reported performance by the original authors of these algorithms. Therefore, these non-dominated sorting algorithms are acknowledged, but excluded from this research.

2.6.6.1 Corner Sort

Corner Sort (CS) is an algorithm developed by [Wang & Yao \(2014\)](#). Similar to Deductive Sort, albeit via a different execution, CS aims to reduce unnecessary comparisons by using non-dominated solutions to ignore those solutions which they dominate. It does this more efficiently than deductive sort by starting with known non-dominated solutions, focussing on corner solutions (see [Figure 2.18](#)). After one non-dominated solution is obtained, CS identifies solutions dominated by it and ignores them in the later stages of the sort for the current rank. The two main loops in this algorithm are finding a non-dominated solution, and then identifying solutions that are dominated by that non-dominated solution. In doing so, a number of solutions may be identified as dominated and thus ignored for unnecessary further comparisons. CS also capitalises on the fact that most real-world problems are based on preference. By the user selecting preferential objectives, corner sort can make non-dominated sorting more efficient.

2.6 Non-dominated sorting

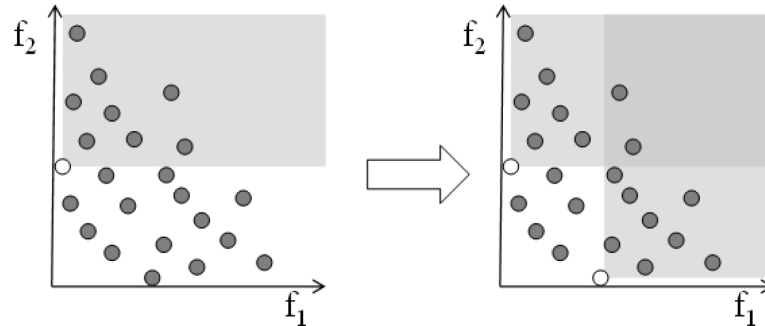


Figure 2.18: The basic premise of Wang's corner sort – using corner solutions to ignore dominated solutions, Wang & Yao (2014)

2.6.6.2 Merge Non-dominated Sort

Merge Non-dominated Sort (MNDS), (Moreno *et al.*, 2018), is one of the most recent non-dominated sorting algorithms. MNDS performs non-dominated sorting by finding a *dominance set* for each solution in the population. A dominance set contains all the solutions which dominate the solution in question. Once the dominance sets of all solutions have been found, the Rank 1 solutions are identified as those solutions whose dominance sets are empty. Then similarly to BOS, the non-dominated rank of solutions which contain other solutions in their dominance set is calculated by finding the highest (worst) ranking solution in that dominance set and adding one to that rank.

The concept of a dominance set is illustrated in Table 2.4 where solutions a to f and their objective values are provided. Assuming a minimisation problem, the dominance set of each solution p_i , $i \in [a, f]$ contains those solutions which dominate p_i . As solutions a, b and e have empty dominance sets, they are categorised as Rank 1 solutions. Solution c contains solutions a and b in its dominance set, both of which are Rank 1 solutions. Therefore, solution c is categorised as Rank 2. Since solution d only has solution b (Rank 1) in its dominance set, it is also categorised as a Rank 2 solution. Finally, solution f has a dominance set containing solutions a, b, d and e . The highest (worst) ranking solution in this set is solution d which is Rank 2. Therefore, solution f is categorised as Rank 3.

2.7 Standardised benchmark test suites

Table 2.4: An example of dominance sets used in MNDS

Solution	Objectives	Dominance Set	Rank
a	{33, 34, 30}	\emptyset	1
b	{32, 32, 31}	\emptyset	1
c	{39, 37, 31}	{a, b}	2
d	{36, 33, 32}	{b}	2
e	{31, 34, 34}	\emptyset	1
f	{38, 38, 37}	{a, b, d, e}	3

2.7 Standardised benchmark test suites

In order to ensure that fair and accurate testing is conducted, it is necessary to test the multi-objective optimisation algorithms on test suites which have undergone thorough scrutiny and review. Three test suites were chosen, all of which offer a variety of multi-objective optimisation problems (MOPs) and are presented next. For each problem within each test suite, a summary of its formulation and characteristics are provided, along with a figure representing its Pareto optimal front.

2.7.1 The DTLZ Benchmark MOPs

The DTLZ (Deb-Thiele-Laumanns-Zitzler) test suite ([Deb *et al.*, 2005](#)) is scalable in terms of the number of dimensions that are possible. Therefore, this test suite is well suited for assessing performance of multi-objective evolutionary algorithms in the many-objective space. The true Pareto fronts are also known, which is useful for calculating the convergence performance that an algorithm achieved on the problem. Tables [2.5](#) and [2.6](#) introduces the DTLZ problems and the construction of each.

2.7 Standardised benchmark test suites

Table 2.5: Standard DTLZ test problems

Function	Definition	Constraints
DTLZ1 (Min)	$f_1(\mathbf{x}) = \frac{1}{2}x_1x_2\dots x_{M-1}(1+g(\mathbf{x}_M))$ $f_2(\mathbf{x}) = \frac{1}{2}x_1x_2\dots(1-x_{M-1})(1+g(\mathbf{x}_M))$ \vdots $f_{M-1}(\mathbf{x}) = \frac{1}{2}x_1(1-x_2)(1+g(\mathbf{x}_M))$ $f_M(\mathbf{x}) = \frac{1}{2}(1-x_1)(1+g(\mathbf{x}_M))$ $g(\mathbf{x}_M) = 100[\mathbf{x}_M + \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5))]$	$0 \leq x_i \leq 1$, for $i=1,2,\dots,n$ $\mathbf{x}_M = k = (n-M+1)$, $ \mathbf{x}_M = k, g \geq 0$
DTLZ2 (Min)	$f_1(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1\pi/2) \dots \cos(x_{M-2}\pi/2) \cos(x_{M-1}\pi/2)$ $f_2(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1\pi/2) \dots \cos(x_{M-2}\pi/2) \sin(x_{M-1}\pi/2)$ $f_3(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1\pi/2) \dots \sin(x_{M-2}\pi/2)$ \vdots $f_M(\mathbf{x}) = (1+g(\mathbf{x}_M)) \sin(x_1\pi/2)$ $g(\mathbf{x}_M) = \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2$	$0 \leq x_i \leq 1$, for $i=1,2,\dots,n$ $\mathbf{x}_M = k = (n-M+1)$
DTLZ3 (Min)	$f_1(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1\pi/2) \dots \cos(x_{M-2}\pi/2) \cos(x_{M-1}\pi/2)$ $f_2(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1\pi/2) \dots \cos(x_{M-2}\pi/2) \sin(x_{M-1}\pi/2)$ $f_3(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1\pi/2) \dots \sin(x_{M-2}\pi/2)$ \vdots $f_M(\mathbf{x}) = (1+g(\mathbf{x}_M)) \sin(x_1\pi/2) \dots \sin(x_1\pi/2)$ $g(\mathbf{x}_M) = 100[\mathbf{x}_M + \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5))]$	$0 \leq x_i \leq 1$, for $i=1,2,\dots,n$ $\mathbf{x}_M = k = (n-M+1)$
DTLZ4 (Min)	$f_1(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1^\alpha\pi/2) \dots \cos(x_{M-2}^\alpha\pi/2) \cos(x_{M-1}^\alpha\pi/2)$ $f_2(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1^\alpha\pi/2) \dots \cos(x_{M-2}^\alpha\pi/2) \sin(x_{M-1}^\alpha\pi/2)$ $f_3(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(x_1^\alpha\pi/2) \dots \sin(x_{M-2}^\alpha\pi/2)$ \vdots $f_M(\mathbf{x}) = (1+g(\mathbf{x}_M)) \sin(x_1^\alpha\pi/2)$ $g(\mathbf{x}_M) = \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2$	$0 \leq x_i \leq 1$, for $i=1,2,\dots,n$ $\mathbf{x}_M = k = (n-M+1)$ $\alpha=3$
DTLZ5 (Min)	$f_1(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(\theta_1\pi/2) \dots \cos(\theta_{M-2}\pi/2) \cos(\theta_{M-1}\pi/2)$ $f_2(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(\theta_1\pi/2) \dots \cos(\theta_{M-2}\pi/2) \sin(\theta_{M-1}\pi/2)$ $f_3(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(\theta_1\pi/2) \dots \sin(\theta_{M-2}\pi/2)$ \vdots $f_M(\mathbf{x}) = (1+g(\mathbf{x}_M)) \sin(\theta_1\pi/2)$ $\theta_i = \frac{\pi}{4(1+g(\mathbf{x}_M))} (1+2g(\mathbf{x}_M)x_i)$, for $i=2,3,\dots,(M-1)$ $g(\mathbf{x}_M) = \sum_{x_i \in \mathbf{x}_M} (x_i - 0.5)^2$	$0 \leq x_i \leq 1$, for $i=1,2,\dots,n$ $\mathbf{x}_M = k = (n-M+1)$

2.7 Standardised benchmark test suites

Table 2.6: Standard DTLZ test problems continued

Function	Definition	Constraints
DTLZ6 (Min)	$f_1(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(\theta_1 \pi/2) \dots \cos(\theta_{M-2} \pi/2) \cos(\theta_{M-1} \pi/2)$ $f_2(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(\theta_1 \pi/2) \dots \cos(\theta_{M-2} \pi/2) \sin(\theta_{M-1} \pi/2)$ $f_3(\mathbf{x}) = (1+g(\mathbf{x}_M)) \cos(\theta_1 \pi/2) \dots \sin(\theta_{M-2} \pi/2)$ \vdots \vdots $f_M(\mathbf{x}) = (1+g(\mathbf{x}_M)) \sin(\theta_1 \pi/2)$ $\theta_i = \frac{\pi}{4(1+g(\mathbf{x}_M))} (1+2g(\mathbf{x}_M)x_i), \text{ for } i=2,3,\dots,(M-1)$ $g(\mathbf{x}_M) = \sum_{x_i \in \mathbf{x}_M} x_i^0.1$	$0 \leq x_i \leq 1, \text{ for } i=1,2,\dots,n$ $\mathbf{x}_M = k = (n-M+1)$
DTLZ7 (Min)	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}) = x_2$ \vdots \vdots $f_{M-1}(\mathbf{x}_{M-1}) = x_{M-1}$ $f_M(\mathbf{x}) = (1+g(\mathbf{x}_M))h(f_1, f_2, \dots, f_{M-1}, g)$ $g(\mathbf{x}_M) = 1 + \frac{9}{ \mathbf{x}_M } \sum_{x_i \in \mathbf{x}_M} x_i$ $h(f_1, f_2, \dots, f_{M-1}, g) = M - \sum_{i=1}^{M-1} \left[\frac{f_i}{1+g} (1 + \sin(3\pi f_i)) \right]$	$0 \leq x_i \leq 1, \text{ for } i=1,2,\dots,n$ $\mathbf{x}_M = k = (n-M+1)$
DTLZ8 (Min)	$f_j(\mathbf{x}) = \frac{1}{\lfloor \frac{n}{M} \rfloor} \sum_{i=\lfloor (j-1) \frac{n}{M} \rfloor}^{\lfloor j \frac{n}{M} \rfloor} x_i$ $g_j(\mathbf{x}) = f_M(\mathbf{x}) + 4f_j(\mathbf{x}) - 1 \geq 0, \text{ for } j=1,2,\dots,(M-1)$ $g_M(\mathbf{x}) = 2f_M(\mathbf{x}) + \min_{i,j=1,i \neq j}^{M-1} [f_i \mathbf{x} + f_j \mathbf{x}] - 1 \geq 0$	$0 \leq x_i \leq 1, \text{ for } i=1,2,\dots,n$ $j=1,2,\dots,M$
DTLZ9 (Min)	$f_j(\mathbf{x}) = \sum_{i=\lfloor (j-1) \frac{n}{M} \rfloor}^{\lfloor j \frac{n}{M} \rfloor} x_i$ $g_j(\mathbf{x}) = f_M^2(\mathbf{x}) + 4f_j^2(\mathbf{x}) - 1 \geq 0, \text{ for } j=1,2,\dots,(M-1)$	$0 \leq x_i \leq 1, \text{ for } i=1,2,\dots,n$ $j=1,2,\dots,M$

2.7 Standardised benchmark test suites

DTLZ1 is a problem which has a linear, separable, multi-modal Pareto front \mathcal{PF}_{True} . The Pareto-optimal solutions are found at $x_M^* = 0$ with the values of the objective functions lying on the hyperplane $\sum_{m=1}^M = 0.5$. The massive search space has $11^k - 1$ local Pareto fronts which can each attract the multi-objective evolutionary algorithm. Figure 2.19 represents the Pareto front \mathcal{PF}_{True} of DTLZ1.

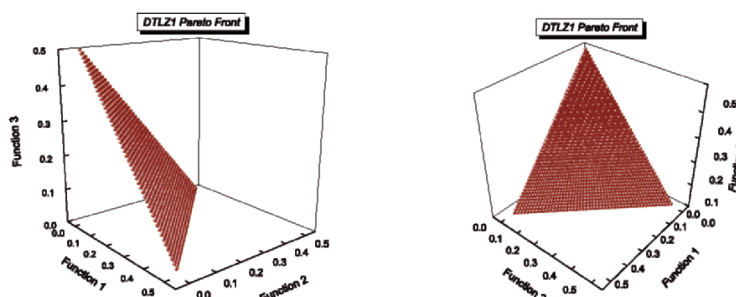


Figure 2.19: Pareto front of DTLZ1 (Coello *et al.*, 2007)

DTLZ2 has Pareto solutions corresponding to $x_i = 0.5$ for all $x_i \in x_M$. Figure 2.20 represents the Pareto front \mathcal{PF}_{True} of DTLZ2.

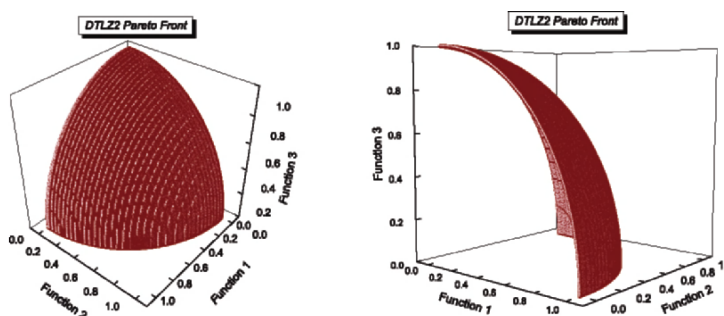
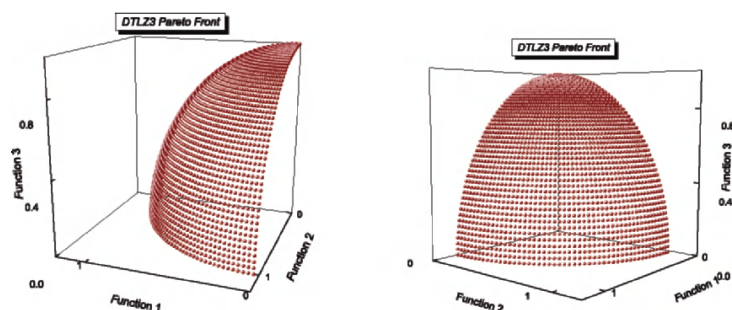


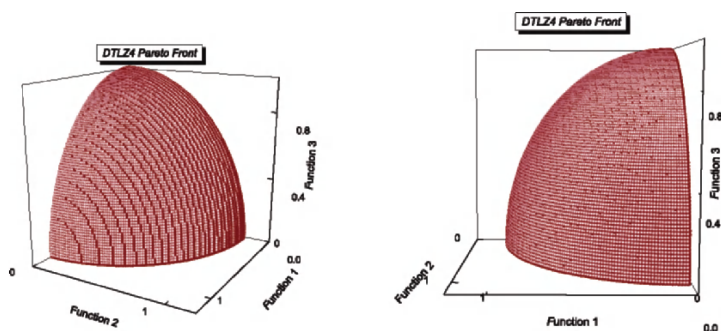
Figure 2.20: Pareto front of DTLZ2 (Coello *et al.*, 2007)

DTLZ3 is the same as DTLZ2, except that a new g function is employed, which provides $3k - 1$ local Pareto fronts, all of which are parallel to one global Pareto front. Figure 2.21 illustrates the Pareto front \mathcal{PF}_{True} is concave, multi-modal and scalable, and is found corresponding to $g^* = 0$ and $x_M = (0.5, \dots, 0.5)^T$.

2.7 Standardised benchmark test suites

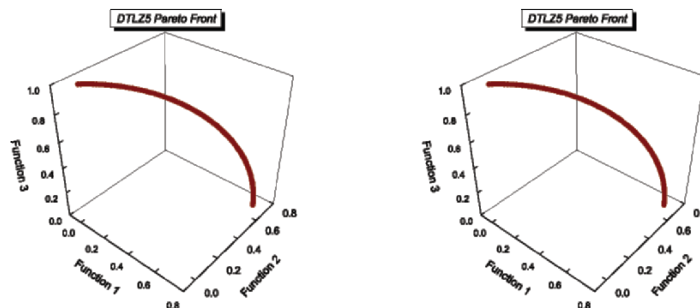
Figure 2.21: Pareto front of DTLZ3 (Coello *et al.*, 2007)

DTLZ4 makes use of an altered variable mapping ($y \leftarrow y^\alpha, \alpha > 0$). A suggestion is made to set $\alpha = 100$ and $k = 10$. The Pareto front \mathcal{PF}_{True} is concave, unimodal and separable and is presented in Figure 2.22.

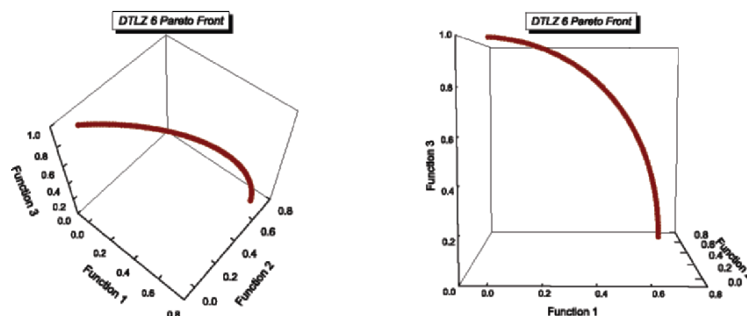
Figure 2.22: Pareto front of DTLZ4 (Coello *et al.*, 2007)

DTLZ5 is another modification of DTLZ2 by replacing all \mathbf{y} with $(1 + 2gf_i)/2(1 + g)$. The Pareto front \mathcal{PF}_{True} corresponding to $x_i = 0.5 \forall x_i \in x_M$ is unimodal and presented in Figure 2.23.

2.7 Standardised benchmark test suites

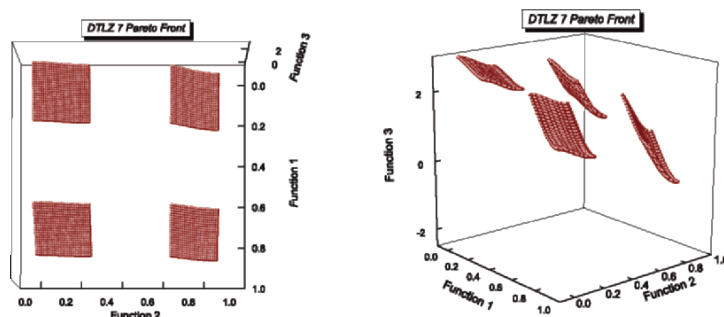
Figure 2.23: Pareto front of DTLZ5 (Coello *et al.*, 2007)

DTLZ6 is created by modifying DTLZ5 into a more difficult problem by replacing g with $\sum_{i=1}^k z_i^{0.1}$. The Pareto front \mathcal{PF}_{True} corresponding to $x_i = 0 \forall x_i \in x_M$ is biased, unimodal, has many-to-one mapping and is presented in Figure 2.24.

Figure 2.24: Pareto front of DTLZ6 (Coello *et al.*, 2007)

DTLZ7 has $2M - 1$ disconnected Pareto segments in the search space. Suggestions are made for constants $k = 20$. The Pareto front \mathcal{PF}_{True} corresponding to $x_M = 0$ and is presented in Figure 2.25.

2.7 Standardised benchmark test suites

Figure 2.25: Pareto front of DTLZ7 (Coello *et al.*, 2007)

The multi-objective evolutionary algorithm MOPSO is unable to perform constraint handling, and therefore DTLZ8 and DTLZ9 are excluded from this study.

General characteristics of the DTLZ problems are summarised and presented in Table 2.7.

Table 2.7: Characteristics of the DTLZ test suite (Huband *et al.*, 2006)

Problem	Obj.	Separability	Modality	Bias	Geometry
DTLZ1	$f_{1:M}$	S	M	-	linear
DTLZ2	$f_{1:M}$	S	U	-	concave
DTLZ3	$f_{1:M}$	S	M	-	concave
DTLZ4	$f_{1:M}$	S	U	+	concave
DTLZ5	$f_{1:M}$	Unknown	U	-	Unknown
DTLZ6	$f_{1:M}$	Unknown	U	+	Unknown
DTLZ7	$f_{1:M-1}$	N/A	U	-	disconnected
	f_M	S	U		

2.7 Standardised benchmark test suites

2.7.2 The Walking Fish Group test suite

The Walking Fish Group (WFG) test suite consists of nine scalable, multi-objective test problems. The test suite was created by [Huband *et al.* \(2006\)](#) after a rigorous study into multi-objective evolutionary algorithm testing. Their research made specific suggestions regarding the creation of test problems, but also standardised definitions describing features of test problems, many of which have been introduced in Section 2.1. WFG problems are all created in a similar manner, where an underlying vector of parameters \mathbf{x} is used to define the problem. By means of a series of transition vectors, \mathbf{x} is derived from another vector of working parameters \mathbf{z} . Each of these transition vectors provide complexity to the problem such as non-separability and multi-modality. The multi-objective evolutionary algorithm tasked with optimising the problem manipulates the vector \mathbf{z} , thereby indirectly manipulating \mathbf{x} ([Huband *et al.*, 2006](#)). All WFG problems adhere to the following format:

$$\begin{aligned}
 \text{Given } \mathbf{z} &= z_1, \dots, z_k, z_{k+1}, \dots, z_n \\
 \text{Minimise } f_{m=1:M}(\mathbf{x}) &= Dx_M + S_m h_m(x_1, \dots, x_{M-1}) \\
 \text{where } \mathbf{x} &= \{x_1, \dots, x_M\} \\
 &= \{\max(t_M^p, A_1)(t_1^p - 0.5) + 0.5, \dots, \\
 &\quad \max(t_M^p, A_{M-1})(t_{M-1}^p - 0.5) + 0.5, t_M^p\} \\
 \mathbf{t}^p &= \{t_1^p, \dots, t_M^p\} \leftarrow \mathbf{t}^{p-1} \leftarrow \dots \leftarrow \mathbf{t}^1 \leftarrow \mathbf{z}_{[0,1]} \\
 \mathbf{z}_{[0,1]} &= \{z_{1,[0,1]}, \dots, z_{n,[0,1]}\} \\
 &= \{z_1/z_{1,max}, \dots, z_n/z_{n,max}\}
 \end{aligned}$$

where M is the number of objectives, the set \mathbf{x} is made up of M underlying parameters, $D > 0$ is a distance scaling constant, \mathbf{z} is a set of $k + l = n \geq M$ working parameters, $A_{1:M-1} \in \{0, 1\}$ are degeneracy constants (for each $A_i = 0$, the Pareto optimal front dimensionality reduces by one). $h_{1:M}$ are shape functions, and determine the nature of the Pareto optimal front. [Huband *et al.* \(2006\)](#) created functions to assist the researcher in shaping the Pareto optimal front according to one of five shapes: linear, convex, concave, mixed convex/concave or

2.7 Standardised benchmark test suites

disconnected. Table 2.8 is presented to indicate how these shapes are created. $S_{1:M} > 0$ are scaling constants, and $\mathbf{t}^{1:p}$ are transition vectors, where “ \leftarrow ” indicates that each transition vector is created from another vector via transformation functions. Similarly to the shape functions, WFG problems can be transformed to introduce bias, deception, multi-modality and non-separability. Table 2.9 indicates how the transformation functions are executed. The domain of all $z_i \in \mathbf{z}$ is $[0, z_{i,max}]$, where all $z_{i,max} > 0$. The domain of $x_i \in \mathbf{x}$ is $[0, 1]$ (Huband *et al.*, 2006). There are numerous characteristics shared by all WFG problems. All are scalable and have known Pareto optimal sets. The number of position-related parameters k has to be a multiple of the number of underlying position parameters $M - 1$. The number of distance-related parameters l may be any positive integer, except for WFG2 and WFG3 where l has to be a multiple of two. Table 2.10 introduces the nine WFG problems and the construction of each, while Figure 2.26 displays their Pareto fronts.

2.7 Standardised benchmark test suites

Table 2.8: Shape functions. In all cases, $x_1, \dots, x_{M-1} \in [0, 1]$. A, α and β are constants (Huband *et al.*, 2006)

<p>Linear</p> $\text{linear}_1(x_1, \dots, x_{M-1}) = \prod_{i=1}^{M-1} x_i$ $\text{linear}_{m=2:M-1}(x_1, \dots, x_{M-1}) = \left(\prod_{i=1}^{M-m} x_i \right) (1 - x_{M-m+1})$ $\text{linear}_M(x_1, \dots, x_{M-1}) = 1 - x_1$ <p>When $h_{m=1:M} = \text{linear}_m$, the Pareto optimal front is a linear hyperplane, where $\sum_{m=1}^M h_m = 1$.</p>
<p>Convex</p> $\text{convex}_1(x_1, \dots, x_{M-1}) = \prod_{i=1}^{M-1} (1 - \cos(x_i \pi / 2))$ $\text{convex}_{m=2:M-1}(x_1, \dots, x_{M-1}) = \left(\prod_{i=1}^{M-m} (1 - \cos(x_i \pi / 2)) \right) (1 - \sin(x_{M-m+1} \pi / 2))$ $\text{convex}_M(x_1, \dots, x_{M-1}) = 1 - \sin(x_1 \pi / 2)$ <p>When $h_{m=1:M} = \text{convex}_m$, the Pareto optimal front is purely convex.</p>
<p>Concave</p> $\text{concave}_1(x_1, \dots, x_{M-1}) = \prod_{i=1}^{M-1} \sin(x_i \pi / 2)$ $\text{concave}_{m=2:M-1}(x_1, \dots, x_{M-1}) = \left(\prod_{i=1}^{M-m} \sin(x_i \pi / 2) \right) \cos(x_{M-m+1} \pi / 2)$ $\text{concave}_M(x_1, \dots, x_{M-1}) = \cos(x_1 \pi / 2)$ <p>When $h_{m=1:M} = \text{concave}_m$, the Pareto optimal front is purely concave, and a region of the hyper-sphere of radius one centred at the origin, where $\sum_{m=1}^M h_m^2 = 1$.</p>
<p>Mixed convex/concave ($\alpha > 0, A \in \{1, 2, \dots\}$)</p> $\text{mixed}_M(x_1, \dots, x_{M-1}) = \left(1 - x_1 - \frac{\cos(2A\pi x_1 + \pi/2)}{2A\pi} \right)^\alpha$ <p>Causes the Pareto optimal front to contain both convex and concave segments, the number of which is controlled by A. The overall shape is controlled by α: when $\alpha > 1$ or when $\alpha < 1$, the overall shape is convex or concave respectively. When $\alpha = 1$, the overall shape is linear.</p>
<p>Disconnected ($\alpha, \beta > 0, A \in \{1, 2, \dots\}$)</p> $\text{disc}_M(x_1, \dots, x_{M-1}) = 1 - (x_1)^\alpha \cos^2(A(x_1)^\beta \pi)$ <p>Causes the Pareto optimal front to have disconnected regions, the number of which is controlled by A. The overall shape is controlled by α (when $\alpha > 1$ or when $\alpha < 1$, the overall shape is concave or convex respectively, and when $\alpha = 1$, the overall shape is linear), and β influences the location of the disconnected regions (larger values push the location of disconnected regions towards larger values of x_1, and vice versa).</p>

2.7 Standardised benchmark test suites

Table 2.9: Transformation functions. Parameters y and $y_1, \dots, y_{|y|}$ have domain $[0, 1]$. A, B, C, α and β are constants (Huband *et al.*, 2006)

<p>Bias: Polynomial ($\alpha > 0, \alpha \neq 1$)</p> $\text{b_poly}(y, \alpha) = y^\alpha$ <p>When $\alpha > 1$ or when $\alpha < 1$, y is biased towards zero or towards one respectively.</p>
<p>Bias: Flat Region ($A, B, C \in [0, 1], B < C, B = 0 \Rightarrow A = 0 \wedge C \neq 1, C = 1 \Rightarrow A = 1 \wedge B \neq 0$)</p> $\text{b_flat}(y, A, B, C) = A + \min(0, \lfloor y - B \rfloor) \frac{A(B-y)}{B} - \min(0, \lfloor C - y \rfloor) \frac{(1-A)(y-C)}{1-C}$ <p>Values of y between B and C (the area of the flat region) are all mapped to the value A.</p>
<p>Bias: Parameter Dependent ($A \in (0, 1), 0 < B < C$)</p> $\text{b_param}(y, \mathbf{y}', A, B, C) = y^{B+(C-B)v(u(\mathbf{y}'))}$ $v(u(\mathbf{y}')) = A - (1 - 2u(\mathbf{y}')) \lfloor 0.5 - u(\mathbf{y}')) \rfloor + A \rfloor$ <p>A, B, C, and the secondary parameter vector \mathbf{y}' together determine the degree to which y is biased by being raised to an associated power: values of $u(\mathbf{y}') \in [0, 0.5]$ are mapped linearly onto $[B, B + (C - B)A]$, and values of $u(\mathbf{y}') \in [0.5, 1]$ are mapped linearly onto $[B + (C - B)A, C]$.</p>
<p>Shift: Linear ($A \in (0, 1)$)</p> $\text{s_linear}(y, A) = \frac{ y-A }{\lfloor A-y + A \rfloor}$ <p>A is the value for which y is mapped to zero.</p>
<p>Shift: Deceptive ($A \in (0, 1), 0 < B \ll 1, 0 < C \ll 1, A - B > 0, A + B < 1$)</p> $\text{s_decept}(y, A, B, C) = 1 + (y - A - B) \times \left(\frac{\lfloor y-A+B (1-C + \frac{A-B}{B}) \rfloor}{A-B} + \frac{\lfloor A+B-y (1-C + \frac{1-A-B}{B}) \rfloor}{1-A-B} + \frac{1}{B} \right)$ <p>A is the value at which y is mapped to zero, and the global minimum of the transformation. B is the ‘‘aperture’’ size of the well/basin leading to the global minimum at A, and C is the value of the deceptive minima (there are always two deceptive minima).</p>
<p>Shift: Multi-modal ($A \in \{1, 2, \dots\}, B \geq 0, (4A+2)\pi \geq 4B, C \in (0, 1)$)</p> $\text{s_multi}(y, A, B, C) = \frac{1 + \cos \left[(4A+2)\pi \left(0.5 - \frac{\lfloor y-C \rfloor}{\lfloor C-y + C \rfloor} \right) \right]}{B+2} + 4B \left(\frac{\lfloor y-C \rfloor}{\lfloor C-y + C \rfloor} \right)^2$ <p>A controls the number of minima, B controls the magnitude of the ‘‘hill sizes’’ of the multi-modality, and C is the value for which y is mapped to zero. When $B = 0$, $2A + 1$ values of y (one at C) are mapped to zero, and when $B \neq 0$, there are $2A$ local minima, and one global minimum at C. Larger values of A and smaller values of B create more difficult problems.</p>
<p>Reduction: Weighted Sum ($\mathbf{w} = \mathbf{y} , w_1, \dots, w_{ \mathbf{y} } > 0$)</p> $\text{r_sum}(\mathbf{y}, \mathbf{w}) = \left(\sum_{i=1}^{ \mathbf{y} } w_i y_i \right) / \sum_{i=1}^{ \mathbf{y} } w_i$ <p>By varying the constants of the weight vector \mathbf{w}, EAs can be forced to treat parameters differently.</p>
<p>Reduction: Non-separable ($A \in \{1, \dots, \mathbf{y} \}, \mathbf{y} \bmod A = 0$)</p> $\text{r_nonsep}(\mathbf{y}, A) = \frac{\sum_{j=1}^{ \mathbf{y} } (y_j + \sum_{k=0}^{A-2} \lfloor y_j - y_{1+(j+k) \bmod \mathbf{y} } \rfloor)}{\frac{ \mathbf{y} }{A} \lceil A/2 \rceil (1+2A-2\lceil A/2 \rceil)}$ <p>A controls the degree of non-separability (noting that $\text{r_nonsep}(\mathbf{y}, 1) = \text{r_sum}(\mathbf{y}, \{1, \dots, 1\})$).</p>

2.7 Standardised benchmark test suites

Table 2.10: The WFG test suite (Huband *et al.*, 2006)

Problem	Type	Setting
All	Constants	$S_{m=1:M} = 2m$ $D = 1$ $A_1 = 1$ $A_{2:M-1} = \begin{cases} 0, & \text{for WFG3} \\ 1, & \text{otherwise} \end{cases}$ The settings for $S_{1:M}$ ensures the Pareto optimal fronts have dissimilar tradeoff magnitudes, and the settings for $A_{1:M-1}$ ensures the Pareto optimal fronts are not degenerate, except in the case of WFG3, which has a one dimensional Pareto optimal front.
All	Domains	$z_{i=1:n, \max} = 2i$ The working parameters have domains of dissimilar magnitude.
WFG1	Shape t^1 t^2 t^3 t^4	$h_{m=1:M-1} = \text{convex}_m$ $h_M = \text{mixed}_M$ (with $\alpha = 1$ and $A = 5$) $t_{i=1:k}^1 = y_i$ $t_{i=k+1:n}^1 = \text{s.linear}(y_i, 0.35)$ $t_{i=1:k}^2 = y_i$ $t_{i=k+1:n}^2 = \text{b.flat}(y_i, 0.8, 0.75, 0.85)$ $t_{i=1:n}^3 = \text{b.poly}(y_i, 0.02)$ $t_{i=1:M-1}^4 = \text{r.sum}(\{y_{(i-1)k/(M-1)+1}, \dots, y_{ik/(M-1)}\}, \{2((i-1)k/(M-1)+1), \dots, 2ik/(M-1)\})$ $t_M^4 = \text{r.sum}(\{y_{k+1}, \dots, y_n\}, \{2(k+1), \dots, 2n\})$
WFG2	Shape t^1 t^2 t^3	$h_{m=1:M-1} = \text{convex}_m$ $h_M = \text{disc}_M$ (with $\alpha = \beta = 1$ and $A = 5$) As t^1 from WFG1. (Linear shift.) $t_{i=1:k}^2 = y_i$ $t_{i=k+1:l/2}^2 = \text{r.nonsep}(\{y_{k+2(i-k)-1}, y_{k+2(i-k)}\}, 2)$ $t_{i=1:M-1}^3 = \text{r.sum}(\{y_{(i-1)k/(M-1)+1}, \dots, y_{ik/(M-1)}\}, \{1, \dots, 1\})$ $t_M^3 = \text{r.sum}(\{y_{k+1}, \dots, y_{k+l/2}\}, \{1, \dots, 1\})$
WFG3	Shape $t^{1:3}$	$h_{m=1:M} = \text{linear}_m$ (degenerate) As $t^{1:3}$ from WFG2. (Linear shift, non-separable reduction, and weighted sum reduction.)
WFG4	Shape t^1 t^2	$h_{m=1:M} = \text{concave}_m$ $t_{i=1:n}^1 = \text{s.multi}(y_i, 30, 10, 0.35)$ $t_{i=1:M-1}^2 = \text{r.sum}(\{y_{(i-1)k/(M-1)+1}, \dots, y_{ik/(M-1)}\}, \{1, \dots, 1\})$ $t_M^2 = \text{r.sum}(\{y_{k+1}, \dots, y_n\}, \{1, \dots, 1\})$
WFG5	Shape t^1 t^2	$h_{m=1:M} = \text{concave}_m$ $t_{i=1:n}^1 = \text{s.decept}(y_i, 0.35, 0.001, 0.05)$ As t^2 from WFG4. (Weighted sum reduction.)
WFG6	Shape t^1 t^2	$h_{m=1:M} = \text{concave}_m$ As t^1 from WFG1. (Linear shift.) $t_{i=1:M-1}^2 = \text{r.nonsep}(\{y_{(i-1)k/(M-1)+1}, \dots, y_{ik/(M-1)}\}, k/(M-1))$ $t_M^2 = \text{r.nonsep}(\{y_{k+1}, \dots, y_n\}, l)$
WFG7	Shape t^1 t^2 t^3	$h_{m=1:M} = \text{concave}_m$ $t_{i=1:k}^1 = \text{b.param}(y_i, \text{r.sum}(\{y_{i+1}, \dots, y_n\}, \{1, \dots, 1\}), \frac{0.98}{49.98}, 0.02, 50)$ $t_{i=k+1:n}^1 = y_i$ As t^1 from WFG1. (Linear shift.) As t^2 from WFG4. (Weighted sum reduction.)
WFG8	Shape t^1 t^2 t^3	$h_{m=1:M} = \text{concave}_m$ $t_{i=1:k}^1 = y_i$ $t_{i=k+1:n}^1 = \text{b.param}(y_i, \text{r.sum}(\{y_1, \dots, y_{i-1}\}, \{1, \dots, 1\}), \frac{0.98}{49.98}, 0.02, 50)$ As t^1 from WFG1. (Linear shift.) As t^2 from WFG4. (Weighted sum reduction.)
WFG9	Shape t^1 t^2 t^3	$h_{m=1:M} = \text{concave}_m$ $t_{i=1:n-1}^1 = \text{b.param}(y_i, \text{r.sum}(\{y_{i+1}, \dots, y_n\}, \{1, \dots, 1\}), \frac{0.98}{49.98}, 0.02, 50)$ $t_n^1 = y_n$ $t_{i=1:k}^2 = \text{s.decept}(y_i, 0.35, 0.001, 0.05)$ $t_{i=k+1:n}^2 = \text{s.multi}(y_i, 30, 95, 0.35)$ As t^2 from WFG6. (Non-separable reduction.)

2.7 Standardised benchmark test suites

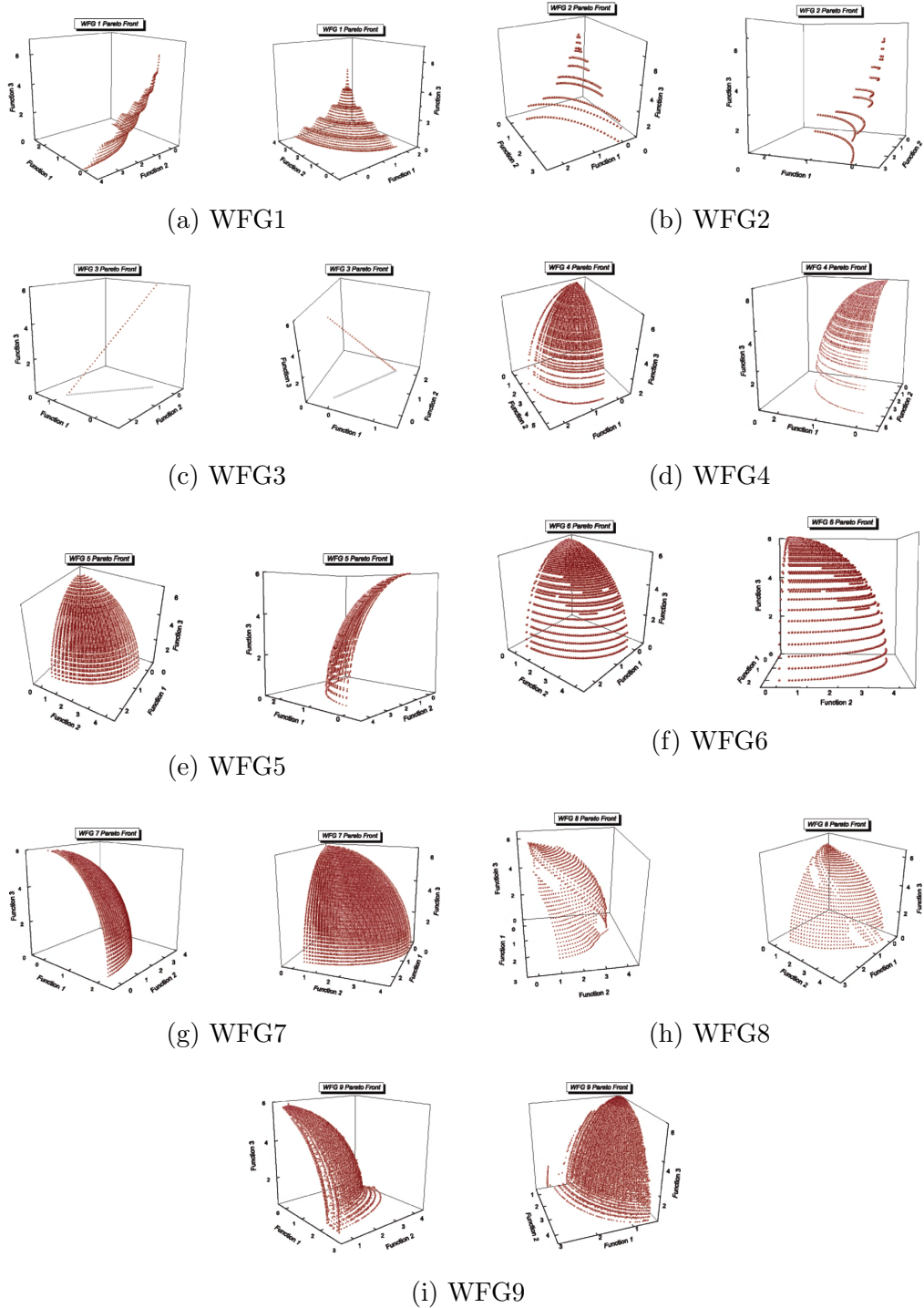


Figure 2.26: Pareto fronts of the nine WFG problems (Huband *et al.*, 2006)

2.7 Standardised benchmark test suites

General characteristics of the WFG problems are summarised and presented in Table 2.11.

Table 2.11: Characteristics of the WFG test suite (Huband *et al.*, 2006)

Problem	Obj.	Separability	Modality	Bias	Geometry
WFG1	$f_{1:M}$	S	U	polynomial, flat	convex, mixed
WFG2	$f_{1:M-1}$	NS	U	-	convex, disconnected
	f_M	NS	M		
WFG3	$f_{1:M}$	NS	U	-	linear, degenerate
WFG4	$f_{1:M}$	S	M	-	concave
WFG5	$f_{1:M}$	S	D	-	concave
WFG6	$f_{1:M}$	NS	U	-	concave
WFG7	$f_{1:M}$	S	U	parameter dependent	concave
WFG8	$f_{1:M}$	NS	U	parameter dependent	concave
WFG9	$f_{1:M}$	NS	M	parameter dependent	concave

2.7.3 The ZDT Benchmark MOPs

The ZDT (Zitzler-Deb-Thiele) test suite (Zitzler *et al.*, 2000) contains six problems, which are all constructed in a similar manner and contain three functions; f_1 , g and h . They are structured according to

$$\begin{aligned}
 \text{Minimise } F(\mathbf{x}) &= (f_1, f_2), \\
 \text{s.t. } f_2(\mathbf{x}) &= g(x_2, \dots, x_m)h(f_1(x_1), g(x_2, \dots, x_m)), \\
 \text{where } : \mathbf{x} &= (x_1, \dots, x_m).
 \end{aligned}$$

f_1 is a function of the first variable only, whilst the remaining $m - 1$ variables are described by function g . The function values of f_1 and g are the parameters of h . The test problems differ in these functions and the number of variables m , as well as their values. A description of the ZDT problems and figures of their Pareto fronts are provided next. The formulae and variable values of the problems are provided in Table 2.12.

ZDT1 has a convex Pareto front \mathcal{PF}_{True} which is formed with $g(\mathbf{x}) = 1$ and is shown in Figure 2.27a.

ZDT2 has a non-convex Pareto front \mathcal{PF}_{True} which is formed with $g(\mathbf{x}) = 1$. It is shown in Figure 2.27b.

2.7 Standardised benchmark test suites

Table 2.12: Standard ZDT test problems

Function	Definition	Constraints
ZDT1 (Min)	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}, g) = g(\mathbf{x}) \cdot (1 - \sqrt{f_1/g(\mathbf{x})})$ $g(\mathbf{x}) = 1 + \frac{9}{n-1} \cdot \sum_{i=2}^n x_i$	$0 \leq x_i \leq 1, n=30$
ZDT2 (Min)	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}, g) = g(\mathbf{x}) \cdot (1 - (f_1/g(\mathbf{x}))^2)$ $g(\mathbf{x}) = 1 + \frac{9}{n-1} \cdot \sum_{i=2}^n x_i$	$0 \leq x_i \leq 1, n=30$
ZDT3 (Min)	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}, g) = g(\mathbf{x}) \cdot (1 - \sqrt{f_1/g(\mathbf{x})}) - f_1/g(\mathbf{x}) \cdot \sin(10\pi f_1)$ $g(\mathbf{x}) = 1 + \frac{9}{n-1} \cdot \sum_{i=2}^n x_i$	$0 \leq x_i \leq 1, n=30$
ZDT4 (Min)	$f_1(\mathbf{x}) = x_1$ $f_2(\mathbf{x}, g) = g(\mathbf{x}) \cdot (1 - \sqrt{f_1/g(\mathbf{x})})$ $g(\mathbf{x}) = 1 + 10 \cdot (n-1) + \sum_{i=2}^n (x_i^2 - 10 \cos(4\pi x_i))$	$0 \leq x_i \leq 1, n=30$
ZDT5 (Min)	$f_1(\mathbf{x}) = 1 + u(x_1)$ $f_2(\mathbf{x}, g) = \frac{g(\mathbf{x})}{f_1(\mathbf{x})}$ $g(\mathbf{x}) = \sum_{i=2}^{11} v(u(x_i))$ $v(u(x_i)) = \begin{cases} 2 + u(x_i), & \text{if } u(x_i) < 5; \\ 1, & \text{if } u(x_i) = 5 \end{cases}$	$0 \leq x_i \leq 1, n=11$ $u(x_1) =$ number of ones in bit vector x_1 $x_1 \in \{0, 1\}^{30}$, $x_2, \dots, x_n \in \{0, 1\}^5$
ZDT6 (Min)	$f_1(\mathbf{x}) = 1 - \exp(-4x_1) \cdot \sin^6(6\pi x_1)$ $f_2(\mathbf{x}, g) = g(\mathbf{x}) \cdot (1 - (\frac{f_1}{g(\mathbf{x})})^2)$ $g(\mathbf{x}) = 1 + 9 \cdot [(\frac{\sum_{i=2}^n x_i}{9})^{0.25}]$	$0 \leq x_i \leq 1, n=30$

ZDT3 has a Pareto front \mathcal{PF}_{True} with multiple disconnected convex sections which are caused by the inclusion of a sine function. The Pareto front \mathcal{PF}_{True} is achieved with $g(\mathbf{x}) = 1$ and it is shown in Figure 2.27c.

ZDT4 is a difficult problem containing 21^9 local Pareto fronts. This is ideal for testing a multi-objective optimisation algorithm's ability to handle multi-frontality. It has a non-convex Pareto front \mathcal{PF}_{True} which is formed with $g(\mathbf{x}) = 1$. It is shown in Figure 2.27d.

ZDT5 has a nuance which differentiates it from the other ZDT problems. As seen in Table 2.12, part of the $g(\mathbf{x})$ function contains x_i which is a binary string, with $u(X_i)$ giving the number of ones in the bit vector x_i . The convex Pareto front \mathcal{PF}_{True} is achieved with $g(\mathbf{x}) = 10$ and it is shown in Figure 2.27e. Since the MOPSO cannot accommodate binary problems, ZDT5 is excluded from this

2.7 Standardised benchmark test suites

study.

ZDT6 has a non-convex Pareto front \mathcal{PF}_{True} which is formed with $g(\mathbf{x}) = 1$ and is shown in Figure 2.27f. The problem is difficult because the true solutions are non-uniformly distributed along the Pareto front.

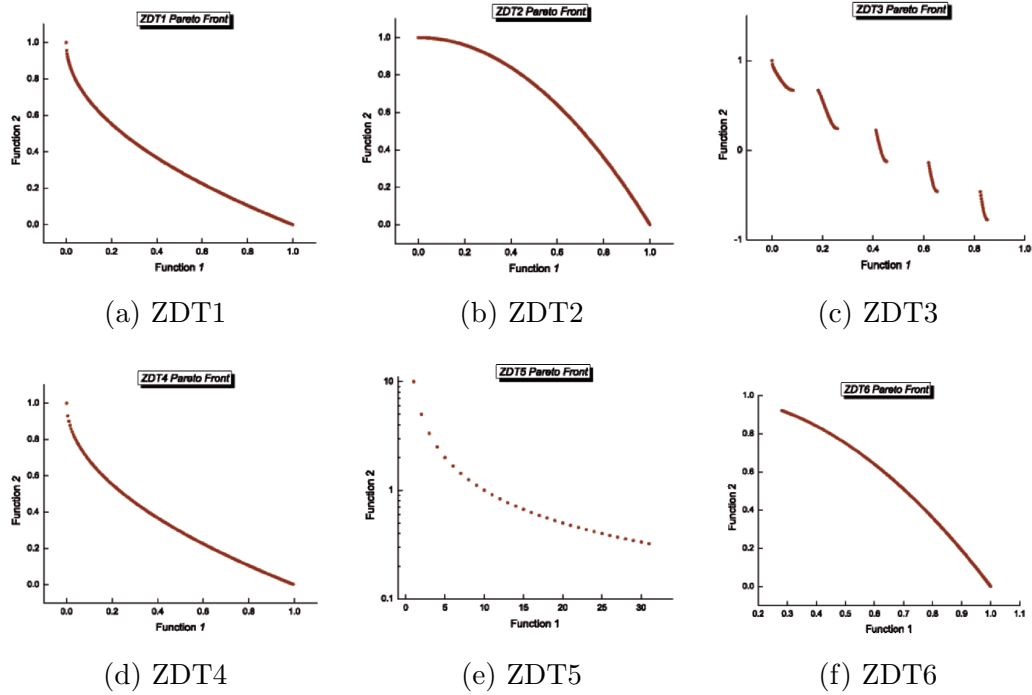


Figure 2.27: Pareto fronts of the six ZDT problems (Coello *et al.*, 2007)

General characteristics of the ZDT problems are summarised and presented in Table 2.13.

2.8 Summary: Chapter 2Table 2.13: Characteristics of the ZDT test suite (Huband *et al.*, 2006)

Problem	Obj.	Separability	Modality	Bias	Geometry
ZDT1	f_1	S	U	-	convex
	f_2	S	U		
ZDT2	f_1	S	U	-	concave
	f_2	S	U		
ZDT3	f_1	S	U	-	disconnected
	f_2	S	M		
ZDT4	f_1	S	U	-	convex
	f_2	S	M		
ZDT6	f_1	S	U	+	concave
	f_2	S	M		

2.8 Summary: Chapter 2

An overview of the literature pertaining to the field of multi-objective optimisation was presented in this chapter. The intention was to provide the reader with enough insight to understand the importance of non-dominated sorting as a subfunction of multi-objective optimisation, and how its own optimisation may improve the overall runtime of a multi-objective optimisation algorithm that uses it. Also discussed were a few of the most recognised evolutionary algorithms and the strategies behind their operation. Not only do these algorithms necessitate the entire existence of non-dominated sorting, but they are ultimately used as the medium through which the performance of a non-dominated sorting algorithm is assessed. Explored next were some of the latest non-dominated sorting algorithms, providing important ideas and techniques to explore during the creation of a new non-dominated sorting algorithm in the following chapter. A common theme centred around the creation of non-dominated sorting algorithms is to reduce the number of solution comparisons as much as possible. This led to the question around which the algorithm developed in the next chapter was created; are solution comparisons necessary at all? The chapter's final section focused on three standardised benchmark test suites, all of which were created by some of the most recognised researchers in the field. Not only were the problems and

2.8 Summary: Chapter 2

their formulations introduced, but a summary of their characteristics was also provided in order to give more detailed feedback after tests are conducted. This allows for specific reporting as to where and when certain multi-objective evolutionary and non-dominated sorting algorithms should be employed. Since the testing of evolutionary algorithms depends on stochastic results, it is important to have a means through which one can determine if there is a statistically significant difference between two results. Therefore, an overview of the Wilcoxon rank-sum test was provided, which will be used to calculate the margins which state whether one algorithm may truly claim a superior result over another. In the next chapter, the novel non-dominated sorting algorithm Shadow Sort, on which this research is based, will be introduced.

Chapter 3

Shadow Sort: a novel non-dominated sorting algorithm

In the previous chapter, the field of multi-objective optimisation was introduced, along with a number of multi-objective evolutionary algorithms which may be used to perform such optimisation. Focus was put on non-dominated sorting, a subfunction required in the majority of multi-objective evolutionary algorithms. Here, existing algorithms and techniques were explored, providing insights to previous ways of thinking during their conception. Based on these building blocks, a novel non-dominated sorting algorithm is presented in this chapter.

3.1 Proposed Method

Proposed is a new algorithm named *Shadow Sort* (SS) for non-dominated sorting that does not require dominance checking between solutions in the population. Rather, the algorithm takes advantage of the relative positions of solutions after being sorted according to the population's various objectives. This concept is based on the following understanding: if a population is ordered by a particular objective from best to worst, then a solution b which finds itself outranked by solution a must be considered worse than a in that particular objective. If this idea is extended to the other objectives, then a solution b which is outranked by solution a on all sorted objectives is dominated by a .

3.2 Shadow Sort

The Shadow Sort algorithm contains three cases, only one of which is called to sort a population. The case that is used depends on the number of objectives in the population to be sorted. The first case is employed on populations with only two objectives, the second case on populations with three objectives to 14 objectives, and the third case on populations with 15 or more objectives. A general note applying to all non-dominated sorting performed by Shadow Sort is that it assumes a minimisation problem. Therefore, all references to sorting refer to sorting in ascending order. Should the optimisation actually strive to maximise the objectives, the input population may simply be multiplied by negative one, which allows the algorithm to remain unchanged as it then becomes a minimisation problem.

3.2.1 Case I: Two objectives

In this case, Shadow Sort performs a procedure similar to that of ENS, however takes advantage of the bi-dimensional setting to reduce additional unnecessary comparisons. The population is ordered according to the first objective, and a flag variable is initialised with a large value. If solutions with identical values for objective one exist, ties are broken with objective two. A solution which has the same values as a previous solution for objectives one and two is a duplicate point and is removed from the population as its presence in a multi-objective algorithm will not improve the solution space. The first solution of a sorted list, solution a , exhibits the best value for the first objective, and is necessarily the first non-dominated solution. The flag variable is set to equal solution a 's second objective value. The algorithm then assesses the second objective value for each remaining solutions in the population as sorted by the first objective. If a solution's second objective value is lower than that of the flag variable, that solution is included as a non-dominated solution and the flag variable is updated to equal that solution's second objective value. Algorithm 4 provides pseudo-code for Shadow Sort Case I, and what follows is an implementation explanation.

3.2 Shadow Sort

Algorithm 4 Shadow Sort: Case I

```

1: procedure SHADOW SORT( $Pop$ ) ▷ The population
2:    $N \leftarrow size(Pop, 1)$  ▷  $N$  is set to the number of solutions in  $Pop$ 
3:    $Pop \leftarrow$  Sort the population based on first objective
4:    $Flag \leftarrow inf$ 
5:   for  $p = 1 : N$  do
6:     if  $Pop(p, 2) < Flag$  then
7:       Add  $Pop(p, :)$  to non-dominated set
8:        $Flag \leftarrow Pop(p, 2)$ 
9:     end if
10:  end for
11: end procedure
    
```

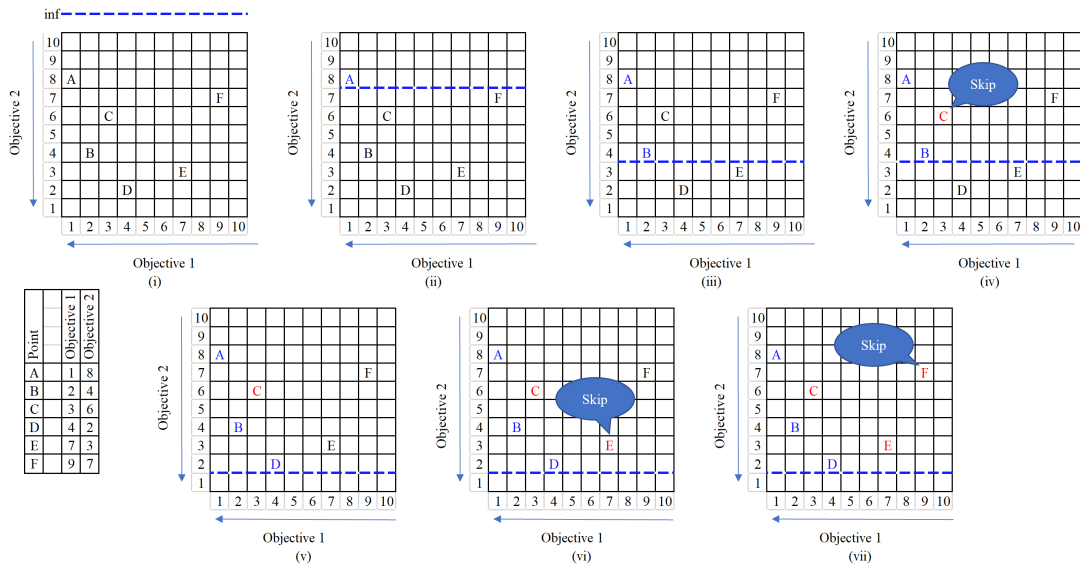


Figure 3.1: Shadow Sort Case I example with one front

Figure 3.1 is provided as a visual aid to explain Case I. In the example, six points (A to F) are provided, and their objective one and two values indicated in the table on the bottom left of the figure. For ease of explanation, the objective values are integers, which allows for simple representation on the diagram grids. Line 3 of Algorithm 4 indicates that the population should be sorted along column one¹. Solutions A to F are presented in a sorted order, and their placement on the

¹sorting is in ascending order, as all non-dominated sorting assumes minimisation

3.2 Shadow Sort

grid is provided in Figure 3.1(i). The variable *Flag* is represented by the dashed blue line and, according to Line 4 of Algorithm 4, initially set to infinity. The algorithm then cycles through all the solutions in the population for $p \in [1, N]$. The first step of the loop (line 6 of Algorithm 4) is to compare the second objective value of solution p to the value of *Flag*. If solution p 's second objective value is larger than the value of *Flag*, then solution p is dominated by a preceding solution, and is skipped. However, if solution p 's second objective value is less than the value of *Flag*, then solution p is added to the non-dominated set, and *Flag* is set to equal p 's second objective value. The loop is then repeated for the remaining solutions in N . In Figure 3.1, the first solution is A , and its second objective value is eight. Since eight is less than infinity, solution A is added to the non-dominated set, and the value of *Flag* is set to eight (dashed line in Figure 3.1(ii)). Solution B has a second objective value equal to four, which is less than *Flag*'s value of eight. Therefore, solution B is also added to the non-dominated set, and *Flag* is set to equal four (dashed line in Figure 3.1(iii)). Solution C has a second objective value of six, which is larger than *Flag*'s four. Therefore, solution C is not included in the non-dominated set, and the value of *Flag* remains four. The algorithm continues this procedure for the remaining solutions in the population, and by the end, the entire Rank 1 non-dominated set is provided.

Should the researcher or multi-objective optimisation algorithm require that additional fronts are provided, the algorithm may be slightly modified that *Flag* becomes a $1 \times R$ vector (where R is the number of ranks requested by the researcher or multi-objective optimisation algorithm). In this manner, should a solution's second objective value not be smaller than *Flag*(1), it would be checked against *Flag*(2) (and further as required). In Figure 3.2, *Flag* is used in the same manner, but if a solution p has an inferior second objective compared to *Flag*(1), it is then compared to *Flag*(2). By the end of the algorithm, solutions that were superior to *Flag*(1) are recorded as Rank 1, solutions that were superior to *Flag*(2) are recorded as Rank 2, *etc.*

3.2 Shadow Sort

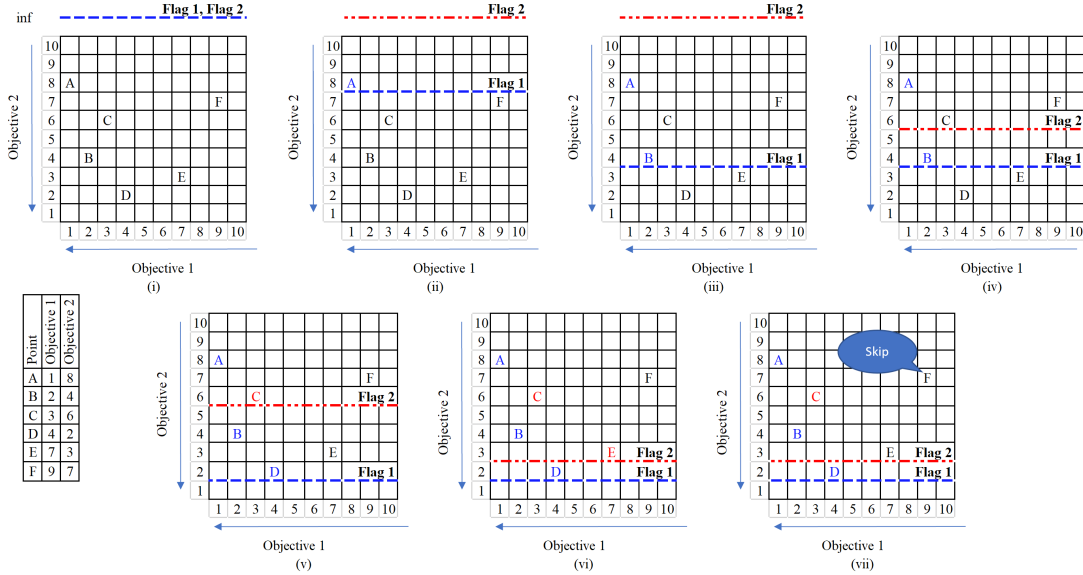


Figure 3.2: Shadow Sort Case I example with 2 fronts

Matlab makes use of the Quick Sort algorithm (Hoare, 1961) which has complexity $\mathcal{O}(N \log N)$. The algorithm performs R loops through solutions 1 to N , where R is the number of ranks requested by the researcher or multi-objective optimisation algorithm. Therefore, the complexity of Shadow Sort Case I is $R \times N + N \log N$, therefore $\mathcal{O}(N \log N)$.

3.2.2 Case II: Three to 14 objectives

As previously mentioned, Shadow Sort is so named as it eliminates solutions which fall in the “shadow” non-dominated solutions. Specifically, in Shadow Sort Case II, the algorithm processes the population by ordering it according to its various objectives. Once the algorithm has determined the rank of each solution along each objective, it uses this information to systematically assess a given solution, and then flag all solutions which it dominates. The actual assessment starts by checking the flagging value of the solution. If the solutions flag is true, it means that the solution is dominated by a preceding solution, and is therefore excluded from any further assessment. If the solutions flag is false, then the

3.2 Shadow Sort

solution is non-dominated, and the algorithm uses the solutions rank along all its objectives to flag all solutions which it dominates.

In order to perform this operation, it is necessary to ensure that solutions are assessed in the correct order. This is achieved by addressing solutions as sorted along its first objective.

Algorithm 5 provides pseudocode for Shadow Sort Case II. This is followed by a description of the pseudocode, along with a toy example to further illustrate key points. The rows are sorted according to the problem's first objective from best to worst (smallest to largest) and saved as a variable called *Key*. The original positions of the solutions are also recorded and saved (line 2). For the sake of reducing unnecessary computations, duplicate solutions are removed from the population (they are reinserted at the end of the algorithm so that the function calling the non-dominated sorting gets the same number of solutions back). Line 3 is used to evaluate the number of solutions to be sorted (N), as well as the number of objectives in the population (M). A zero-vector called *FrontNo* is created in line 4 to keep record of solution N 's rank number once it is calculated. Lines 5 to 10 prepare *Key* to be used as a position indicator for Shadow Sort as described in the beginning of this section. This is achieved by replacing the first column (which has already been sorted by line 2) with integers 1 to N . These numbers not only indicate a solution's rank in objective one, but will also be used as the solution's identity for the remainder of the algorithm. Lines 6 to 9 contain a *for*-loop over objectives $i \in [2, M]$, each time sorting the rows of *Key* according to objective i , and then replacing the values of column i with integers 1 to N . Line 10 then returns *Key* to its original sequence by sorting the rows by objective one, and line 11 sets the rank counter k to zero. The algorithm's main loop (lines 12 to 29) is cycled until the stopping criterium is met (*i.e.* when all solutions have been ranked). Before evaluating solutions, the rank counter k is first incremented by one, and the binary-vector *Flag* is set to false. Next is a *for*-loop between lines 16 and 25 for $p \in [1, N]$. First, *Flag*(p) is checked; if it is *true*, then solution p has already been dominated and is not to undergo further processing. If *Flag*(p) is *false*, then solution p is assigned a rank value of k . Since a solution was just ranked, another variable *count* counting the number of solutions ranked is incremented by one (line 22). Line 23 involves updating

3.2 Shadow Sort

a *binary matrix* with a logic operation called *Bitmat*. For each column, the statement instructs *Bitmat* to indicate which elements in *Key* are larger than the value in *Key* row p . This concept is illustrated in Figure 3.3. The left frame provides an example of *Key*, seen here with 10 solutions and four objectives. Assume that the highlighted solution $p = 6$ is being assessed, with *Key* values $[6, 5, 3, 2]$. Line 23 of Algorithm 5 calls for all entries in *Key* larger than $Key(p)$ to be marked as *true* in each column. Therefore, in the left frame of Figure 3.3 in column i , the bold values $[7, 8, 9, 10]$ in rows $[G, H, I, J]$ are given values of *one* (*true*) in column i of *Bitmat*. Likewise for the bold values $[9, 8, 7, 6, 10]$ in rows $[D, E, G, H, I]$ of column (ii) , $[10, 8, 9, 6, 5, 7, 4]$ in rows $[A, B, C, E, G, H, I]$ of column (iii) and $[9, 5, 6, 8, 3, 7, 10, 4]$ in rows $[A, B, C, D, E, G, H, J]$ of column (iV) . Line 24 of Algorithm 5 then performs a dual calculation as explained in the centre and right frames of Figure 3.3. The first step is to take the conjunction of each row of *Bitmat* (also known as an *AND* operation). As depicted in the centre frame, *Bitmat* becomes a binary column vector, and only the rows having all their objective columns equal *one* are ultimately given a value of *one*. The second step is to take the disjunction (also known as an *OR* operation) of the resulting *Bitmat* column vector of step one, and the already existing column vector *Flag*. In the right frame of Figure 3.3, it can be seen that *Bitmat* has *true* entries in rows 7 and 8, but *Flag* has only zeros. But since an *OR* operation only requires one of the arguments to be *true*, *Flag* is updated with the *true* rows of *Bitmat*. The next loop starting on line 16 then assesses the next solution $p = 7$. Line 19 determines that $Flag(7)$ is *true*, and therefore skips the remaining code in the loop before continuing to solution $p = 8$. Line 26 is used if the decision-maker or multi-objective optimisation algorithm requested only a specific number of ranked solutions $nSort$. If the counter variable is more than or equal to $nSort$, then the algorithm is terminated. Line 30 calculates the largest rank number of the ranked solutions before line 31 returns the population to the calling function in its original order.

As mentioned in Subsection 3.2.1, *Matlab* makes use of the Quick Sort algorithm for sorting which has complexity $\mathcal{O}(N \log N)$. Therefore, the overhead to initialise Shadow Sort exhibits complexity $\mathcal{O}(MN \log N)$.

3.2 Shadow Sort

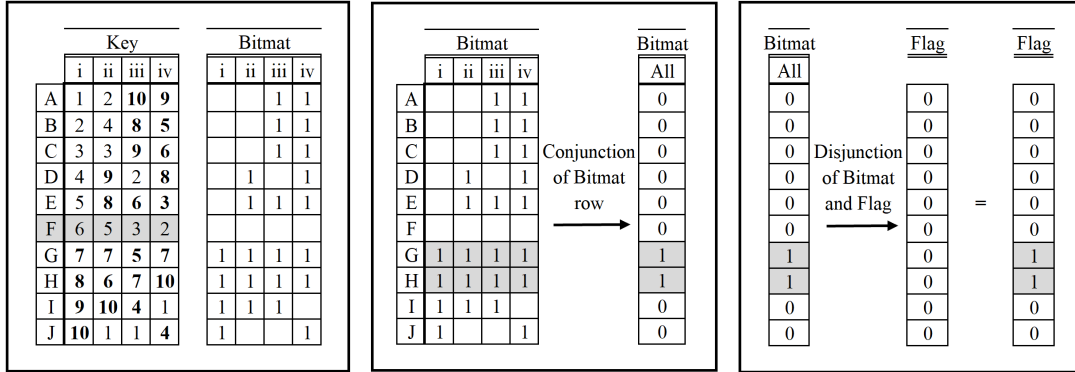
Algorithm 5 Shadow Sort: Case II

```

1: procedure SHADOW SORT(Pop, nSort) ▷ Population & No. of solutions to
   rank
2:   [Key,  $\sim$ , Loc] = unique(pop, 'rows');
3:   [N, M] = size(pop);
4:   FrontNo = zeros(N, 1);           ▷ Zero column vector for rank number
5:   Key(:, 1) = [1 : N]';           ▷ Replace actual values of Obj 1 with ID vector
6:   for i = 2 : M do                 ▷ For the remaining objectives
7:     Key = sortrows(Key, i);
8:     Key(:, i) = [1 : N]';       ▷ Replace actual values of Obj i with ID vector
9:   end for
10:  Key = sortrows(Key, 1);           ▷ Sort 'Key' according to Obj 1
11:  k = 0                               ▷ Rank counter
12:  while min(FrontNo) == 0 do
13:    k = k + 1                         ▷ Increment k
14:    Flag = false(N, 1);
15:    KeyM = Key(FrontNo == 0, 1)     ▷ Only assess unranked solutions
16:    for i = 1 : size(KeyM, 1) do     ▷ For all unranked solutions
17:      p = KeyM(i);
18:      if Flag(p) == true then     ▷ If point i's flag is already 1, skip.
19:        continue
20:      end if
21:      FrontNo(p) = k;
22:      count = count + 1;
23:      Bitmat = Key > Key(p, :);
24:      Flag = bitor(Flag, all(Bitmat, 2))
25:    end for
26:    if count >= nSort then
27:      break
28:    end if
29:  end while
30:  MaxFNo = max(FrontNo)             ▷ Find highest calculated rank
31:  FrontNo = FrontNo(Loc)           ▷ Return results in original order
32: end procedure

```

3.2 Shadow Sort

Figure 3.3: Illustrative example of the Case II *Bitmat* operation

3.2.3 Case III: 15 or more objectives

The average sorting time for Shadow Sort using Case II increases as the number of objectives increase. This is due to two reasons:

- The number of row-sorting required to create the matrix *Key* equals the number of objectives in the population.
- The number of objectives to assess in the *Bitmat* operation per solution also equals the number of objectives.

Specifically, since *Matlab* uses *QuickSort*, if there are M objectives, the computational complexity to create the matrix *Key* as described in Case II is $\mathcal{O}(MN \log N)$, and this is before any non-dominated sorting has taken place. Furthermore, as described by Shultz & Fahlman (2017), the addition of extra dimensions results in an exponential increase in the search domain, a phenomenon known as the curse of dimensionality (Bellman, 1958). A modification to Shadow Sort was considered where, instead of performing non-dominated sorting with *Bitmat* as in Case II, traditional dominance comparisons could be made between a specific reduced set of solutions. In Case III, all objectives of solution p are examined, and the objective that yields the worst component of solution p is selected. All solutions ranked worse than solution p in this selected objective are then shortlisted for traditional domination comparison. A final check is made to further reduce this shortlist, which eliminates all solutions with a better ranking than solution p in

3.2 Shadow Sort

the first objective (easily identifiable as a solution's identity is its rank in the first objective). A domination check is then performed between solution p and those solutions remaining in the shortlist. As the number of objectives increases, the chance of a solution performing poorly on at least one objective increases, thereby reducing the length of the shortlist of solutions needing to undergo dominance comparisons, ultimately reducing the algorithm runtime. Consequently, the non-dominated sorting time of Shadow Sort using Case III is slower than Case II in low-objective populations, but a cross-over point exists where Case III becomes the more efficient non-dominated sorting technique. Through experimental studies using randomly generated populations over a varying number of objectives, it was found that it is more efficient to perform non-dominated sorting with Case III on populations exhibiting 15 or more objectives.

Algorithm 6 provides the *Matlab* pseudocode for the above-described Case III. The first few lines are much the same as Algorithm 5, except for the initialisation of an additional $N \times M$ matrix called *Standing*. This new matrix is similar to *Key* in that it contains the integers $1 : N$ in each column, however the interpretation of a specific cell is slightly different. Figure 3.4 illustrates the difference between *Key* and *Standing*. *Key* can be interpreted as follows: if the solutions $N \in [A, J]$ of the original population had to be sorted along any objective $M \in [i, iv]$, then the rank of solution N in objective M can be found in cell (N, M) . The interpretation of *Standing* is as follows: if the solutions $N \in [A, J]$ of the original population had to be sorted along any objective $M \in [i, iv]$, then the solution of rank N in objective M can be found in cell (N, M) . Thus, looking at solution A in *Key*, we see that its rank in objectives i, ii, iii and iv are 1, 2, 10 and 9, respectively. It is shown in *Standing* that if the population had to be sorted along objectives i, ii, iii and iv , the solutions with rank 1 are A, J, J and I , respectively.

Line 27 of Algorithm 6 is used to create and clear the contents of another new vector called *ShortList*. As mentioned above, the strategy of this case is to determine a reduced set of solutions against which a classic domination check may be performed. In line 26, q and o are set to equal the worst performing objective of solution p , and which objective this is respectively. The vector *ShortList* is then set to equal the solutions in *Key* column o below row q . This comprises the shortest list of solutions that could potentially be dominated by solution p .

3.2 Shadow Sort

	Key				Standing			
	i	ii	iii	iv	i	ii	iii	iv
A	1	2	10	9	1	10	10	9
B	2	4	8	5	2	1	4	6
C	3	3	9	6	3	3	6	5
D	4	9	2	8	4	2	9	10
E	5	8	6	3	5	6	7	2
F	6	5	3	2	6	8	5	3
G	7	7	5	7	7	7	8	7
H	8	6	7	10	8	5	2	4
I	9	10	4	1	9	4	3	1
J	10	1	1	4	10	9	1	8

Figure 3.4: An example of matrices *Key* and *Standing*

Line 28 performs the final grooming of this list to ensure it is as short as possible, namely by removing all solutions with a better rank than solution p in Objective 1. Line 29 checks whether the vector *ShortList* contains any remaining solutions. If not, then solution p does not dominate any other solutions in the population, and the *for*-loop on line 19 increments to the next solution. If, however, there are solutions remaining in *ShortList*, then they are each assessed against solution p by means of classic dominance checking as described by Algorithm 7. The solutions dominated by solution p are flagged in line 32 to ensure they do not undergo expensive assessment for the current rank number. As in Case II, a check exists after each *while*-loop to check whether enough solutions have been ranked (based on the decision-maker's or multi-objective evolutionary algorithm's requirements). Finally, line 41 provides the maximum rank number of the solutions ranked, and line 421 returns the rank of the input population in the original order.

3.2 Shadow Sort

Algorithm 6 Shadow Sort: Case III

```

1: procedure SHADOW SORT( $Pop, nSort$ )  $\triangleright$  Population, No. solutions to rank
2:    $[Key, \sim, Loc] = unique(pop, 'rows')$ 
3:    $[N, M] = size(pop)$ 
4:    $FrontNo = zeros(N, 1)$   $\triangleright$  Zero column vector for Rank number
5:    $Key(:, 1) = [1 : N]'$ 
6:    $Standing = [[1 : N]', zeros(N, M - 1)]$ 
7:   for  $i = 2 : M$  do  $\triangleright$  For the remaining objectives
8:      $Key = sortrows(Key, i)$ 
9:      $Key(:, i) = [1 : N]'$ 
10:     $Standing(:, i) = Key(:, 1)$ 
11:  end for
12:   $Key = sortrows(Key, 1)$ 
13:   $Standing = sortrows(Standing, 1)$   $\triangleright$  Sort  $Standing$  by Obj 1
14:   $k = 0$   $\triangleright$  Rank counter
15:  while  $min(FrontNo) == 0$  do
16:     $k = k + 1$   $\triangleright$  Increment  $k$ 
17:     $Flag = false(N, 1)$ 
18:     $KeyM = Key(FrontNo == 0, 1)$   $\triangleright$  Only assess unranked solutions
19:    for  $i = 1 : size(KeyM, 1)$  do
20:       $p = KeyM(i)$ 
21:      if  $Flag(p) == true$  then  $\triangleright$  If point  $i$ 's flag is already 1, skip
22:        continue
23:      end if
24:       $FrontNo(p) = k$ 
25:       $count = count + 1$ 
26:       $[q, o] = max(Key(p, :))$ 
27:       $ShortList = Standing(q + 1 : end, o)$ 
28:       $ShortList(ShortList < p) = []$ 
29:      if  $\sim isempty(ShortList)$  then
30:        for  $x = 1 : size(ShortList)$  do
31:          if  $DomCheck(Key(p, :), Key(ShortList(x), :))$  then
32:             $Flag(ShortList(x)) = Flag(ShortList(x)) + 1$ 
33:          end if
34:        end for
35:      end if
36:    end for
37:    if  $count \geq nSort$  then
38:      break
39:    end if
40:  end while
41:   $MaxFNo = max(FrontNo)$   $\triangleright$  Find highest calculated rank
42:   $FrontNo = FrontNo(Loc)$   $\triangleright$  Return results in original order
43: end procedure

```

3.2 Shadow Sort

Algorithm 7 Classic dominance check

```

1: procedure DOMCHECK( $s, t$ ) ▷ Solutions  $s$  and  $t$ 
2:    $r = \text{size}(s, 2)$  ▷ Establish how many objectives
3:   for  $y = 2 : r$  do ▷ Solution  $p \prec$  solution  $t$  in objective 1
4:     if  $t(y) < s(y)$  then
5:        $\text{result} = \text{false}$ 
6:       break
7:     end if
8:      $\text{result} = \text{true}$ 
9:   end for
10: end procedure

```

3.2.4 Analysis of Computational Complexity

Some of the non-dominated sorting algorithms discussed in Chapter 2.2 require every solution to be assessed and ranked for the algorithm to work. Although comprehensive, this is seldom necessary. A major advantage of Shadow Sort is that the algorithm can accommodate the particular output required by the MOO algorithm using it. Since Shadow Sort identifies all preceding solutions which are dominated by the current solution under assessment, no time is wasted in assessing a solution that is not fit for the current rank in question. This is a useful strategy if only a specified number of solutions are requested. Furthermore, after the algorithm has ranked enough solutions as specified by the decision-maker, it simply returns the rank of the assessed solutions, and assigns a rank of infinity to those that were not.

Computational complexity was discussed in Section 2.2 and is now explored with reference to Shadow Sort.

Case I: The algorithm prepares the population by first sorting along the first objective (with Quick Sort (Hoare, 1961), therefore $\mathcal{O}(N \log N)$), before performing R loops through solutions 1 to N , where R is the number of ranks requested by the decision-maker or multi-objective optimisation algorithm. Best case time complexity for Shadow Sort Case I is $\mathcal{O}(N \log N)$, which is the case when all solutions are Rank 1 and worst case time complexity is $\mathcal{O}(N^2)$, which is the case when each solution forms part of its own rank (*i.e.* there are N ranks).

3.3 Summary: Chapter 3

Case II: The overhead preparation for Case II involves sorting the population along each objective, therefore exhibiting complexity $\mathcal{O}(MN \log N)$. During the actual non-dominated sorting, the loop may run up to N times if all solutions are to be ranked, and each solution forms part of its own rank. Therefore, best case time complexity for Shadow Sort Case II is $\mathcal{O}(MN \log N)$ when all solutions form part of Rank 1, and worst case time complexity is $\mathcal{O}(MN^2)$ when each solution has its own rank.

Case III: Since the initialisation of Case III is very similar to that of Case II, the overhead preparation is also $\mathcal{O}(MN \log N)$. During non-dominated sorting, the best case computation is realised when all solutions are Rank 1, and time complexity for Shadow Sort Case III is $\mathcal{O}(N \log N)$. The worst case scenario is when each solution forms part of its own rank and worst case time complexity is $\mathcal{O}(MN^2)$.

3.3 Summary: Chapter 3

In this chapter, the novel non-dominated sorting algorithm Shadow Sort was introduced in an effort to improve the runtime of multi-objective evolutionary algorithms. It was shown that it is applicable to different cases based on the number of objectives, while the pseudocode for all the cases was presented. The complexity of Shadow Sort was finally discussed.

In subsequent chapters, Shadow Sort will be compared to other recently-proposed non-dominated sorting algorithms. Next, both Shadow Sort and the best-performing competitor algorithm will be implemented into the five multi-objective evolutionary algorithms discussed in Subsection 2.5.

Chapter 4

Evaluation against existing non-dominated sorting algorithms

In order to assess the performance of Shadow Sort, various tests were conducted against three of the non-dominated sorting algorithms introduced in Chapter 2, namely Deductive Sort (DS) (McClymont & Keedwell, 2012), Efficient Non-dominated Sort (ENS) (Zhang *et al.*, 2015), and Best Order Sort (BOS) (Roy *et al.*, 2016). The test procedure and test results are subsequently presented.

4.1 Test Procedure

The test procedure for comparing Shadow Sort against existing non-dominated sorting algorithms is summarised as follows:

1. Generate random population data of varying sizes and dimensions.
2. Perform non-dominated sorting on the randomly-generated populations with DS, ENS, BOS and SS. For Test 1, this will be non-dominated sorting for the entire population, whereas for Test 2, only the solutions belonging to the first non-dominated set are to be ranked.

4.2 Test 1: Entire population ranking

3. Compare the outputs of Step 2 to ensure non-dominated sorting algorithms provide the same ranking.
4. Repeat steps 1-3 for 100 replications.
5. Calculate the mean runtime and standard deviations for each population variation and for each non-dominated sorting algorithm.

The code for all the non-dominated sorting algorithms included in these tests is supplied in Appendix B. All tests were conducted on a Microsoft Windows 10 operating system with an Intel[®] Core[™] i7-8550U CPU at 1.80 GHz.

4.2 Test 1: Entire population ranking

The first set of tests were conducted by having the non-dominated sorting algorithms perform ranking on the entire population of a randomly-generated dataset. The datasets were uniform random data generated by MATLAB's *rand(n,m)* function. This function creates an array with n rows (solutions) and m columns (objectives). The effect of the size of a population was explored by performing non-dominated sorting on populations of the following sizes; 100, 200, 300, 400, 500, 1000, 2000, 3000, 4000, 5000 and 10000. Furthermore, the effect of the number of objectives was also explored by performing non-dominated sorting on populations with the following number of objectives; 2, 3, 4, 5, 10, 15 and 20.

100 replications of each test were performed, and each non-dominated sorting algorithm was given the same population to rank during each replication, all of which yielded the same ranking results. The runtimes of the four non-dominated sorting algorithms are provided in Figures 4.1 to 4.7. Note that the results are presented on logarithmic axes. Table 4.1 presents the mean runtimes of the algorithms after the 100 replications, as well as the calculated standard deviations.

As described in Subsection 3.2.1, SS employs Case I for problems with two dimensions. The results presented in Figure 4.1 indicate that SS is the superior algorithm for two-dimensional problems. This is true for all the population sizes tested. The second best algorithm was ENS, which had runtimes similar to

4.2 Test 1: Entire population ranking

SS for smaller population sizes. However, once the population size grew above 1 000, the difference in runtimes between SS and ENS became larger. DS achieved faster runtimes than BOS for smaller population sizes, but was outperformed by BOS once the population size exceeded around 500.

Figures 4.2 to 4.7 present a similar pattern for populations with three to 20 objectives. They indicate that ENS is the superior algorithm for populations with three or more objectives and for all sizes when entire population non-dominated sorting is required. As in the two-dimensional populations presented in Figure 4.1, DS outperforms BOS when the population size is low. However, as the number of objectives increase, the population size at which BOS outperforms DS gets smaller, ultimately lying between 100 and 200 for 20-dimensional populations.

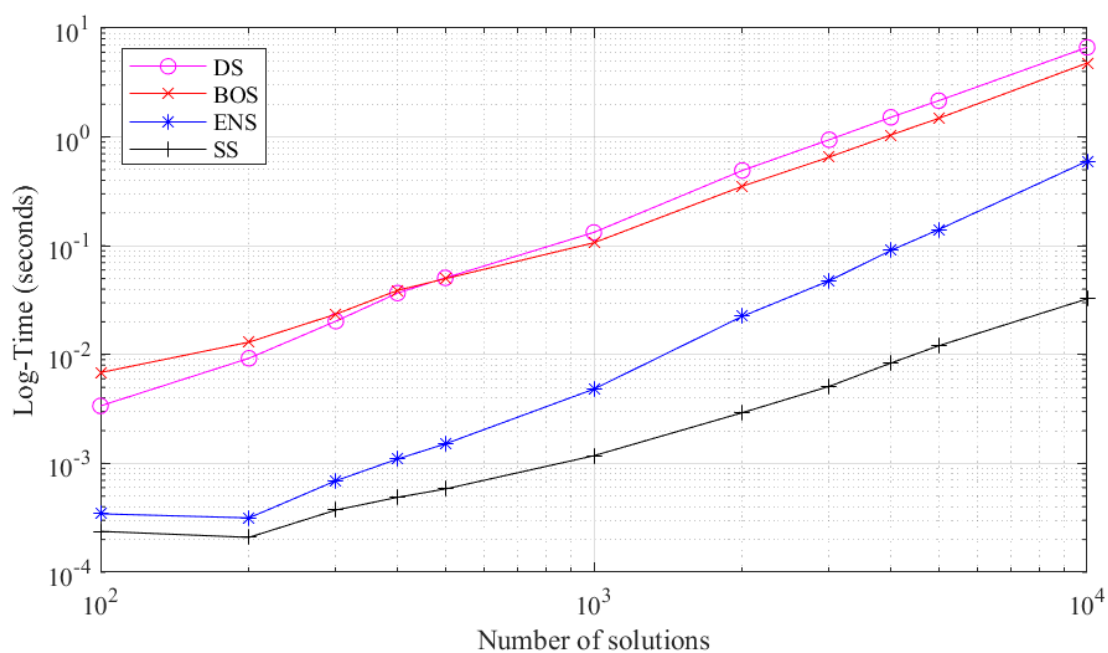


Figure 4.1: Entire population sorting of a randomly-generated dataset with two dimensions

4.2 Test 1: Entire population ranking

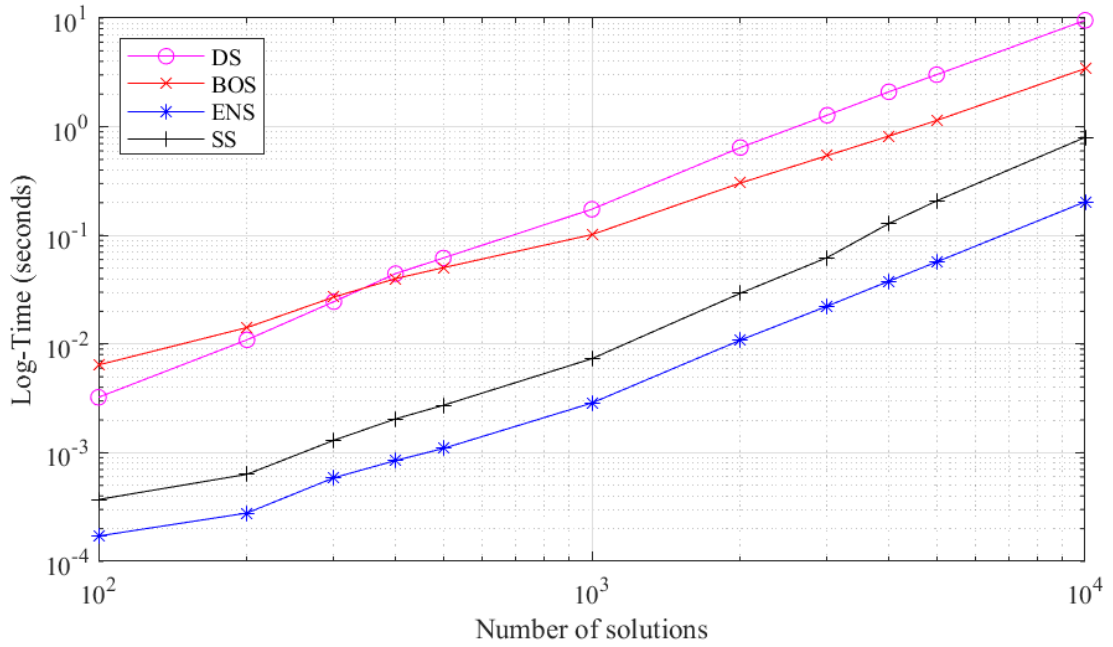


Figure 4.2: Entire population sorting of a randomly-generated dataset with three dimensions

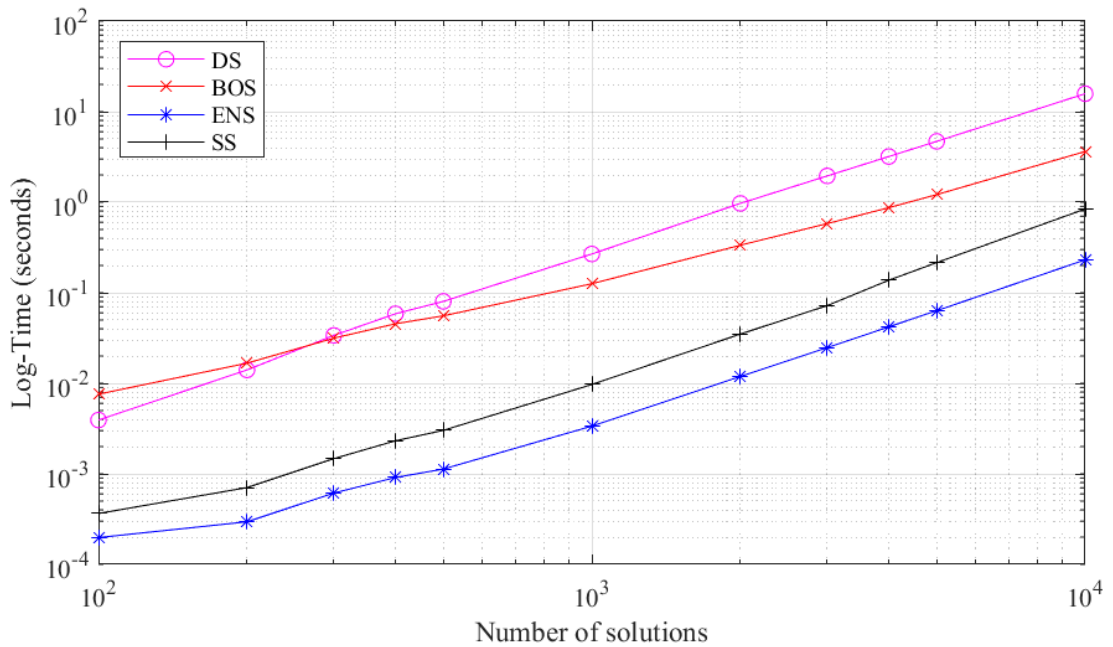


Figure 4.3: Entire population sorting of a randomly-generated dataset with four dimensions

4.2 Test 1: Entire population ranking

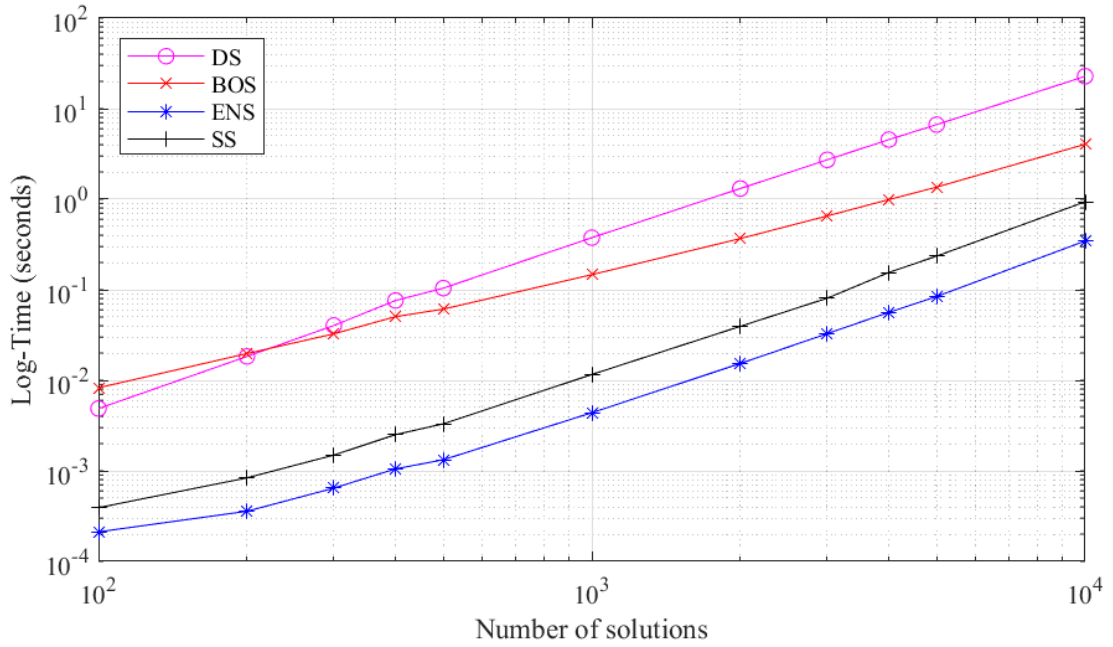


Figure 4.4: Entire population sorting of a randomly-generated dataset with five dimensions

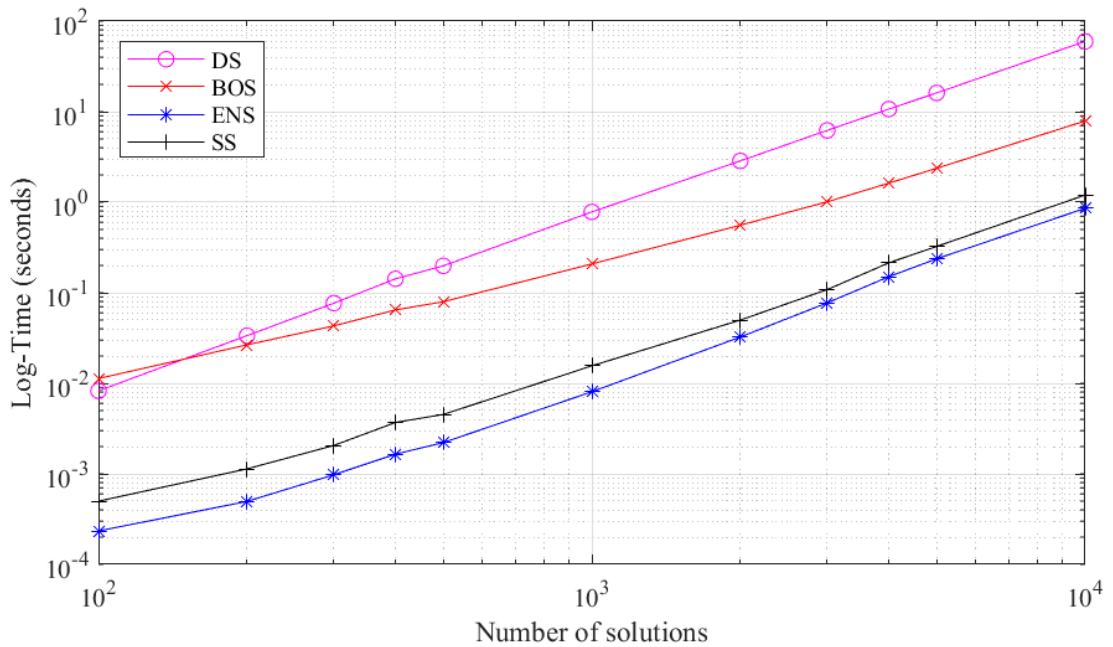


Figure 4.5: Entire population sorting of a randomly-generated dataset with 10 dimensions

4.2 Test 1: Entire population ranking

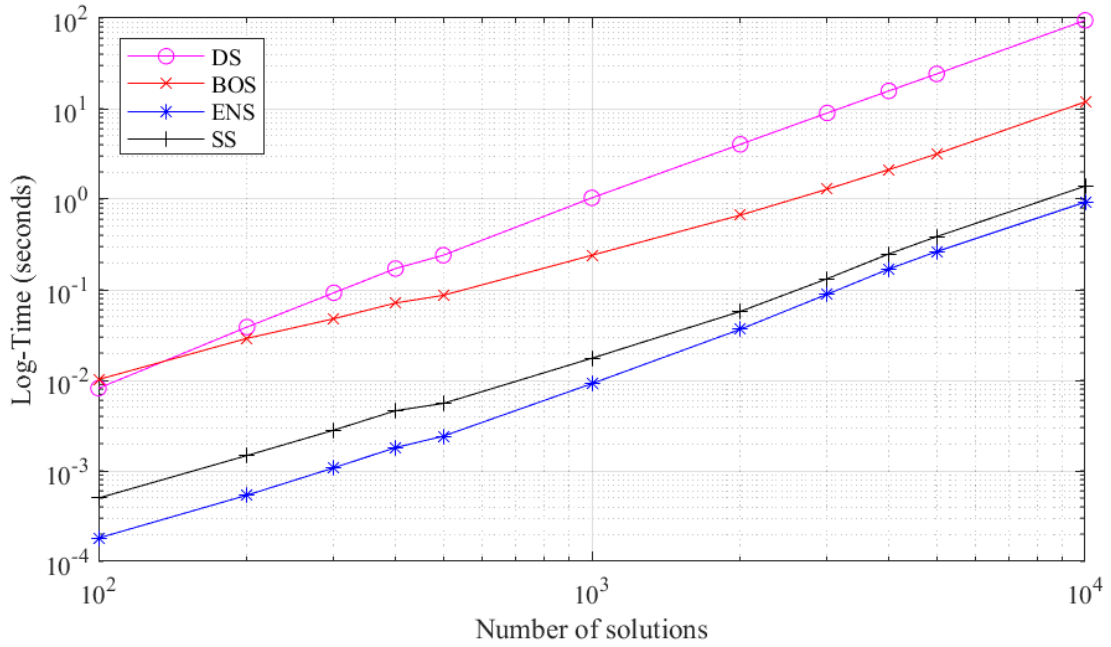


Figure 4.6: Entire population sorting of a randomly-generated dataset with 15 dimensions

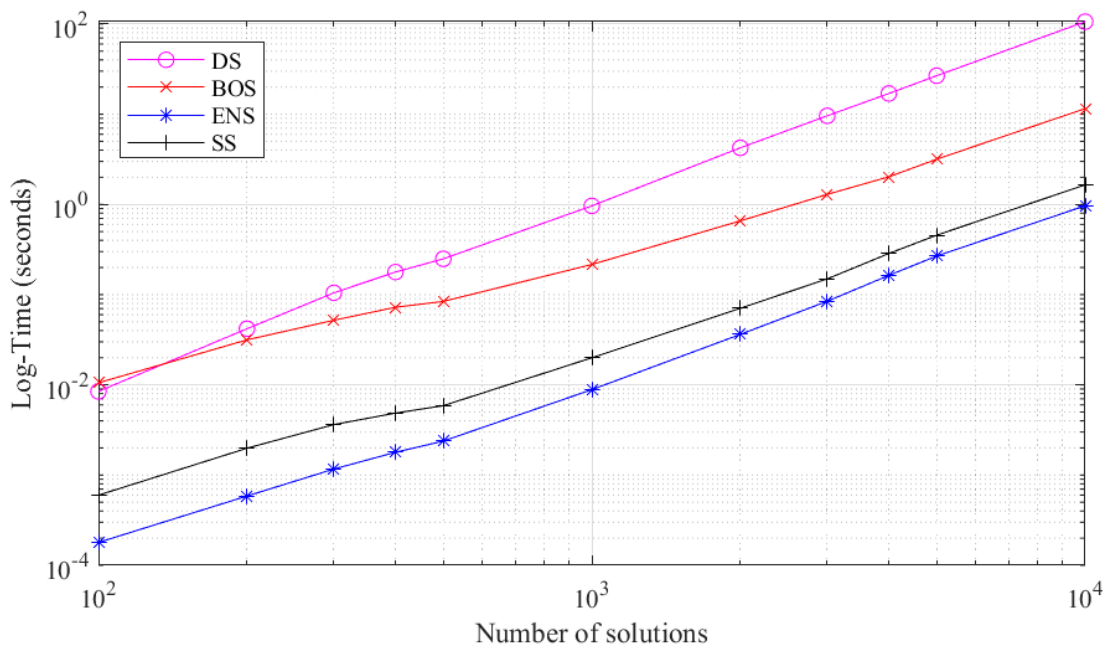


Figure 4.7: Entire population sorting of a randomly-generated dataset with 20 dimensions

Table 4.1: Results of the entire population ranking test

		M2		M3		M4		M5		M10		M15		M20	
		Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD
100	DS	3.38	0.99	3.24	0.55	3.93	0.50	4.88	0.70	8.26	1.35	8.17	0.24	8.46	0.67
	ENS	0.34	0.93	0.17	0.05	0.20	0.05	0.21	0.05	0.23	0.05	0.18	0.03	0.18	0.03
	BOS	6.84	2.30	6.43	0.80	7.58	0.74	8.26	1.39	11.28	2.35	10.20	1.95	10.54	2.00
	SS	0.24	0.36	0.37	0.38	0.36	0.08	0.39	0.07	0.50	0.10	0.50	0.04	0.60	0.04
200	DS	9.25	0.82	10.94	1.00	14.05	1.44	18.34	2.02	33.57	1.51	38.70	1.74	41.92	3.41
	ENS	0.32	0.05	0.28	0.05	0.30	0.05	0.36	0.05	0.50	0.03	0.54	0.03	0.59	0.06
	BOS	13.03	1.04	14.19	1.03	16.69	1.69	19.73	2.40	26.58	4.10	29.04	4.21	31.54	5.56
	SS	0.21	0.02	0.63	0.04	0.70	0.09	0.84	0.08	1.13	0.06	1.48	0.07	1.99	0.26
300	DS	20.29	1.37	24.57	2.16	33.83	3.53	40.33	3.30	76.75	4.51	92.57	2.56	104.76	5.45
	ENS	0.69	0.07	0.59	0.04	0.61	0.07	0.64	0.04	0.98	0.07	1.08	0.07	1.17	0.13
	BOS	23.52	1.25	27.08	2.32	31.50	3.35	32.60	2.77	43.10	4.59	47.87	6.86	52.02	7.71
	SS	0.37	0.03	1.30	0.05	1.47	0.20	1.49	0.06	2.06	0.15	2.82	0.28	3.63	0.40
400	DS	36.77	2.47	44.47	3.84	58.62	5.00	76.06	5.81	142.59	7.80	171.43	8.79	178.06	9.22
	ENS	1.10	0.12	0.85	0.07	0.91	0.09	1.05	0.08	1.65	0.09	1.81	0.16	1.79	0.10
	BOS	38.92	2.33	40.01	2.10	45.44	3.88	50.43	3.42	64.58	7.20	71.40	11.70	72.46	12.96
	SS	0.49	0.06	2.05	0.10	2.32	0.22	2.51	0.10	3.69	0.35	4.62	0.50	4.87	0.70
500	DS	50.72	3.32	61.95	4.83	80.30	5.04	104.18	9.19	198.23	9.49	240.64	5.58	249.51	8.41
	ENS	1.51	0.16	1.10	0.08	1.13	0.07	1.32	0.10	2.21	0.11	2.42	0.11	2.38	0.06
	BOS	49.86	3.18	50.47	2.78	55.61	3.72	61.14	5.19	79.62	8.64	86.80	10.82	84.11	9.95
	SS	0.58	0.07	2.73	0.10	3.02	0.11	3.31	0.25	4.53	0.35	5.55	0.49	5.88	0.45
1000	DS	132.73	7.08	174.10	9.36	268.04	15.92	376.03	23.49	780.84	34.76	1 034.64	42.80	963.31	52.32
	ENS	4.82	0.57	2.87	0.22	3.36	0.30	4.38	0.48	8.06	0.36	9.19	0.55	8.86	2.50
	BOS	106.57	5.97	101.68	5.90	125.78	7.34	146.71	14.46	208.01	15.81	240.20	25.02	215.99	28.81
	SS	1.18	0.19	7.34	0.43	9.69	0.46	11.59	0.75	15.59	1.36	17.52	1.83	19.96	2.96

4.2 Test 1: Entire population ranking

Table 4.1: Results of the entire population ranking test (continued)

		M2		M3		M4		M5		M10		M15		M20	
		Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD
2000	DS	492.11	21.14	643.82	26.86	970.30	38.19	1 312.05	77.44	2 859.39	141.96	4 035.62	76.48	4 242.22	59.20
	ENS	22.47	3.13	10.87	0.67	11.91	0.73	15.34	1.02	32.44	1.68	36.61	0.93	36.27	0.90
	BOS	352.11	13.62	303.49	12.10	334.86	15.43	367.49	18.59	557.24	38.23	666.88	50.99	657.06	52.13
	SS	2.93	0.23	29.35	1.03	34.96	0.88	39.43	2.63	49.93	3.70	57.60	3.71	70.64	5.31
3000	DS	942.87	34.15	1 276.60	54.51	1 951.49	82.38	2 722.25	113.29	6 210.86	241.13	8 960.18	202.38	9 596.17	177.84
	ENS	47.70	4.97	22.35	1.52	24.73	1.20	32.75	2.33	77.31	3.90	89.24	5.89	84.45	6.55
	BOS	653.60	23.44	542.60	34.53	580.45	35.02	653.79	45.37	1 010.30	62.70	1 290.29	141.33	1 292.05	122.94
	SS	5.11	0.30	62.59	2.99	72.10	2.73	80.63	3.89	108.35	6.86	132.39	13.72	149.51	11.43
4000	DS	1 515.55	45.47	2 098.12	80.75	3 195.01	130.58	4 551.90	158.09	10 685.56	434.30	15 705.36	337.97	16 969.19	258.13
	ENS	90.69	12.33	37.90	1.95	42.05	1.82	56.32	3.97	150.49	7.21	169.52	5.67	163.46	6.12
	BOS	1 042.51	31.96	820.62	31.84	870.57	36.81	990.36	59.34	1 623.77	122.20	2 120.68	149.18	2 022.79	129.00
	SS	8.36	0.85	128.58	4.36	137.54	7.69	155.07	12.56	214.02	19.16	248.75	16.57	285.78	15.09
5000	DS	2 154.06	58.77	3 016.89	108.27	4 697.05	151.38	6 664.32	248.17	16 113.25	573.83	24 195.50	494.10	26 696.02	1 094.42
	ENS	140.85	16.20	57.08	3.28	63.37	3.38	84.18	4.99	237.05	9.72	263.96	6.14	267.95	8.63
	BOS	1 483.05	48.94	1 143.77	40.80	1 212.37	58.27	1 360.05	88.67	2 369.65	180.74	3 164.23	228.41	3 168.63	331.03
	SS	12.05	0.68	208.15	9.34	215.01	13.15	235.32	18.71	324.77	15.51	385.87	17.61	455.99	26.59
10000	DS	6 663.11	148.26	9 544.98	319.16	15 800.61	1 108.58	22 790.83	769.96	59 668.67	3 596.56	94 444.72	2 076.15	106 116.04	2 188.13
	ENS	599.48	59.43	204.13	9.38	230.89	23.76	345.84	12.26	857.31	39.18	928.15	21.21	966.83	9.10
	BOS	4 780.43	131.88	3 432.31	159.55	3 630.86	687.51	4 044.38	157.08	7 898.00	592.37	11 874.15	819.48	11 545.94	757.11
	SS	32.53	1.83	800.68	51.66	842.08	65.48	936.87	30.37	1 200.22	70.46	1 395.34	52.54	1 648.12	56.22

 4.3 Test 2: Partial population ranking

4.3 Test 2: Partial population ranking

As described in Subsection 2.5.2, it is generally not necessary for a non-dominated sorting algorithm to rank the entire population of solutions. In the case of NSGA II, it is only necessary to rank enough solutions so as to form complete rank sets summing to at least the size of the initial population P_t . Figure 2.5 is repeated for the reader's benefit.

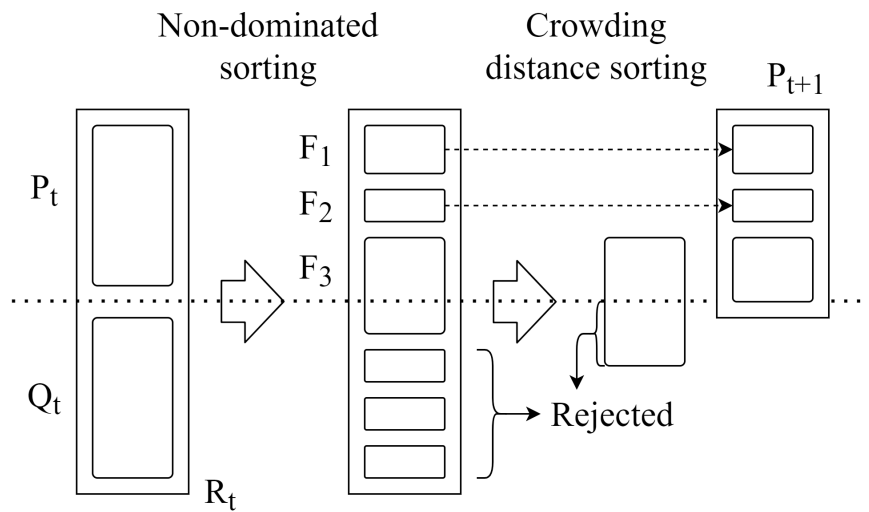


Figure 2.5: Main NSGA II procedure (Deb *et al.*, 2002)

Those solutions whose entire rank set can fit into the next population P_{t+1} will be added. The solutions of the rank set which does not entirely fit into P_{t+1} will then have their crowding-distance parameter calculated, which will then ultimately be used to dictate which of those solutions are to be added to P_{t+1} .

The second test performed on the non-dominated sorting algorithms requires that they only process Rank 1 solutions. This is a more practical test as it simulates how the non-dominated sorting algorithms are actually used in many multi-objective evolutionary algorithms (Bekker, 2012; Deb *et al.*, 2002). Of the four non-dominated sorting algorithms tested, the runtimes of DS (McClymont & Keedwell, 2012) and SS should benefit most from this configuration, as they are designed in such a manner that non-dominated sorting takes place per rank. It is possible to stop the algorithm once the required number of solutions have

4.3 Test 2: Partial population ranking

been ranked for DS, ENS and SS. BOS (Roy *et al.*, 2016) employs a different strategy where solutions are processed in an order dictated by their position in the various objectives. It is therefore not possible to interrupt the algorithm until all solutions have been ranked. The runtimes reported for BOS are therefore for entire population sorting.

Again, 100 replications of the test were performed, and each non-dominated sorting algorithm was given the same population to rank during each replication. The runtime results of the four non-dominated sorting algorithms are provided in Figures 4.8 to 4.14. Table 4.2 presents the mean runtimes of the algorithms after the 100 replications, as well as the calculated standard deviations.

Figure 4.8 indicates that SS is still the superior algorithm for populations with two objectives. ENS achieved the second fastest runtimes for all population sizes up to 5 000, after which DS achieved the second fastest runtime. The slowest runtimes were achieved by BOS throughout the different population sizes. As shown in Figure 4.9, ENS and SS achieved very similar runtimes for populations with three objectives. ENS was marginally faster on populations with less than 300 solutions, and again for populations with between 500 and 1 000 solutions. For populations with more than 1 000 solutions, SS achieved the fastest runtimes. As seen in Figures 4.10 to 4.14, ENS achieved the fastest runtimes on all sizes for populations with four or more objectives. Figure 4.12 shows that DS becomes the slowest non-dominated sorting algorithm once the number of objectives exceeds 10. Figure 4.14 shows that by the time the number of objectives equals 20, DS becomes significantly slower than the other non-dominated sorting algorithms.

4.3 Test 2: Partial population ranking

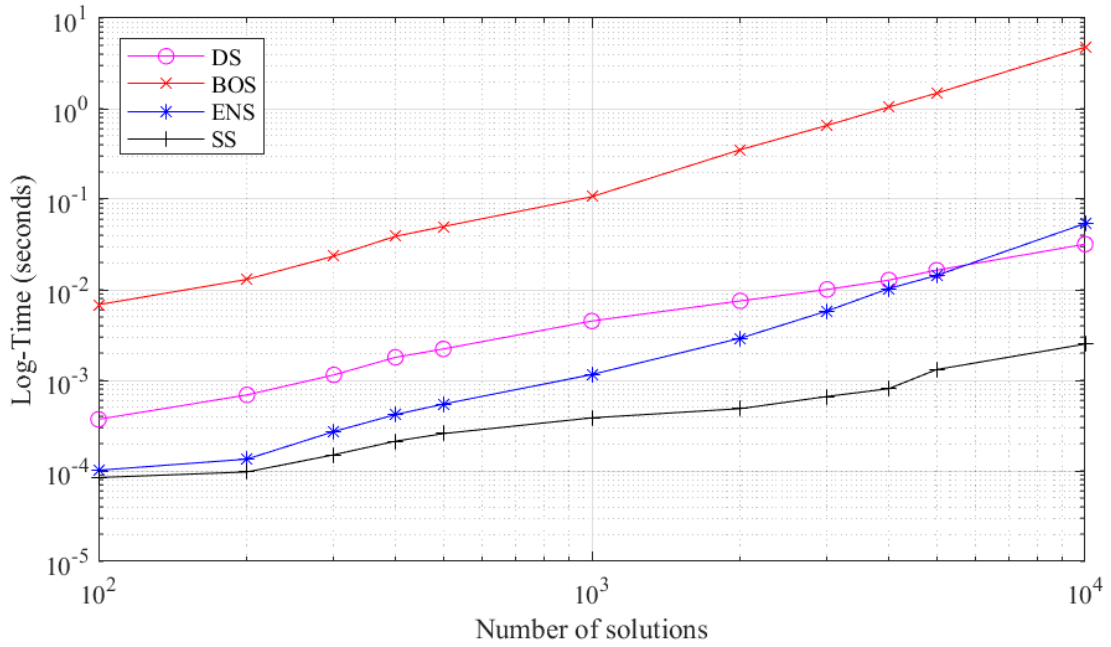


Figure 4.8: Partial population sorting of a randomly-generated dataset with two dimensions

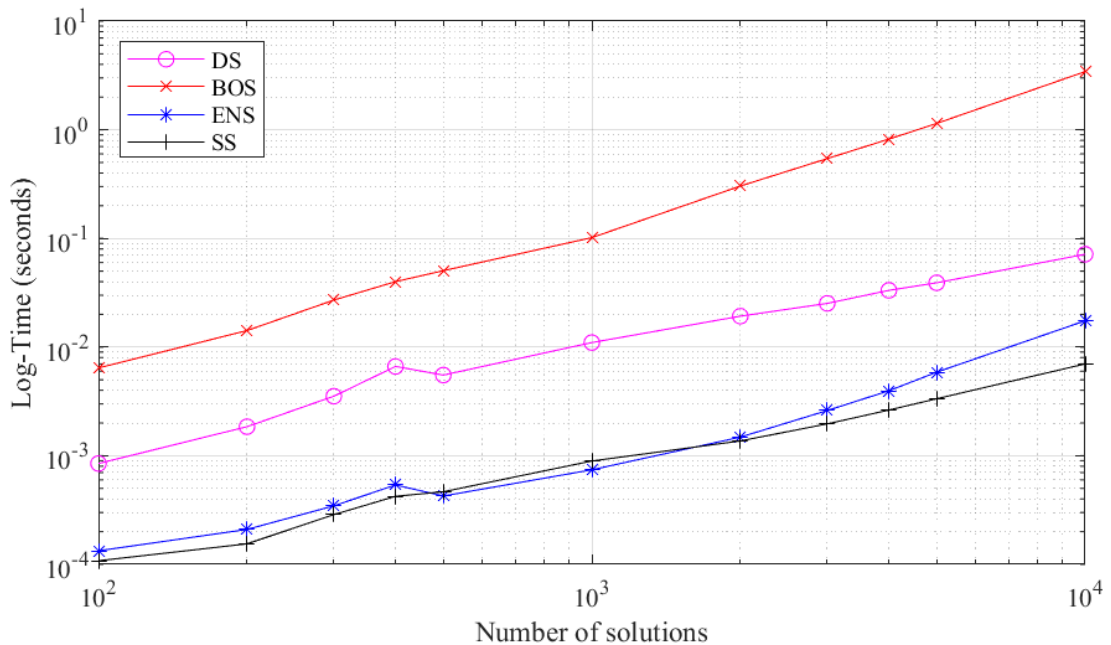


Figure 4.9: Partial population sorting of a randomly-generated dataset with three dimensions

4.3 Test 2: Partial population ranking

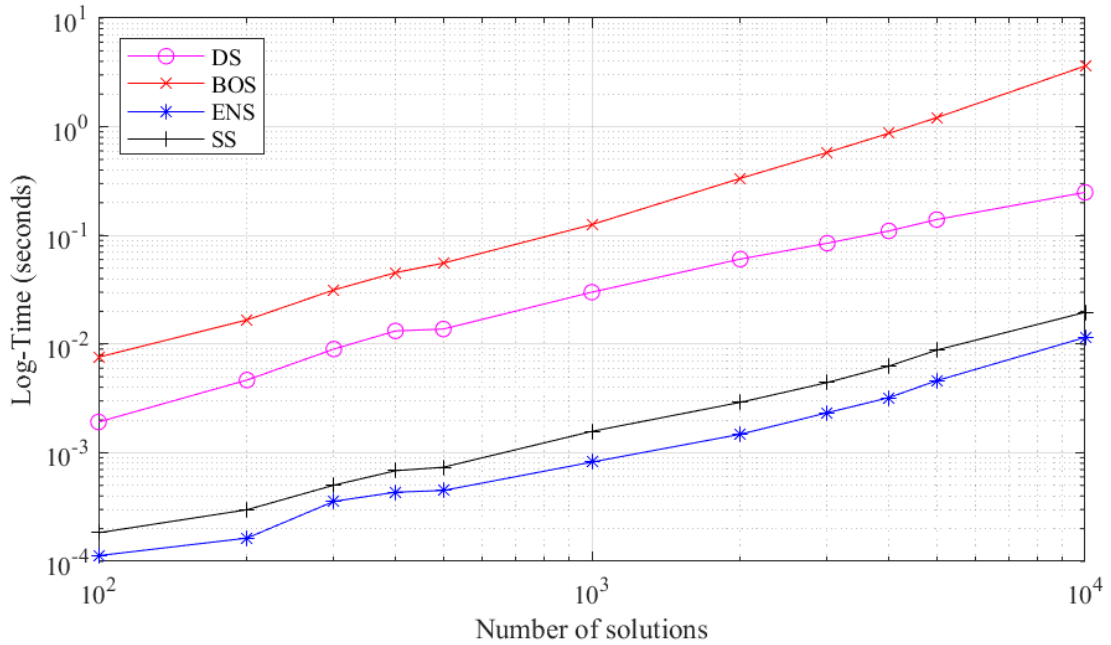


Figure 4.10: Partial population sorting of a randomly-generated dataset with four dimensions

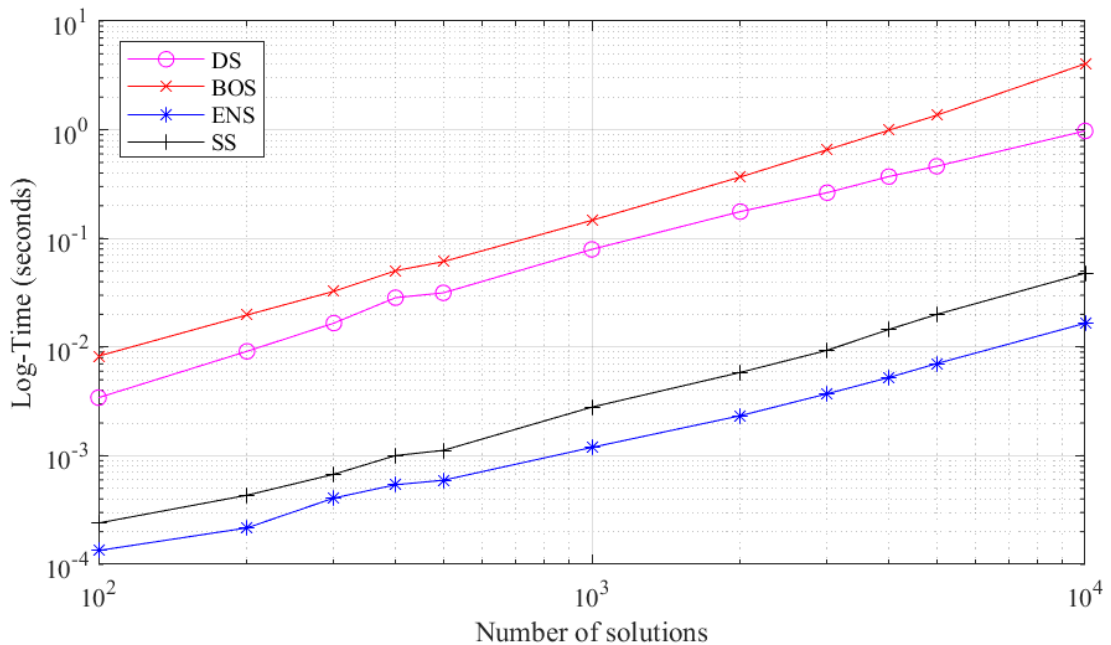


Figure 4.11: Partial population sorting of a randomly-generated dataset with five dimensions

4.3 Test 2: Partial population ranking

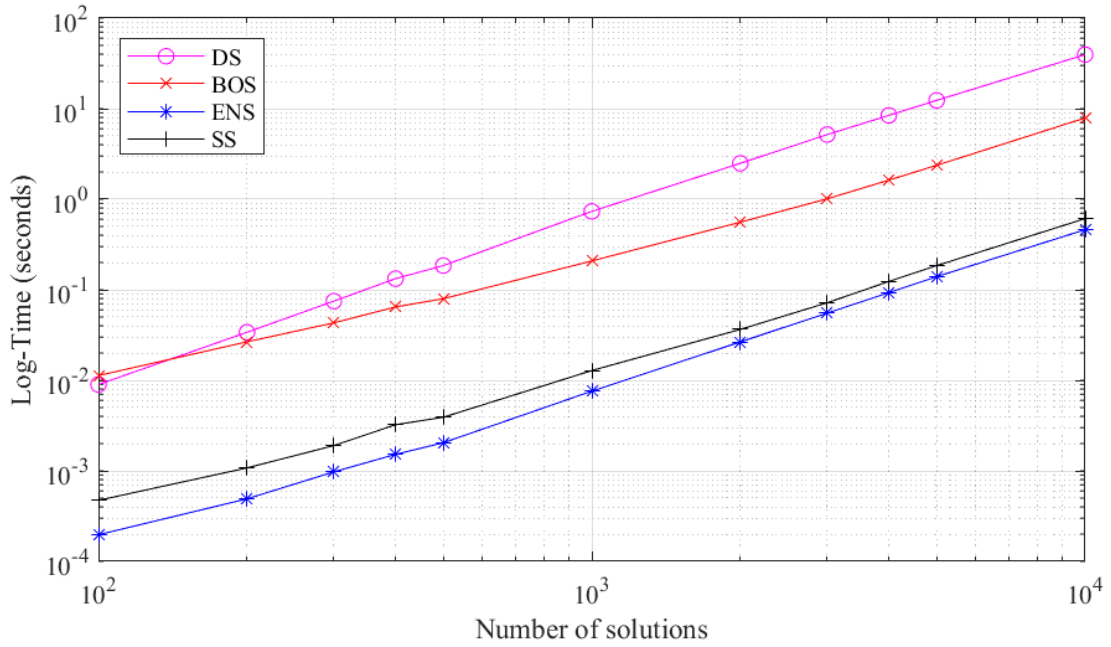


Figure 4.12: Partial population sorting of a randomly-generated dataset with 10 dimensions

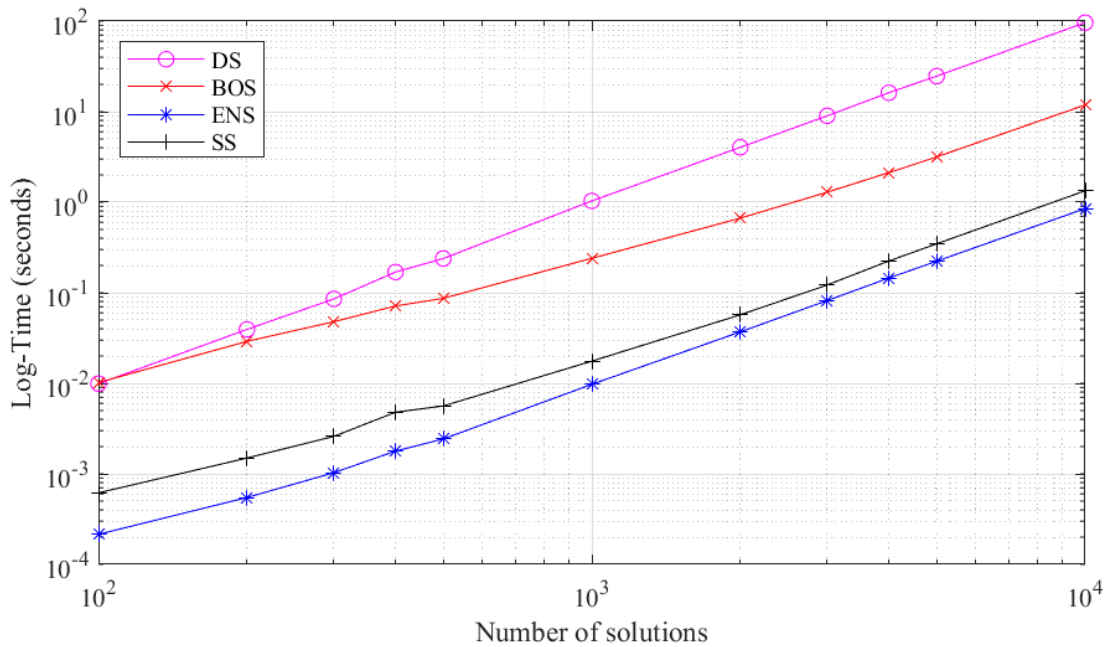


Figure 4.13: Partial population sorting of a randomly-generated dataset with 15 dimensions

4.3 Test 2: Partial population ranking

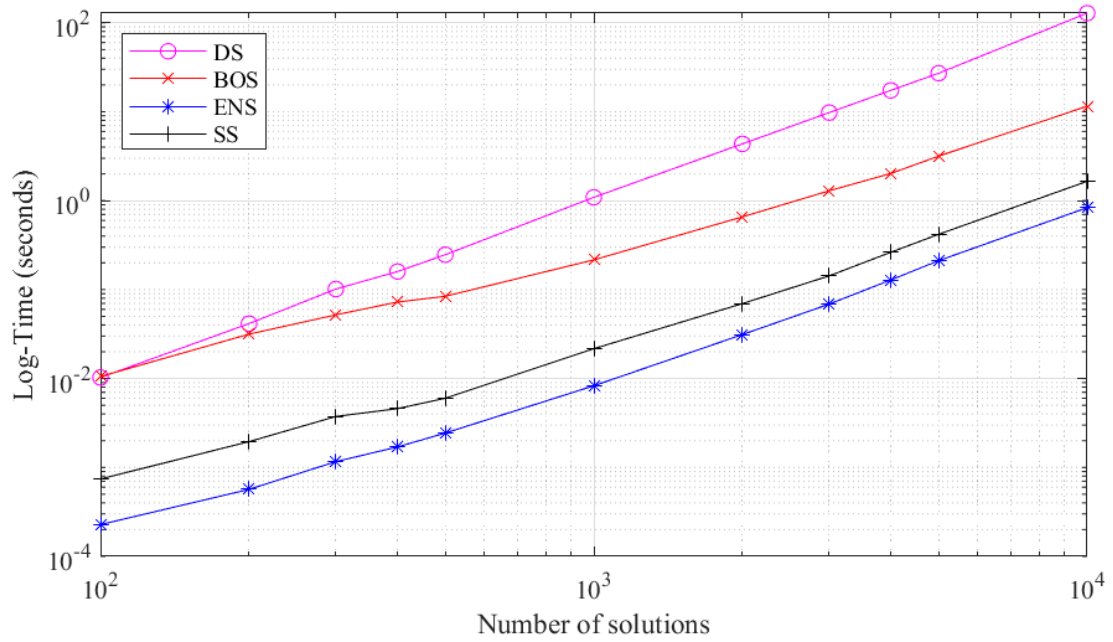


Figure 4.14: Partial population sorting of a randomly-generated dataset with 20 dimensions

Table 4.2: Results of the partial population ranking test

		M2		M3		M4		M5		M10		M15		M20	
		Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD
100	DS	0.37	0.10	0.85	0.28	1.92	0.52	3.42	0.82	8.98	1.03	9.92	1.05	10.27	0.27
	ENS	0.10	0.04	0.13	0.01	0.11	0.00	0.13	0.03	0.20	0.03	0.21	0.05	0.23	0.05
	BOS	6.84	2.30	6.43	0.80	7.58	0.74	8.26	1.39	11.28	2.35	10.20	1.95	10.54	2.00
	SS	0.08	0.04	0.10	0.04	0.18	0.02	0.24	0.03	0.47	0.05	0.62	0.09	0.74	0.06
200	DS	0.69	0.15	1.85	0.58	4.67	1.24	9.14	2.18	33.92	1.82	39.39	1.19	41.47	1.45
	ENS	0.13	0.02	0.21	0.01	0.16	0.02	0.22	0.04	0.49	0.05	0.55	0.06	0.57	0.05
	BOS	13.03	1.04	14.19	1.03	16.69	1.69	19.73	2.40	26.58	4.10	29.04	4.21	31.54	5.56
	SS	0.10	0.01	0.15	0.02	0.30	0.03	0.43	0.05	1.08	0.06	1.49	0.08	1.95	0.20
300	DS	1.15	0.32	3.53	1.32	8.99	2.75	16.61	4.44	74.84	8.43	85.48	2.46	101.48	18.76
	ENS	0.27	0.06	0.35	0.06	0.36	0.04	0.41	0.06	0.97	0.11	1.03	0.03	1.16	0.16
	BOS	23.52	1.25	27.08	2.32	31.50	3.35	32.60	2.77	43.10	4.59	47.87	6.86	52.02	7.71
	SS	0.15	0.03	0.29	0.06	0.50	0.06	0.68	0.08	1.90	0.34	2.59	0.21	3.74	0.83
400	DS	1.79	0.43	6.63	2.41	13.22	4.78	28.47	7.23	132.60	16.03	169.32	26.80	159.62	9.79
	ENS	0.42	0.09	0.54	0.19	0.43	0.11	0.54	0.09	1.52	0.15	1.78	0.23	1.70	0.14
	BOS	38.92	2.33	40.01	2.10	45.44	3.88	50.43	3.42	64.58	7.20	71.40	11.70	72.46	12.96
	SS	0.21	0.06	0.42	0.14	0.68	0.18	1.00	0.19	3.23	0.56	4.79	1.12	4.59	0.51
500	DS	2.22	0.70	5.52	1.72	13.75	4.02	31.56	8.05	184.76	11.67	238.68	6.76	246.90	5.02
	ENS	0.55	0.34	0.43	0.06	0.45	0.05	0.59	0.10	2.04	0.11	2.43	0.07	2.44	0.05
	BOS	49.86	3.18	50.47	2.78	55.61	3.72	61.14	5.19	79.62	8.64	86.80	10.82	84.11	9.95
	SS	0.26	0.30	0.47	0.07	0.74	0.09	1.12	0.16	3.91	0.55	5.62	0.36	6.00	0.27
1000	DS	4.51	1.01	11.00	2.98	30.02	7.95	79.27	15.98	733.82	45.33	1 032.81	34.91	1 091.37	18.96
	ENS	1.16	0.29	0.74	0.13	0.82	0.10	1.19	0.14	7.59	0.34	9.74	0.35	8.29	0.50
	BOS	106.57	5.97	101.68	5.90	125.78	7.34	146.71	14.46	208.01	15.81	240.20	25.02	215.99	28.81
	SS	0.38	0.10	0.89	0.16	1.57	0.23	2.80	0.36	12.82	1.06	17.57	1.21	21.65	1.79

Table 4.2: Results of the partial population ranking test (continued)

	M2		M3		M4		M5		M10		M15		M20		
	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	Time (ms)	SD	
2000	DS	7.53	1.37	19.25	4.24	60.58	15.79	176.60	30.07	2 491.00	208.80	4 038.07	79.79	4 351.71	35.75
	ENS	2.92	0.97	1.48	0.30	1.48	0.14	2.34	0.21	26.52	1.30	36.97	1.25	31.19	0.50
	BOS	352.11	13.62	303.49	12.10	334.86	15.43	367.49	18.59	557.24	38.23	666.88	50.99	657.06	52.13
	SS	0.49	0.09	1.37	0.22	2.91	0.46	5.85	0.77	36.48	3.25	57.08	3.76	69.59	3.93
3000	DS	10.08	2.02	25.23	6.33	84.71	21.20	263.65	56.97	5 176.16	353.72	9 002.92	198.51	9 788.17	197.27
	ENS	5.84	1.96	2.61	0.62	2.33	0.21	3.72	0.30	55.29	2.96	81.99	2.64	68.73	2.23
	BOS	653.60	23.44	542.60	34.53	580.45	35.02	653.79	45.37	1 010.30	62.70	1 290.29	141.33	1 292.05	122.94
	SS	0.66	0.11	1.96	0.58	4.42	0.76	9.33	1.41	71.83	4.80	121.86	6.81	142.67	8.57
4000	DS	12.78	2.42	33.27	9.01	109.93	23.67	371.18	70.89	8 435.11	518.10	16 194.83	492.92	17 396.79	413.22
	ENS	10.31	4.00	3.96	0.98	3.21	0.34	5.25	0.46	92.76	3.93	145.39	4.99	128.51	9.00
	BOS	1 042.51	31.96	820.62	31.84	870.57	36.81	990.36	59.34	1 623.77	122.20	2 120.68	149.18	2 022.79	129.00
	SS	0.81	0.13	2.63	0.45	6.27	1.40	14.52	1.87	123.41	7.71	224.34	14.42	263.15	17.72
5000	DS	16.45	2.98	39.05	11.06	139.93	27.47	461.43	91.15	12 353.65	787.11	24 697.72	493.03	27 113.92	291.71
	ENS	14.39	5.76	5.87	2.25	4.60	1.01	7.04	0.53	139.51	9.85	223.04	12.33	211.46	3.98
	BOS	1 483.05	48.94	1 143.77	40.80	1 212.37	58.27	1 360.05	88.67	2 369.65	180.74	3 164.23	228.41	3 168.63	331.03
	SS	1.31	0.16	3.34	0.46	8.80	1.33	19.97	2.60	184.75	17.49	347.27	19.24	418.61	24.45
10000	DS	31.88	6.60	71.49	17.03	249.20	57.30	976.49	216.66	39 551.82	3 319.37	96 220.14	1 680.40	128 414.08	173 209.55
	ENS	53.95	19.69	17.47	5.45	11.52	1.88	16.57	2.45	461.04	29.20	851.44	24.17	834.50	48.62
	BOS	4 780.43	131.88	3 432.31	159.55	3 630.86	687.51	4 044.38	157.08	7 898.00	592.37	11 874.15	819.48	11 545.94	757.11
	SS	2.51	0.24	7.00	0.83	19.61	3.01	47.95	8.32	610.66	45.45	1 336.09	60.62	1 651.55	65.25

4.3 Test 2: Partial population ranking

Table 4.3: The number of Rank 1 solutions found in a randomly-generated cloud population with 10 000 solutions

Dimensions	Rank 1	Percentage	Dimensions	Rank 1	Percentage
1	N/A	N/A	11	5872	58.7%
2	11	0.1%	12	7323	73.2%
3	56	0.6%	13	7928	79.3%
4	133	1.3%	14	8578	85.8%
5	414	4.1%	15	9105	91.1%
6	925	9.3%	16	9547	95.5%
7	1545	15.5%	17	9645	96.5%
8	2859	28.6%	18	9813	98.1%
9	3960	39.6%	19	9919	99.2%
10	5034	50.3%	20	9942	99.4%

It is interesting to note that the runtimes for entire population sorting and partial population sorting become very similar as the number of objectives increases. This is due to a phenomenon known as the *curse of dimensionality* where there is an exponential increase in volume in the search space with adding extra dimensions (objectives) (Shultz & Fahlman, 2017). Therefore, for a given random population size N , the percentage of Rank 1 non-dominated solutions present increases as the number of dimensions increase. According to Bentley *et al.* (1978), the number of non-dominated k -dimensional vectors in a population of size n is $\mathcal{O}(\ln n^{k-1})$. Unfortunately, this implies that the optimisation of problems with a large number of objectives is carried out almost randomly. In fact, Mostaghim & Schmeck (2008) performed experiments demonstrating that a random search optimiser could achieve better results than NSGA II in a 10-objective problem. This concept is illustrated in Table 4.3 and Figure 4.15, where the number of Rank 1 solutions extracted by Shadow Sort on a random population for two to 20 objectives are shown.

It is partly due to the curse of dimensionality that the class of many-objective evolutionary algorithms has been formed, as it is necessary to develop new methods and strategies to evolve the population towards the Pareto optimal set. However, according to Coello Coello *et al.* (2019), decomposition-based multi-objective evolutionary algorithms (which transform multi-objective problems into

4.4 Summary: Chapter 4

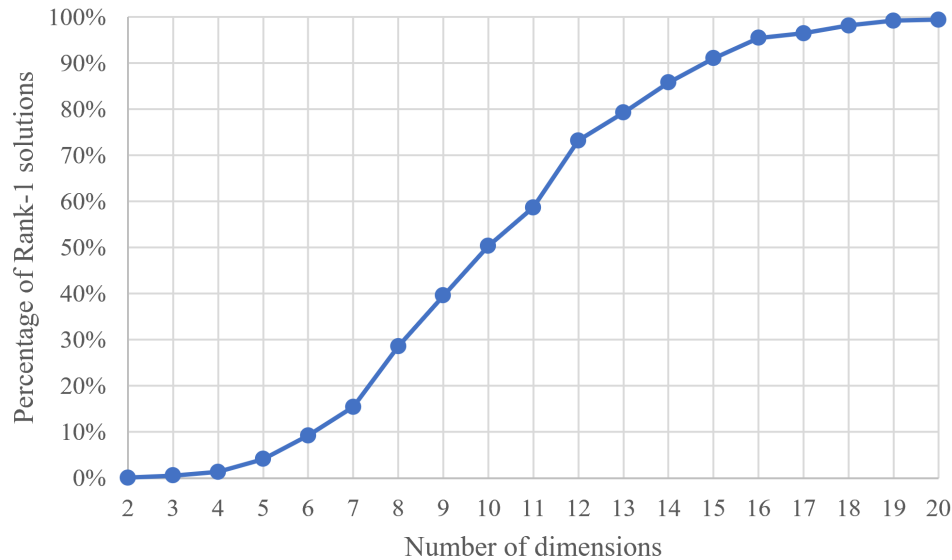


Figure 4.15: The number of Rank 1 solutions found in a randomly-generated cloud population with 10 000 solutions

several single-objective problems which are then optimised in isolation) are expected to be one of the main topics of research over the next few years. Based on this prediction and the test results obtained in this chapter, SS could be expected to compliment such decomposition-based multi-objective evolutionary algorithms well.

4.4 Summary: Chapter 4

Deductive Sort (DS) (McClymont & Keedwell, 2012), Efficient Non-dominated Sort (ENS) (Zhang *et al.*, 2015), Best Order Sort (BOS) (Roy *et al.*, 2016) and Shadow Sort (SS) were extensively compared in this chapter. Two types of tests were performed; one where the entire population underwent non-dominated sorting, and a second where only the Rank 1 solutions were identified. Both of these tests were performed on populations with number of solutions ranging from 100 to 10 000, and for number of objectives ranging from two to 20. The *curse of dimensionality* was also introduced, and its effects were emphasized when comparing the runtime results for entire non-dominated sorting and partial (Rank 1)

4.4 Summary: Chapter 4

non-dominated sorting for populations with 15 and 20 objectives.

Both tests suggest that SS should be employed on problems with two objectives. For problems with three objectives, the results indicate that either ENS or SS may be the best choice as their runtimes are very similar, especially for Test 2 in which partial population sorting was conducted. Despite achieving substantially faster runtimes than DS and BOS, SS was unable to outperform ENS on problems with more than four objectives. This includes problems with 15 or more objectives, where SS changed its sorting strategy to Case III as described in Subsection 3.2.3. Therefore, whilst SS has proven to be a fair choice for non-dominated sorting for problems with any number of dimensions, it is most competitive in lower multi-dimensional problems (with two and three objectives).

In the next chapter, ENS and SS are implemented into the five multi-objective evolutionary algorithms presented in Section 2.5. These modified multi-objective evolutionary algorithms are then used to solve the problems found in the three test suites presented in Section 2.7.

Chapter 5

Shadow Sort tested on multi-objective evolutionary algorithms

In this chapter, the two fastest non-dominated sorting algorithms identified in the previous chapter, SS and ENS, were implemented into the five multi-objective evolutionary algorithms presented in Subsection 2.5. These modified multi-objective evolutionary algorithms were then used to solve the problems created by the test suites presented in Subsection 2.7.

Since the purpose of these tests is to evaluate the effect of SS and ENS on the overall algorithm runtimes, the quality of the algorithm results themselves (as far as optimising the problem is concerned) is of little value. It is, however, important to quantify the quality of the results achieved with the different non-dominated sorting algorithm implementations, as there should not be any significant difference between them. The quality of the results are reported by the hypervolume indicator and presented in Appendix A. Since multi-objective evolutionary algorithms are stochastic techniques, the mean hypervolumes will not be identical. Therefore, the Wilcoxon rank-sum statistical test was performed to identify if any statistically significant difference exists. As in Chapter 4, all tests were conducted in MATLAB. To facilitate such complex testing requirements, as well as the statistical analysis of the results, the Evolutionary Multi-Objective Optimisation Platform *PlatEMO* (Tian *et al.*, 2017) was used.

5.1 Test procedure

As mentioned in Subsection 2.7, the ZDT test suite problems are limited to two objectives. The DTLZ and WFG test suites are both capable of varying the number of objectives in the problems. The test results from Chapter 4, specifically the results presented in Figures 4.1 to 4.3 and Figures 4.8 to 4.10 suggest that SS could be implemented effectively in multi-objective evolutionary algorithms for solving lower-dimensional problems containing two or three objectives. For problems containing four or more dimensions, it was proven that ENS is a more suitable choice for non-dominated sorting. Therefore, for the tests conducted in this chapter, the number of objectives for the DTLZ and WFG test suites was set to three. The stopping criterium for each algorithm was set to 10 000 function evaluations, and each execution was repeated 100 times.

The test procedure for comparing runtimes of multi-objective evolutionary algorithms with different non-dominated sorting algorithms (SS and ENS) is:

1. Modify the original multi-objective evolutionary algorithms (MOPSO, NSGA II, NSGA III, AGE-MOEA and A-NSGA III) to utilise SS and ENS for non-dominated sorting.
2. Use the two variations of each algorithm from Step 1 to solve the applicable two-dimensional ZDT test problems, as well as three-dimensional DTLZ and WFG test problems.
3. Repeat Step 2 for 100 replications.
4. Calculate the mean runtimes and their standard deviations for each algorithm variation and for each test problem.
5. Calculate the mean hypervolumes of the results for each test problem.
6. Perform the Wilcoxon rank-sum test to evaluate if any statistically significant differences in the runtimes and hypervolumes exist between the algorithm variations.

5.1 Test procedure

A standardised population size of 200 was selected for all multi-objective evolutionary algorithms. This is as per the tests conducted by [Coello Coello & Lechuga \(2002\)](#). For each of the subsections which follow, the runtime results are reported. An overview of the multi-objective evolutionary algorithm in question is repeated to provide context for an analysis and discussion of the results. Should there be a specific problem which was more aptly solved by either implementation, the characteristics of that problem are highlighted.

5.2 Testing with the MOPSO algorithm

The runtime results of the SS and ENS implementations within the MOPSO algorithm are presented in Figure 5.1 and Table 5.1. The mean hypervolume calculations for the optimisation results are provided in Table A.1 of Appendix A, and they indicate that there is no statistically significant difference in the quality of the results of the two non-dominated sorting algorithm implementations. Figure 5.1 indicates that implementing SS marginally improved the overall runtime of the MOPSO algorithm for all the problems that were tested. However, Table 5.1 indicates that there is no statistically significant difference in the runtime results for the following problems: DTLZ 1, DTLZ 3, DTLZ 6, DTLZ 7, WFG 5 and WFG 6. The symbols $+ / - / \approx$ in Table 5.1 indicate the result of the Wilcoxon rank-sum test and assume that the SS implementation is the subject of the test (*i.e.* $+$ indicates the SS implementation achieved a statistically significantly superior result).

As reported in Table 2.7, which summarises the characteristics of the DTLZ test suite, the only property that DTLZ 1, DTLZ 3 and DTLZ 7 have in common is having low bias. Similarly, Table 2.11 indicates that the only similarity between WFG 5 and WFG 6 is their concave geometry. It is therefore difficult to classify a specific type of problem which indicates whether implementing ENS or SS is more appropriate.

5.2 Testing with the MOPSO algorithm

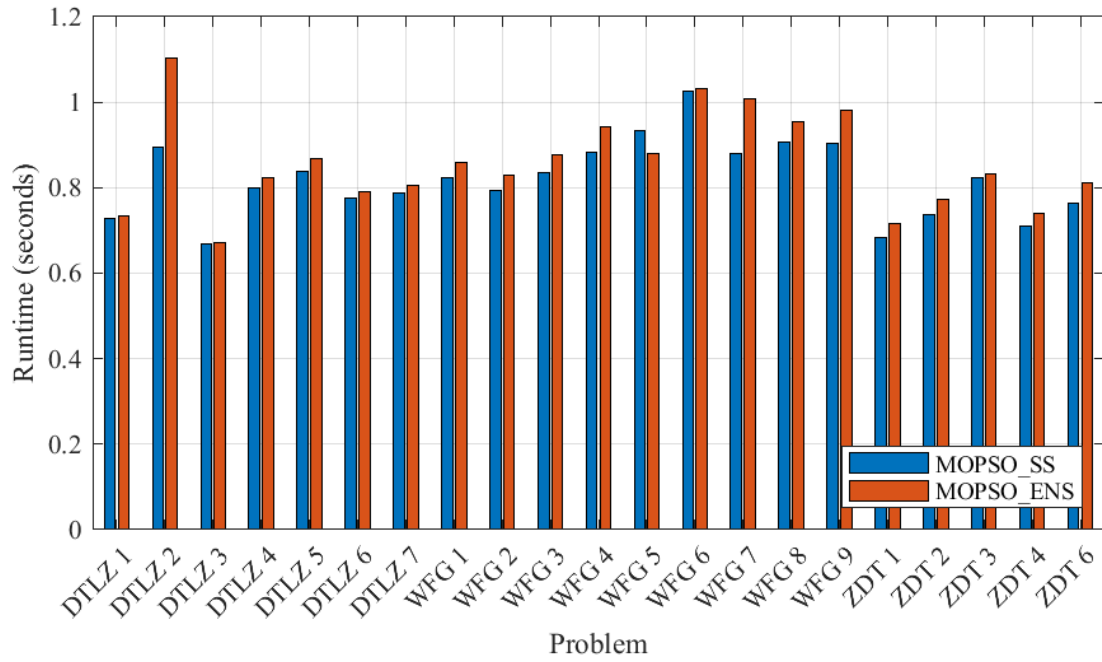


Figure 5.1: Runtimes of the MOPSO algorithm using SS and ENS as non-dominated sorting algorithms

5.2 Testing with the MOPSO algorithm

Table 5.1: Runtime results of the MOPSO algorithm using SS and ENS as non-dominated sorting algorithms

Problem	N	M	D	MOPSO_SS	MOPSO
DTLZ1	200	3	7	7.2918e-1 (6.73e-2) \approx	7.3429e-1 (1.33e-1)
DTLZ2	200	3	12	8.9556e-1 (6.65e-2) +	1.1044e+0 (1.39e-1)
DTLZ3	200	3	12	6.6917e-1 (4.26e-2) \approx	6.7095e-1 (3.56e-2)
DTLZ4	200	3	12	7.9903e-1 (4.31e-2) +	8.2299e-1 (6.44e-2)
DTLZ5	200	3	12	8.3745e-1 (3.61e-2) +	8.6829e-1 (5.52e-2)
DTLZ6	200	3	12	7.7678e-1 (3.56e-2) \approx	7.8948e-1 (4.52e-2)
DTLZ7	200	3	22	7.8655e-1 (6.88e-2) \approx	8.0443e-1 (9.36e-2)
WFG1	200	3	12	8.2198e-1 (5.19e-2) +	8.5886e-1 (8.08e-2)
WFG2	200	3	12	7.9350e-1 (3.05e-2) +	8.2890e-1 (7.49e-2)
WFG3	200	3	12	8.3564e-1 (3.42e-2) +	8.7686e-1 (7.35e-2)
WFG4	200	3	12	8.8353e-1 (3.30e-2) +	9.4340e-1 (1.01e-1)
WFG5	200	3	12	9.3212e-1 (1.59e-1) \approx	8.7977e-1 (6.59e-2)
WFG6	200	3	12	1.0262e+0 (9.75e-2) \approx	1.0328e+0 (8.67e-2)
WFG7	200	3	12	8.7963e-1 (3.14e-2) +	1.0078e+0 (1.82e-1)
WFG8	200	3	12	9.0708e-1 (2.90e-2) +	9.5347e-1 (4.53e-2)
WFG9	200	3	12	9.0446e-1 (3.18e-2) +	9.8007e-1 (9.09e-2)
ZDT1	200	2	30	6.8210e-1 (2.77e-2) +	7.1568e-1 (3.59e-2)
ZDT2	200	2	30	7.3634e-1 (4.60e-2) +	7.7386e-1 (4.67e-2)
ZDT3	200	2	30	8.2189e-1 (4.70e-2) +	8.3204e-1 (3.83e-2)
ZDT4	200	2	10	7.1047e-1 (2.82e-2) +	7.4119e-1 (3.13e-2)
ZDT6	200	2	10	7.6315e-1 (5.05e-2) +	8.1263e-1 (7.94e-2)
+ / - / \approx				15/0/6	

5.3 Testing with NSGA II

Subsection 2.5.2 explains how part of the NSGA II selection process for which solutions are to be included in the next generation is achieved through the calculation of a crowding distance indicator. A requirement to calculate the crowding distance indicator is to sort the solutions along each objective before calculating the distance between a solution and its nearest neighbours. This sorting activity has computational complexity $\mathcal{O}(N \log N)$ per objective. Since SS also requires sorting the population per objective as part of its non-dominated sorting methodology (which happens to be the most computationally expensive procedure in SS), the arrays of the sorted population per objective were saved so that NSGA II could capitalise on this step already having been performed. In contrast, the ENS implementation was unable to assist the NSGA II algorithm in this regard, as it does not require sorting for each objective in its methodology.

As mentioned in Subsection 5.1, the population size is set to 200. This means that non-dominated sorting would take place on 400 solutions, as the size of R_t is equal to twice that of P_t (see Figure 2.5).

Figure 5.2 and Table 5.2 present the runtime results for the implementations of SS and ENS in NSGA II. These results indicate that there is a clear advantage of using SS as the non-dominated sorting algorithm for both two and three dimensional problems. Table 5.2 further presents the results of the Wilcoxon rank-sum test, which indicate statistically significant improvements through the use of SS.

The mean hypervolume calculations for the optimisation results are provided in Table A.2 of Appendix A, and they indicate that there is no statistically significant difference in the quality of the results between the two non-dominated sorting algorithm implementations. The sharing of the arrays of the sorted population per objective mentioned above, proved to be beneficial for reducing the overall runtime of the NSGA II algorithm.

5.3 Testing with NSGA II

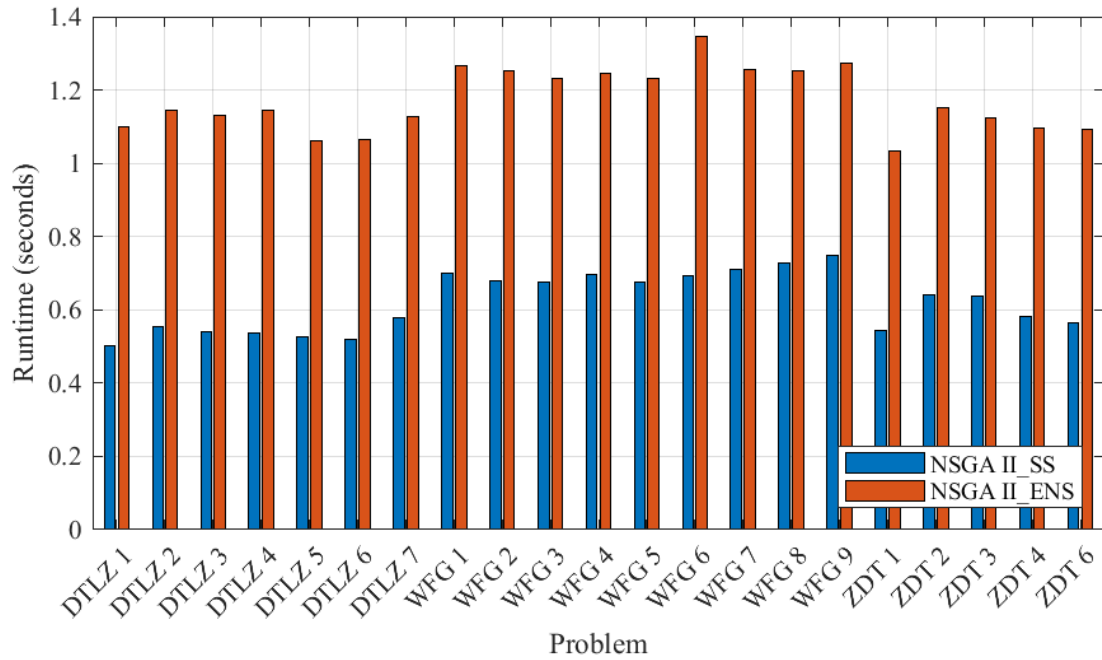


Figure 5.2: Runtimes of NSGA II using SS and ENS as non-dominated sorting algorithms

5.3 Testing with NSGA II

Table 5.2: Runtime results of NSGA II using SS and ENS as non-dominated sorting algorithms

Problem	N	M	D	NSGAILSS	NSGAI
DTLZ1	200	3	7	5.0265e-1 (6.04e-2) +	1.0995e+0 (1.12e-1)
DTLZ2	200	3	12	5.5322e-1 (7.77e-2) +	1.1436e+0 (1.52e-1)
DTLZ3	200	3	12	5.4072e-1 (5.87e-2) +	1.1301e+0 (1.03e-1)
DTLZ4	200	3	12	5.3634e-1 (6.31e-2) +	1.1446e+0 (1.42e-1)
DTLZ5	200	3	12	5.2583e-1 (6.05e-2) +	1.0630e+0 (1.20e-1)
DTLZ6	200	3	12	5.2044e-1 (5.91e-2) +	1.0632e+0 (1.21e-1)
DTLZ7	200	3	22	5.7689e-1 (6.17e-2) +	1.1277e+0 (1.27e-1)
WFG1	200	3	12	6.9935e-1 (4.55e-2) +	1.2678e+0 (1.23e-1)
WFG2	200	3	12	6.8030e-1 (4.50e-2) +	1.2531e+0 (1.03e-1)
WFG3	200	3	12	6.7728e-1 (4.00e-2) +	1.2316e+0 (1.02e-1)
WFG4	200	3	12	6.9712e-1 (4.60e-2) +	1.2453e+0 (8.46e-2)
WFG5	200	3	12	6.7546e-1 (4.23e-2) +	1.2298e+0 (7.44e-2)
WFG6	200	3	12	6.9273e-1 (6.01e-2) +	1.3462e+0 (1.19e-1)
WFG7	200	3	12	7.0882e-1 (4.36e-2) +	1.2544e+0 (8.08e-2)
WFG8	200	3	12	7.2635e-1 (5.91e-2) +	1.2540e+0 (9.43e-2)
WFG9	200	3	12	7.4908e-1 (4.25e-2) +	1.2740e+0 (9.47e-2)
ZDT1	200	2	30	5.4404e-1 (4.07e-2) +	1.0335e+0 (6.05e-2)
ZDT2	200	2	30	6.3974e-1 (4.20e-2) +	1.1502e+0 (5.25e-2)
ZDT3	200	2	30	6.3838e-1 (5.43e-2) +	1.1235e+0 (6.08e-2)
ZDT4	200	2	10	5.8146e-1 (4.05e-2) +	1.0968e+0 (5.84e-2)
ZDT6	200	2	10	5.6411e-1 (3.43e-2) +	1.0915e+0 (8.38e-2)
				+ / - / \approx	21/0/0

5.4 Testing with NSGA III

Since NSGA III is an adaptation of NSGA II, a population size of 200 also requires that non-dominated sorting to take place on 400 solutions. This is because the NSGA II procedure of creating an offspring population equal to the size of the parent population is also present in NSGA III.

The runtime results of the SS and ENS implementations within NSGA III are provided in Figure 5.3 and Table 5.3. They indicate that faster runtimes are achieved with the SS implementation on problems that have two or three objectives. Table 5.3 further indicates that the results of the Wilcoxon rank-sum test indicate statistically significant improvements through the use of SS. The mean hypervolume calculations for the optimisation results are provided in Appendix A in Table A.3, and they indicate that there is no statistically significant difference in the quality of the results of the two non-dominated sorting algorithm implementations. This implies that NSGA III with SS has achieved a comparable hypervolume to that of NSGA III with ENS, but in a shorter time.

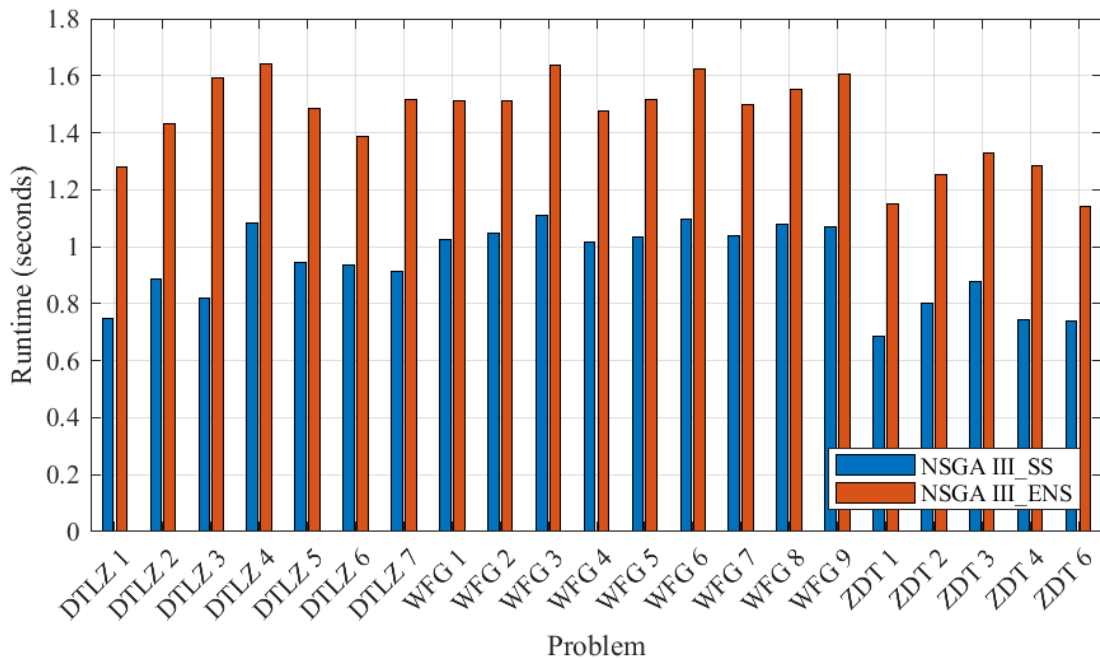


Figure 5.3: Runtimes of NSGA III using SS and ENS as non-dominated sorting algorithms

5.4 Testing with NSGA III

Table 5.3: Runtime results of NSGA III using SS and ENS as non-dominated sorting algorithms

Problem	N	M	D	NSGAIII _{SS}	NSGAIII _{ENS}
DTLZ1	200	3	7	7.4764e-1 (5.23e-2) +	1.2790e+0 (1.40e-1)
DTLZ2	200	3	12	8.8507e-1 (8.70e-2) +	1.4294e+0 (2.21e-1)
DTLZ3	200	3	12	8.1958e-1 (6.98e-2) +	1.5916e+0 (2.93e-1)
DTLZ4	200	3	12	1.0846e+0 (1.16e-1) +	1.6405e+0 (1.48e-1)
DTLZ5	200	3	12	9.4316e-1 (6.44e-2) +	1.4830e+0 (1.63e-1)
DTLZ6	200	3	12	9.3730e-1 (8.05e-2) +	1.3856e+0 (1.55e-1)
DTLZ7	200	3	22	9.1485e-1 (6.07e-2) +	1.5169e+0 (1.94e-1)
WFG1	200	3	12	1.0229e+0 (7.14e-2) +	1.5133e+0 (7.99e-2)
WFG2	200	3	12	1.0453e+0 (6.49e-2) +	1.5102e+0 (8.92e-2)
WFG3	200	3	12	1.1079e+0 (4.98e-2) +	1.6373e+0 (9.15e-2)
WFG4	200	3	12	1.0161e+0 (6.45e-2) +	1.4756e+0 (7.50e-2)
WFG5	200	3	12	1.0362e+0 (6.35e-2) +	1.5144e+0 (7.53e-2)
WFG6	200	3	12	1.0986e+0 (1.55e-1) +	1.6228e+0 (1.48e-1)
WFG7	200	3	12	1.0400e+0 (7.26e-2) +	1.4995e+0 (8.25e-2)
WFG8	200	3	12	1.0801e+0 (9.43e-2) +	1.5524e+0 (7.49e-2)
WFG9	200	3	12	1.0683e+0 (6.18e-2) +	1.6051e+0 (7.62e-2)
ZDT1	200	2	30	6.8513e-1 (3.99e-2) +	1.1482e+0 (6.20e-2)
ZDT2	200	2	30	8.0301e-1 (5.13e-2) +	1.2507e+0 (6.06e-2)
ZDT3	200	2	30	8.7783e-1 (6.84e-2) +	1.3276e+0 (7.19e-2)
ZDT4	200	2	10	7.4496e-1 (6.42e-2) +	1.2862e+0 (7.81e-2)
ZDT6	200	2	10	7.4047e-1 (4.53e-2) +	1.1431e+0 (4.91e-2)
				+ / - / \approx	21/0/0

5.5 Testing with the AGE-MOEA

Subsection 2.5.4 explained that AGE-MOEA is an adaptation of NSGA II. However, AGE-MOEA differs in that it replaces the crowding distance indicator with a survival score metric which is calculated based on the diversity and proximity of the non-dominated sets.

The runtime results of AGE-MOEA are presented in Figure 5.4 and Table 5.4. It is clear that there is little difference between the runtimes of the two non-dominated sorting algorithm implementations. The results of the Wilcoxon rank-sum test are presented in Table 5.4, and they indicate that there is no statistically significant difference for 10 out of the 21 problems. Where differences are reported, the general pattern indicates that the SS implementation is more suited for two-dimensional problems, as the SS implementation achieved statistically significantly faster runtimes on the ZDT 2 and ZDT 4 problems. On the other hand, all the cases where the ENS implementation achieved statistically significantly faster results were on three-dimensional problems. Only the WFG 2 and WFG 4 problems were solved faster with the SS implementation.

As reported by the Wilcoxon rank-sum test in Table A.4 of Appendix A, the quality of the results supplied by both non-dominated sorting algorithm implementations were even.

5.5 Testing with the AGE-MOEA

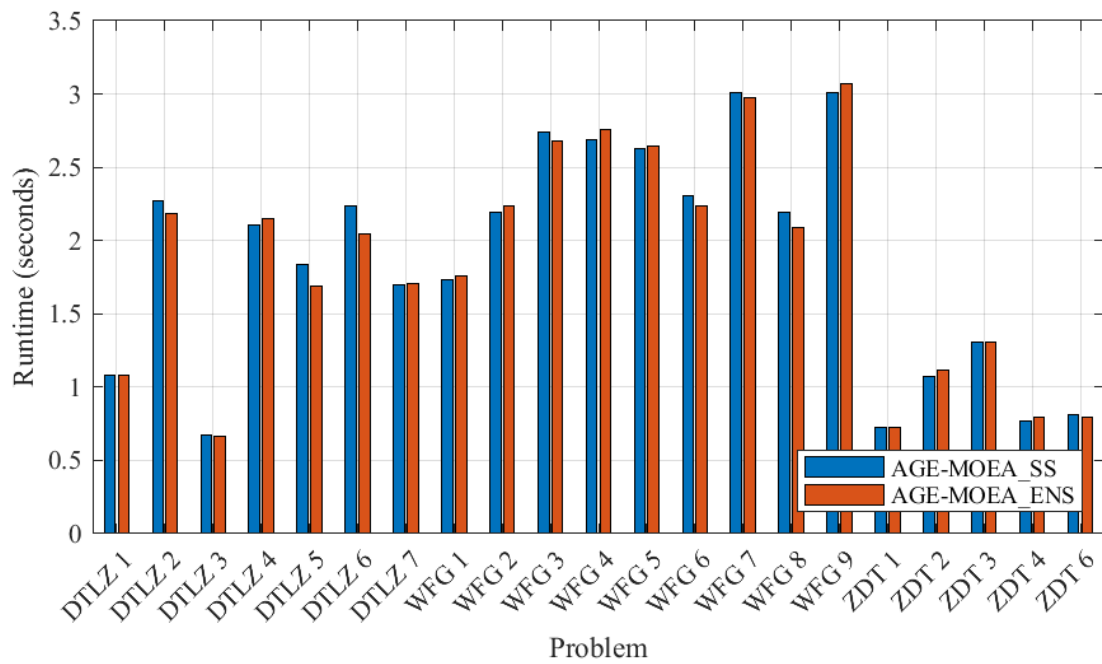


Figure 5.4: Runtimes of AGE-MOEA using SS and ENS as non-dominated sorting algorithms

5.5 Testing with the AGE-MOEA

Table 5.4: Runtime results of AGE-MOEA using SS and ENS as non-dominated sorting algorithms

Problem	N	M	D	AGEMOEA_SS	AGEMOEA_ENS
DTLZ1	200	3	7	1.0806e+0 (1.34e-1) \approx	1.0853e+0 (1.45e-1)
DTLZ2	200	3	12	2.2745e+0 (1.15e-1) $-$	2.1827e+0 (1.70e-1)
DTLZ3	200	3	12	6.7356e-1 (8.09e-2) \approx	6.6810e-1 (1.20e-1)
DTLZ4	200	3	12	2.1053e+0 (2.32e-1) \approx	2.1473e+0 (2.76e-1)
DTLZ5	200	3	12	1.8365e+0 (2.21e-1) $-$	1.6930e+0 (1.65e-1)
DTLZ6	200	3	12	2.2349e+0 (2.23e-1) $-$	2.0451e+0 (2.74e-1)
DTLZ7	200	3	22	1.6989e+0 (1.42e-1) \approx	1.7060e+0 (1.80e-1)
WFG1	200	3	12	1.7302e+0 (1.14e-1) \approx	1.7572e+0 (1.39e-1)
WFG2	200	3	12	2.1971e+0 (9.91e-2) $+$	2.2328e+0 (3.02e-1)
WFG3	200	3	12	2.7439e+0 (1.04e-1) $-$	2.6759e+0 (1.48e-1)
WFG4	200	3	12	2.6839e+0 (8.17e-2) $+$	2.7583e+0 (1.66e-1)
WFG5	200	3	12	2.6308e+0 (1.19e-1) \approx	2.6457e+0 (1.16e-1)
WFG6	200	3	12	2.3082e+0 (1.20e-1) $-$	2.2380e+0 (1.06e-1)
WFG7	200	3	12	3.0132e+0 (1.07e-1) $-$	2.9747e+0 (1.43e-1)
WFG8	200	3	12	2.1938e+0 (8.40e-2) $-$	2.0876e+0 (1.02e-1)
WFG9	200	3	12	3.0118e+0 (1.80e-1) \approx	3.0706e+0 (1.91e-1)
ZDT1	200	2	30	7.2648e-1 (6.67e-2) \approx	7.2663e-1 (5.78e-2)
ZDT2	200	2	30	1.0765e+0 (7.32e-2) $+$	1.1188e+0 (1.20e-1)
ZDT3	200	2	30	1.3093e+0 (9.69e-2) \approx	1.3096e+0 (8.76e-2)
ZDT4	200	2	10	7.6827e-1 (7.71e-2) $+$	7.9287e-1 (7.20e-2)
ZDT6	200	2	10	8.0826e-1 (7.11e-2) \approx	7.9298e-1 (7.48e-2)
$+ / - / \approx$				4/7/10	

5.6 Testing with the A-NSGA III

As explained in Subsection 2.5.5, A-NSGA III is also an adaptation of NSGA II, but differs in its selection of solutions from the current generation to be added to the next generation.

The runtime results of the SS and ENS implementations within A-NSGA III are presented in Figure 5.5 and Table 5.5. These results indicate that there is a marginal advantage when implementing SS as the non-dominated sorting algorithm for all the tested problems. Table A.5 of Appendix A presents the hypervolumes of the optimisations performed by A-NSGA III for both the SS and ENS implementations. The Wilcoxon rank-sum test performed on these hypervolume values indicate that there is no statistically significant difference between the quality of the results supplied by the two non-dominated sorting implementations.

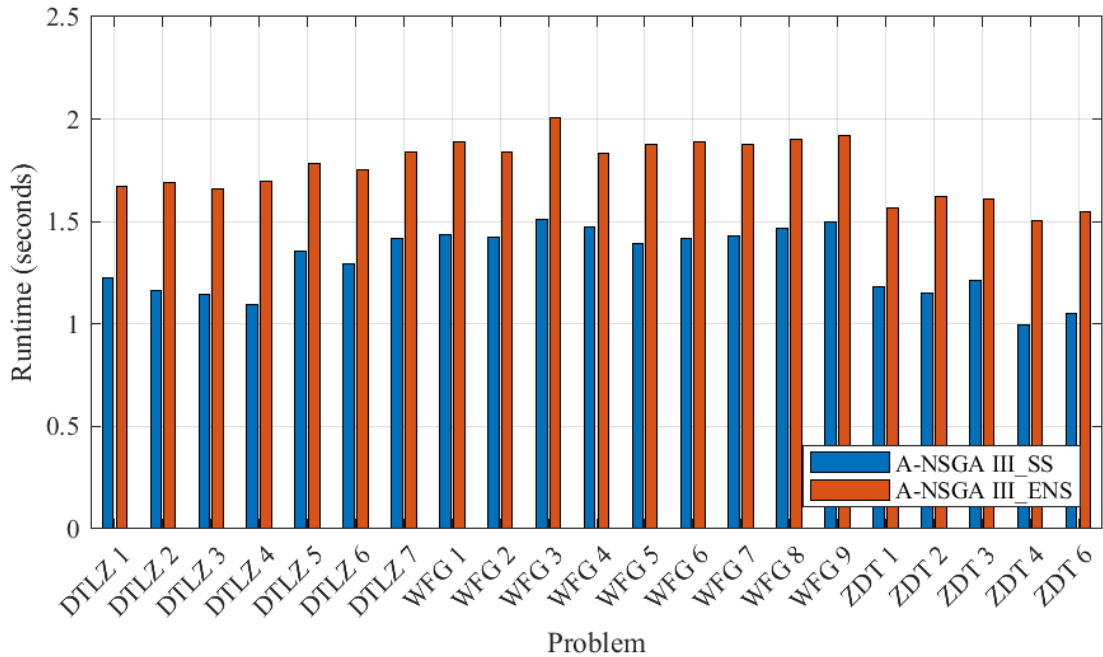


Figure 5.5: Runtimes of A-NSGA III using SS and ENS as non-dominated sorting algorithms

5.6 Testing with the A-NSGA III

Table 5.5: Runtime results of A-NSGA III using SS and ENS as non-dominated sorting algorithms

Problem	N	M	D	ANSGAIII _{SS}	ANSGAIII _{ENS}
DTLZ1	200	3	7	1.2258e+0 (8.91e-2) +	1.6699e+0 (1.48e-1)
DTLZ2	200	3	12	1.1637e+0 (1.25e-1) +	1.6932e+0 (2.33e-1)
DTLZ3	200	3	12	1.1443e+0 (9.78e-2) +	1.6603e+0 (1.44e-1)
DTLZ4	200	3	12	1.0970e+0 (1.30e-1) +	1.6957e+0 (2.05e-1)
DTLZ5	200	3	12	1.3549e+0 (7.67e-2) +	1.7844e+0 (1.47e-1)
DTLZ6	200	3	12	1.2935e+0 (7.87e-2) +	1.7504e+0 (1.64e-1)
DTLZ7	200	3	22	1.4195e+0 (1.05e-1) +	1.8415e+0 (2.38e-1)
WFG1	200	3	12	1.4343e+0 (7.45e-2) +	1.8878e+0 (9.10e-2)
WFG2	200	3	12	1.4241e+0 (7.92e-2) +	1.8423e+0 (8.67e-2)
WFG3	200	3	12	1.5084e+0 (5.81e-2) +	2.0094e+0 (7.72e-2)
WFG4	200	3	12	1.4761e+0 (9.50e-2) +	1.8328e+0 (6.90e-2)
WFG5	200	3	12	1.3913e+0 (6.72e-2) +	1.8736e+0 (8.30e-2)
WFG6	200	3	12	1.4161e+0 (7.37e-2) +	1.8873e+0 (7.38e-2)
WFG7	200	3	12	1.4315e+0 (7.23e-2) +	1.8783e+0 (8.67e-2)
WFG8	200	3	12	1.4642e+0 (7.64e-2) +	1.9025e+0 (8.83e-2)
WFG9	200	3	12	1.4982e+0 (6.73e-2) +	1.9219e+0 (7.22e-2)
ZDT1	200	2	30	1.1829e+0 (5.60e-2) +	1.5656e+0 (5.58e-2)
ZDT2	200	2	30	1.1489e+0 (5.41e-2) +	1.6194e+0 (6.41e-2)
ZDT3	200	2	30	1.2124e+0 (7.17e-2) +	1.6103e+0 (6.52e-2)
ZDT4	200	2	10	9.9868e-1 (8.18e-2) +	1.5069e+0 (8.98e-2)
ZDT6	200	2	10	1.0490e+0 (5.69e-2) +	1.5476e+0 (2.14e-1)
				+ / - / \approx	21/0/0

5.7 Discussion of results

The first of the multi-objective evolutionary algorithms to be tested was the MOPSO algorithm. The results concluded that, although there was a decrease in runtime when using SS instead of ENS, the difference was minimal. The Wilcoxon rank-sum test confirmed that there were no statistically significant differences between the runtimes of six of the 21 problems. Since ENS proved to be the superior algorithm for problems with more than three objectives, a pragmatic recommendation is to use the ENS with the MOPSO algorithm, as the flexibility offered by ENS to solve higher dimensional problems is arguably more beneficial than the marginal decrease in runtime offered by SS on two and three objective problems.

The second multi-objective optimisation algorithm to undergo testing was NSGA II. The results indicate a clear advantage when using SS, as the runtimes of the SS implementation were significantly lower. For completeness, the Wilcoxon rank-sum test was performed and confirmed that the SS implementation was statistically significantly faster. The requirement of sorting the population along each of its objectives is shared by both SS when performing non-dominated sorting and NSGA II to calculate the crowding distance indicator. This allows for this computationally expensive calculation to only have to be performed once. In comparison, ENS, which achieves very similar non-dominated sorting times to SS, as reported in Chapter 4, still requires NSGA II to sort the population along each objective after ENS has provided the ranked solutions. Therefore, if two and three dimensional problems are to be optimised by NSGA II, it is recommended that SS be used as the non-dominated sorting algorithm.

The third multi-objective optimisation algorithm to be tested was NSGA III. The results also indicate that SS provides a statistically significant advantage as the runtimes for all the problems were lower with the SS implementation. The reduction in runtime when using SS, whilst substantial, is less than that achieved in NSGA II. This is likely to be due to the fact that NSGA III employs an alternative method for selecting which solutions of the current population should be included in the next generation. Furthermore, NSGA III was ultimately developed as a many-objective optimisation algorithm. This is evident in the overall

5.7 Discussion of results

longer runtimes achieved by NSGA III when compared to NSGA II for both the SS and ENS implementations. A conclusion may therefore be drawn that, for lower dimensional problems, the crowding distance indicator used by NSGA II is a more efficient method to determine which of the current solutions should be used in the next generation than the reference point method employed by NSGA III. Therefore, it is recommended that NSGA II be used instead of NSGA III when optimising lower objective problems. Should NSGA III be employed, SS offers reduced overall runtimes in lower objective settings. The testing of NSGA III using SS and ENS to optimise higher dimensional problems is recommended as future work.

The fourth multi-objective evolutionary algorithm to undergo testing was AGE-MOEA. Like NSGA III, this algorithm is also an adaptation of NSGA II. The runtime results of this algorithm were very similar for SS and ENS. The difference between the runtimes of the SS and ENS implementations on nine out of the 21 runtime results were calculated to be statistically insignificant. Of the remaining results, eight problems achieved faster runtimes with the ENS implementation, while the remaining four problems were optimised faster with the SS implementation. AGE-MOEA, like NSGA III, was also intended to solve many-objective problems. In lower dimensional settings, calculation of the geometry of the first non-dominated set appears to be suboptimal compared to the crowding distance indicator of NSGA II. Therefore, like NSGA III, it is recommended that the AGE-MOEA algorithm be used for the optimisation of many-objective problems in conjunction with the ENS non-dominated sorting algorithm.

The final algorithm to undergo testing was A-NSGA III. The runtime results are similar to those achieved by NSGA III, albeit slightly slower. This is not surprising as both of these algorithms were based on the NSGA II algorithm and both employ a similar reference point system to determine which solutions from the current population should be selected for inclusion in the next generation. The runtime results indicate that A-NSGA III is able to achieve faster runtimes when optimising two and three objective problems with SS as the non-dominated sorting algorithm. However, as an algorithm that was developed for the optimisation of many-objective problems, it is recommended that further testing of SS

and ENS implementations in higher dimensional settings be considered in future work.

5.8 Summary: Chapter 5

Presented in this chapter were the runtime results of the five multi-objective algorithms presented in Subsection 2.5 with implementations of SS and ENS solving the test suite problems presented in Subsection 2.7. The Wilcoxon rank-sum test was then performed on the reported runtimes to investigate whether there were statistically significant differences or not between the SS and ENS implementations.

The quality of the optimisation results of both the SS and ENS implementations of each of the multi-objective optimisation algorithms was calculated by means of the hypervolume indicator. Since 100 replications of each test were conducted, the mean hypervolume value was calculated and reported in Appendix A. Furthermore the Wilcoxon rank-sum test was performed on all the hypervolumes to investigate whether or not there were any statistically significant differences between the results provided by the different non-dominated sorting algorithm implementations, and no differences were found.

A discussion of the results was then provided, including recommendations for the use of these multi-objective optimisation and non-dominated sorting algorithms.

This concludes the experimentation performed with SS in this research. The following chapter presents the final conclusions of the overall research project.

Chapter 6

Research summary and conclusions

This chapter provides a summary of the work conducted in this thesis, as well as a presentation of the final research conclusions. Also included is a reflection on which items could be refined and improved in future work.

6.1 Project summary and conclusions

This study identified the requirement of non-dominated sorting as a potential research area in the field of multi-objective optimisation using evolutionary algorithms. The research assignment was formally defined in Subsection 1.2. This research set out to design a novel non-dominated sorting algorithm for use in multi-objective evolutionary algorithms that would reduce the overall runtime during multi-objective optimisation.

After understanding the approaches taken by existing methodologies found in literature, an attempt to approach the problem from a different angle was made by trying to reduce the number of solution-to-solution comparisons. The idea was to investigate if simply ordering the solutions in a specific manner, and then finding a way to utilise the knowledge gained from the solutions' relative positions to one another to determine their dominance relationships and thereby perform non-dominated sorting could be achieved.

6.1 Project summary and conclusions

To ensure a reliable and fundamental understanding of the problem and its environment, a study of the literature pertaining to various aspects of the field was conducted in Chapter 2. This literature study included existing non-dominated sorting algorithms, some of the recent multi-objective evolutionary algorithms and test suites created to quantify the performance of these algorithms.

An effort was made to create a competitive non-dominated sorting algorithm that was able to perform non-dominated sorting on populations with any number of objectives. The development of Shadow Sort (SS) as a novel non-dominated sorting algorithm was presented in Chapter 3. The final MATLAB code of SS is made available in Appendix B.

Having developed SS as a functional non-dominated sorting algorithm, the process and documentation of testing it against existing non-dominated sorting algorithms in a controlled manner were presented in Chapter 4. The environment was considered controlled because, although the tests utilised completely random data, the same random data were provided to all the non-dominated algorithms undergoing testing. Since non-dominated sorting is an “*exact science*,” given the same input data, the outputs of all non-dominated sorting algorithms should be identical. Two types of tests were conducted and presented in Chapter 4, namely entire population sorting and partial population sorting. The purpose of the first test was to gain insights regarding the efficiency of the various techniques adopted by different non-dominated sorting algorithms as described in Subsection 2.6.4. The second test was performed as a practical test to understand how the non-dominated sorting algorithms would perform the tasks required by most multi-objective evolutionary algorithms, namely to perform non-dominated sorting on a selected number of solutions.

The results obtained from the tests conducted in Chapter 4, specifically those from Test 2, guided the focus of tests performed in Chapter 5. It became clear that the strategy employed by SS was most suitable in lower dimensional settings – performing non-dominated sorting on populations with two or three objectives. The five multi-objective evolutionary algorithms reviewed in Chapter 2 were modified to utilise SS and its closest competitor ENS for non-dominated sorting. These two implementations of each multi-objective evolutionary algorithm were then used to optimise the test suite problems also reviewed in Chapter 2. Since the

6.2 Future research

results of Chapter 4 indicated that SS was competitive primarily on two and three objective problems, the tests conducted in Chapter 5 were limited to these dimensions. The results of Chapter 5 indicated that SS offered a statistically significant improvement in NSGA II, NSGA III and A-NSGA III. It is recommended that SS be considered for non-dominated sorting when performing optimisation using these algorithms for two and three objective problems.

In summary, the research assignment formalised in Chapter 1 was achieved because:

1. SS is a novel non-dominated sorting algorithm.
2. SS managed to achieve faster times for non-dominated sorting of random datasets under certain circumstances, namely lower-dimensional populations. SS can contribute to future research on decomposition of problems with many objectives.
3. The overall runtime of some multi-objective evolutionary algorithms was reduced.

The summary and conclusions of this work led to the suggestions for further research as presented in the next section.

6.2 Future research

The research and work presented in this study cannot be considered complete. Although there are many potential avenues to explore to improve the study, the following suggestions are highlighted:

1. Conduct more comprehensive testing similar to that presented in Chapter 5, but with a focus on higher dimensional problems.
2. Extend the work performed by SS to assist with the next step required of most multi-objective evolutionary algorithms, which is to calculate a metric used to decide which solutions of the current generation should be included in the next generation.

6.3 Appraisal of research

3. Investigate the use of sparse matrices for performing non-dominated sorting. This topic was investigated informally as part of this research, but a decision was made to abandon the idea due to an insufficient knowledge of data structures used in computing.
4. SS may be compared to other non-dominated sorting algorithms such as Corner Sort (CS) and Merge Non-dominated Sort (MNDS).
5. Evaluate the performance of SS with multi-objective optimisation algorithms other than evolutionary algorithms.
6. Implement SS in a language more suitable to efficient, high-speed programming such as C or Java.

6.3 Appraisal of research

The work undertaken to create this thesis provided an opportunity for the researcher to gain deep insight into the field of multi-objective optimisation and non-dominated sorting. The process of developing a new non-dominated sorting algorithm provided many hours of problem solving and abstract thinking. The testing of many different ideas and techniques brought moments of disappointment and frustration, but also of excitement and achievement. The iteration of SS which underwent testing in this research had already undergone many revisions and improvements. To compete with some of the latest and fastest non-dominated sorting algorithms in the field of multi-objective optimisation proved to be a significant challenge. The researcher is confident that sufficient evidence has been provided to motivate the value to be gained through the implementation of SS in the correct environment. Although the original intent was to provide an improved non-dominated sorting algorithm for use in all multi-objective evolutionary algorithms, the complexity of the field limited a meaningful contribution to lower dimensional problems. This is still a useful result, as many engineering optimisation problems with more than one objective can be reduced to bi-objective or tri-objective problems.

6.4 Concluding remarks

The field of optimisation is often associated with trying to find the best answer to a problem. Relatively speaking, this is a simpler task for single-objective optimisation. However, as soon as one enters the realm of multi-objective optimisation, it becomes clear that it is often impossible to optimise the outcome for every requirement simultaneously. Therefore, the goal changes from trying to find the best solution for a particular objective, to finding a balance which, through some trade-off, adequately satisfies the whole system.

In the busy and bustling environment called life, people often have their focus so finely set on one specific goal. However, whilst one must never stop striving to achieve their goals, it is important to occasionally stop and understand the trade-offs being made to facilitate their pursuit. Although there is no single correct answer to the priorities one should establish in their life, there is a balance to be found, and it is this balance which one should ultimately strive to optimise.

Bibliography

- BEKKER, J. (2012). Applying the cross-entropy method in multi-objective optimisation of dynamic stochastic systems. 1–243. [87](#)
- BELLMAN, R. (1958). Dynamic programming and stochastic control processes. *Information and Control*, **1**, 228–239. [73](#)
- BENTLEY, J.L., KUNG, H.T., SCHKOLNICK, M. & THOMPSON, C.D. (1978). On the Average Number of Maxima in a Set of Vectors and Applications. *Journal of the ACM (JACM)*, **25**, 536–543. [95](#)
- CHENG, Q., DU, B., ZHANG, L. & LIU, R. (2019). ANSGA-III: A Multi-objective Endmember Extraction Algorithm for Hyperspectral Images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, **12**, 700–721. [27](#), [28](#), [29](#)
- COELLO, C.A.C., LAMONT, G.B., VELDHUIZEN, D.A.V., GOLDBERG, D.E. & KOZA, J.R. (2007). *Evolutionary Algorithms for Solving Multi-Objective Problems Second Edition Genetic and Evolutionary Computation Series Series Editors Selected titles from this series* ∴ Springer. [16](#), [17](#), [50](#), [51](#), [52](#), [53](#), [62](#)
- COELLO COELLO, C.A. & LECHUGA, M.S. (2002). MOPSO: A proposal for multiple objective particle swarm optimization. *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, **2**, 1051–1056. [19](#), [100](#)
- COELLO COELLO, C.A., GONZÁLEZ BRAMBILA, S., FIGUEROA GAMBOA, J., CASTILLO TAPIA, M.G. & HERNÁNDEZ GÓMEZ, R. (2019). Evolutionary multiobjective optimization: open research areas and some challenges lying ahead. *Complex & Intelligent Systems*. [95](#)

BIBLIOGRAPHY

- DEB, K. & JAIN, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, Part I: Solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, **18**, 577–601. [22](#), [23](#), [24](#), [25](#)
- DEB, K., PRATAP, A., AGARWAL, S. & MEYARIVAN, T. (2002). A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Transactions on Evolutionary Computation*, **6**, 182–197. [19](#), [20](#), [21](#), [23](#), [31](#), [87](#)
- DEB, K., THIELE, L., LAUMANN, M. & ZITZLER, E. (2005). Scalable Test Problems for Evolutionary Multiobjective Optimization. *Evolutionary Multiobjective Optimization*, 105–145. [47](#)
- DENNIS, J.E. (1998). Normal-Boundary Intersection: A new method for generating the Pareto surface in nonlinear multi-criteria optimization problems. *SIAM J. OPTIM.*, **8**, 631–657. [23](#)
- GARCIA, S. & TRINH, C.T. (2019). Comparison of multi-objective evolutionary algorithms to solve the modular cell design problem for novel biocatalysis. *Processes*, **7**. [20](#)
- HEPPNER, F.H. & GREANDER, U. (1990). A Stochastic Non-Linear Model for Bird Flocking. *The Ubiquity of Chaos*, 233–238. [17](#)
- HOARE, C.A.R. (1961). Quicksort. *Communications of the ACM*, **4**, 321. [69](#), [77](#)
- HUBAND, S., HINGSTON, P., BARONE, L. & WHILE, L. (2006). A review of multiobjective test problems and a scalable test problem toolkit. *IEEE Transactions on Evolutionary Computation*, **10**, 477–506. [8](#), [9](#), [10](#), [11](#), [53](#), [54](#), [55](#), [56](#), [57](#), [58](#), [59](#), [60](#), [63](#)
- KENNEDY, J. & EBERHART, R. (1995). Particle swarm optimization. *ICNN'95-international conference on neural networks*, 1942–1948. [19](#)
- KNOWLES, J.D. & CORNE, D.W. (2000). Approximating the nondominated front using the Pareto Archived Evolution Strategy. *Evolutionary computation*, **8**, 149–172. [19](#), [20](#)

BIBLIOGRAPHY

- MARTÍNEZ, S.Z., SOSA HERNÁNDEZ, V.A., AGUIRRE, H., TANAKA, K. & COELLO COELLO, C.A. (2014). Using a family of curves to approximate the pareto front of a multi-objective optimization problem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **8672**, 682–691. [26](#)
- MCCLYMONT, K. & KEEDWELL, E. (2012). Deductive sort and climbing sort: New methods for non-dominated sorting. *Evolutionary Computation*, **20**, 1–26. [31](#), [32](#), [39](#), [79](#), [87](#), [96](#)
- MISHRA, K.K. & HARIT, S. (2010). A Fast Algorithm for Finding the Non Dominated Set in Multi objective Optimization. *International Journal of Computer Applications*, **1**, 35–39. [30](#)
- MISHRA, S., SAHA, S. & MONDAL, S. (2018). MBOS: Modified Best Order Sort Algorithm for Performing Non-Dominated Sorting. *2018 IEEE Congress on Evolutionary Computation, CEC 2018 - Proceedings*, 1–8. [42](#)
- MONTGOMERY, D.C. & RUNGER, G.C. (1994). *Applied Statistics and Probability for Engineers*, vol. 19. Wiley. [14](#)
- MORENO, J., RODRIGUEZ, D., ANTONIO NEBRO, C. & LOZANO, J.A. (2018). Merge non-dominated sorting algorithm for many-objective optimization. *arXiv*, 1–13. [46](#)
- MOSTAGHIM, S. & SCHMECK, H. (2008). Distance based ranking in many-objective particle swarm optimization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, **5199 LNCS**, 753–762. [95](#)
- PANICHELLA, A. (2019). An adaptive evolutionary algorithm based on non-euclidean geometry for many-objective optimization. *GECCO 2019 - Proceedings of the 2019 Genetic and Evolutionary Computation Conference*, 595–603. [26](#)
- REDDY, M.J. & KUMAR, D.N. (2006). Multi-Objective Optimization using Evolutionary Algorithms. *Water Resources Management*, **20**, 861–878. [23](#)

BIBLIOGRAPHY

- REYNOLDS, C.W. (1987). Flocks, Herds and Schools: A distributed behavioural model. *Computer Graphics*, **21**, 25–34. [17](#)
- ROY, P.C., ISLAM, M.M. & DEB, K. (2016). Best order sort: A new algorithm to non-dominated sorting for evolutionary multi-objective optimization. *GECCO 2016 Companion - Proceedings of the 2016 Genetic and Evolutionary Computation Conference*, 1113–1120. [31](#), [42](#), [79](#), [88](#), [96](#)
- SEAR, R., LAWSON, D.W. & DICKINS, T.E. (2007). Synthesis in the human evolutionary behavioural sciences. *Journal of Evolutionary Psychology*, **5**, 3–28. [18](#)
- SHENG, L.K., IBRAHIM, Z., BUYAMIN, S., AHMAD, A., TUMARI, M.Z.M., JUSOF, M.F.M. & AZIZ, N.A.A. (2014). Multi-Objective particle swarm optimization algorithms – A leader selection overview. *International Journal of Simulation: Systems, Science and Technology*, **15**, 6–19. [9](#)
- SHULTZ, T.R. & FAHLMAN, S.E. (2017). *Encyclopedia of Machine Learning and Data Mining*. Springer. [73](#), [95](#)
- TIAN, Y., CHENG, R., ZHANG, X. & JIN, Y. (2017). PlatEMO: A MATLAB Platform for Evolutionary Multi-Objective Optimization [Educational Forum]. *IEEE Computational Intelligence Magazine*, **12**, 73–87. [98](#)
- TONG, Q., XUE, Y. & ZHANG, L. (2014). Progress in hyperspectral remote sensing science and technology in China over the past three decades. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, **7**, 70–91. [27](#)
- UNIVERSITY OF COLORADO BOULDER (2016). The Engineering Design Process. [5](#)
- WALPOLE, R.E. & MYERS, R.H. (1993). *Probability and Statistics for Engineers and Scientists*. MacMillan Publishing Company, New York, 5th edn. [15](#)
- WANG, H. & YAO, X. (2014). Corner sort for Pareto-based many-objective optimization. *IEEE Transactions on Cybernetics*, **44**, 92–102. [8](#), [45](#), [46](#)

BIBLIOGRAPHY

- XU, M., ZHANG, L. & DU, B. (2015). An Image-Based Endmember Bundle Extraction Algorithm Using Both Spatial and Spectral Information. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, **8**, 2607–2617. [27](#)
- ZHANG, X., TIAN, Y., CHENG, R. & JIN, Y. (2015). An Efficient Approach to Non-dominated Sorting for Evolutionary Multi-objective Optimization. *IEEE Transactions on Evolutionary Computation*, **19**, 1–15. [31](#), [35](#), [36](#), [37](#), [38](#), [39](#), [40](#), [79](#), [96](#)
- ZITZLER, E. & THIELE, L. (1999). Multiobjective evolutionary algorithms: A comparative case study and the strength Pareto approach. *IEEE Transactions on Evolutionary Computation*, **3**, 257–271. [20](#), [23](#)
- ZITZLER, E., DEB, K. & THIELE, L. (2000). Comparison of multiobjective evolutionary algorithms: empirical results. *Evolutionary computation*, **8**, 173–195. [60](#)

Appendix A

Algorithm

hypervolume-per-volume results: ENS vs Shadow Sort

The choice of non-dominated sorting implemented in a multi-objective evolutionary algorithm should have no impact on the quality of the final solution provided. After all, given input data, all non-dominated sorting algorithms should provide the same result- there is no randomness in a non-dominated sorting procedure. However, to ensure that the multi-objective evolutionary algorithms with the ENS and Shadow Sort implementations did indeed provide equivalent quality in terms of solutions, the hypervolumes of the final solutions were calculated for each repetition and their means reported in Tables A.1 to A.5. The values in parentheses report the standard deviation of the hypervolumes for the 100 repetitions.

Table A.1: Hypervolume results of MOPSO using ENS and SS as non-dominated sorting algorithms

Problem	N	M	D	MOPSO_SS	MOPSO_ENS
DTLZ1	200	3	7	0.0000e+0 (0.00e+0) \approx	0.0000e+0 (0.00e+0)
DTLZ2	200	3	12	4.4188e-1 (2.09e-2) \approx	4.4088e-1 (2.23e-2)
DTLZ3	200	3	12	0.0000e+0 (0.00e+0) \approx	0.0000e+0 (0.00e+0)
DTLZ4	200	3	12	2.8800e-1 (9.84e-2) \approx	2.6464e-1 (9.56e-2)
DTLZ5	200	3	12	1.8677e-1 (5.35e-3) \approx	1.8631e-1 (5.15e-3)
DTLZ6	200	3	12	7.9955e-4 (8.00e-3) \approx	0.0000e+0 (0.00e+0)
DTLZ7	200	3	22	9.3345e-4 (8.01e-3) \approx	0.0000e+0 (0.00e+0)
WFG1	200	3	12	1.3860e-1 (5.04e-2) \approx	1.3130e-1 (4.14e-2)
WFG2	200	3	12	8.0589e-1 (1.93e-2) \approx	8.0429e-1 (1.80e-2)
WFG3	200	3	12	2.1364e-1 (3.34e-2) \approx	2.1006e-1 (3.49e-2)
WFG4	200	3	12	4.0071e-1 (1.64e-2) \approx	4.0205e-1 (1.54e-2)
WFG5	200	3	12	3.7937e-1 (3.66e-2) \approx	3.7498e-1 (3.76e-2)
WFG6	200	3	12	3.5257e-1 (2.18e-2) \approx	3.5448e-1 (2.01e-2)
WFG7	200	3	12	3.6648e-1 (2.08e-2) \approx	3.6996e-1 (2.19e-2)
WFG8	200	3	12	2.9105e-1 (1.63e-2) \approx	2.9117e-1 (1.57e-2)
WFG9	200	3	12	4.2116e-1 (1.68e-2) \approx	4.1816e-1 (1.59e-2)
ZDT1	200	2	30	1.0746e-1 (1.16e-1) \approx	1.0333e-1 (9.85e-2)
ZDT2	200	2	30	3.3509e-4 (3.35e-3) \approx	0.0000e+0 (0.00e+0)
ZDT3	200	2	30	2.5833e-2 (4.05e-2) \approx	2.5017e-2 (5.00e-2)
ZDT4	200	2	10	0.0000e+0 (0.00e+0) \approx	0.0000e+0 (0.00e+0)
ZDT6	200	2	10	2.3962e-1 (1.76e-1) \approx	2.3633e-1 (1.75e-1)
+ / - / \approx				0/0/21	

Table A.2: Hypervolume results of NSGA II using ENS and SS as non-dominated sorting algorithms

Problem	N	M	D	NSGAILSS	NSGAILENS
DTLZ1	200	3	7	4.0168e-1 (3.50e-1) \approx	3.8497e-1 (3.47e-1)
DTLZ2	200	3	12	5.2936e-1 (4.04e-3) \approx	5.2947e-1 (4.16e-3)
DTLZ3	200	3	12	8.6754e-4 (8.68e-3) \approx	0.0000e+0 (0.00e+0)
DTLZ4	200	3	12	4.9924e-1 (9.16e-2) \approx	5.0775e-1 (7.75e-2)
DTLZ5	200	3	12	1.9851e-1 (2.66e-4) \approx	1.9858e-1 (2.61e-4)
DTLZ6	200	3	12	1.9918e-1 (1.62e-3) \approx	1.9833e-1 (7.66e-3)
DTLZ7	200	3	22	2.4641e-1 (7.45e-3) \approx	2.4700e-1 (5.95e-3)
WFG1	200	3	12	7.0698e-1 (4.35e-2) \approx	7.0800e-1 (4.26e-2)
WFG2	200	3	12	9.0521e-1 (4.61e-3) \approx	9.0521e-1 (5.56e-3)
WFG3	200	3	12	3.7943e-1 (6.91e-3) \approx	3.7761e-1 (7.29e-3)
WFG4	200	3	12	5.0398e-1 (4.98e-3) \approx	5.0250e-1 (5.25e-3)
WFG5	200	3	12	4.8086e-1 (4.91e-3) \approx	4.8227e-1 (4.98e-3)
WFG6	200	3	12	4.4911e-1 (1.67e-2) \approx	4.5099e-1 (1.70e-2)
WFG7	200	3	12	5.1151e-1 (5.06e-3) \approx	5.1127e-1 (5.24e-3)
WFG8	200	3	12	4.2560e-1 (5.58e-3) \approx	4.2511e-1 (5.88e-3)
WFG9	200	3	12	4.8103e-1 (1.54e-2) \approx	4.8261e-1 (1.69e-2)
ZDT1	200	2	30	6.7204e-1 (1.02e-2) \approx	6.7025e-1 (1.18e-2)
ZDT2	200	2	30	4.1710e-1 (2.93e-2) \approx	4.0979e-1 (4.26e-2)
ZDT3	200	2	30	5.9952e-1 (2.48e-2) \approx	5.9800e-1 (2.19e-2)
ZDT4	200	2	10	4.7539e-1 (1.60e-1) \approx	4.8028e-1 (1.43e-1)
ZDT6	200	2	10	3.0822e-1 (3.28e-2) \approx	3.1657e-1 (3.14e-2)
+ / - / \approx				0/0/21	

Table A.3: Hypervolume results of NSGA III using ENS and SS as non-dominated sorting algorithms

Problem	N	M	D	NSGAIII _{SS}	NSGAIII _{ENS}
DTLZ1	200	3	7	4.0861e-1 (3.28e-1) \approx	3.6339e-1 (3.38e-1)
DTLZ2	200	3	12	5.5572e-1 (6.64e-4) \approx	5.5564e-1 (7.12e-4)
DTLZ3	200	3	12	0.0000e+0 (0.00e+0) \approx	0.0000e+0 (0.00e+0)
DTLZ4	200	3	12	4.9195e-1 (1.10e-1) \approx	5.2729e-1 (7.29e-2)
DTLZ5	200	3	12	1.9334e-1 (1.19e-3) \approx	1.9353e-1 (1.09e-3)
DTLZ6	200	3	12	1.8832e-1 (1.99e-2) \approx	1.8931e-1 (1.25e-2)
DTLZ7	200	3	22	2.4353e-1 (6.96e-3) \approx	2.4223e-1 (9.51e-3)
WFG1	200	3	12	6.3872e-1 (4.14e-2) \approx	6.3860e-1 (4.38e-2)
WFG2	200	3	12	9.0797e-1 (7.53e-3) \approx	9.0872e-1 (6.52e-3)
WFG3	200	3	12	3.5786e-1 (9.59e-3) \approx	3.5837e-1 (8.40e-3)
WFG4	200	3	12	5.2907e-1 (3.73e-3) \approx	5.2899e-1 (3.28e-3)
WFG5	200	3	12	5.0476e-1 (3.99e-3) \approx	5.0433e-1 (4.00e-3)
WFG6	200	3	12	4.7689e-1 (1.38e-2) \approx	4.7621e-1 (1.57e-2)
WFG7	200	3	12	5.3217e-1 (3.30e-3) \approx	5.3257e-1 (3.53e-3)
WFG8	200	3	12	4.4238e-1 (4.31e-3) \approx	4.4178e-1 (4.12e-3)
WFG9	200	3	12	5.0157e-1 (1.26e-2) \approx	4.9664e-1 (2.21e-2)
ZDT1	200	2	30	5.8580e-1 (1.71e-2) \approx	5.8261e-1 (2.36e-2)
ZDT2	200	2	30	3.7627e-1 (4.68e-2) \approx	3.7225e-1 (4.77e-2)
ZDT3	200	2	30	5.8775e-1 (2.25e-2) \approx	5.8719e-1 (2.23e-2)
ZDT4	200	2	10	1.4774e-1 (1.41e-1) \approx	1.3898e-1 (1.33e-1)
ZDT6	200	2	10	1.3789e-1 (6.32e-2) \approx	1.2865e-1 (6.59e-2)
+ / - / \approx				0/0/21	

Table A.4: Hypervolume results of AGE-MOEA using ENS and SS as non-dominated sorting algorithms

Problem	N	M	D	AGEMOEA_SS	AGEMOEA_ENS
DTLZ1	200	3	7	5.7846e-1 (3.10e-1) \approx	5.3217e-1 (3.29e-1)
DTLZ2	200	3	12	5.5815e-1 (7.82e-4) \approx	5.5821e-1 (7.47e-4)
DTLZ3	200	3	12	0.0000e+0 (0.00e+0) \approx	0.0000e+0 (0.00e+0)
DTLZ4	200	3	12	5.5106e-1 (4.14e-2) \approx	5.5106e-1 (4.33e-2)
DTLZ5	200	3	12	1.9871e-1 (2.37e-4) \approx	1.9867e-1 (2.14e-4)
DTLZ6	200	3	12	1.9986e-1 (1.29e-4) \approx	1.9984e-1 (2.06e-4)
DTLZ7	200	3	22	2.6095e-1 (1.38e-2) \approx	2.6238e-1 (1.18e-2)
WFG1	200	3	12	7.3037e-1 (4.99e-2) \approx	7.3510e-1 (3.59e-2)
WFG2	200	3	12	9.2500e-1 (3.78e-3) \approx	9.2470e-1 (2.93e-3)
WFG3	200	3	12	3.7470e-1 (6.23e-3) \approx	3.7434e-1 (6.31e-3)
WFG4	200	3	12	5.3781e-1 (3.06e-3) \approx	5.3831e-1 (2.78e-3)
WFG5	200	3	12	5.1083e-1 (4.00e-3) \approx	5.1089e-1 (4.02e-3)
WFG6	200	3	12	4.8810e-1 (1.39e-2) \approx	4.8682e-1 (1.57e-2)
WFG7	200	3	12	5.4575e-1 (1.86e-3) \approx	5.4607e-1 (1.73e-3)
WFG8	200	3	12	4.5165e-1 (3.90e-3) \approx	4.5186e-1 (3.36e-3)
WFG9	200	3	12	5.1504e-1 (1.31e-2) \approx	5.1518e-1 (7.03e-3)
ZDT1	200	2	30	6.7101e-1 (1.22e-2) \approx	6.7276e-1 (1.28e-2)
ZDT2	200	2	30	4.0408e-1 (5.15e-2) \approx	4.1451e-1 (4.25e-2)
ZDT3	200	2	30	6.0119e-1 (2.81e-2) \approx	6.0190e-1 (2.77e-2)
ZDT4	200	2	10	4.5246e-1 (1.46e-1) \approx	4.4662e-1 (1.55e-1)
ZDT6	200	2	10	3.3326e-1 (2.76e-2) \approx	3.3316e-1 (2.74e-2)
+ / - / \approx				0/0/21	

Table A.5: Hypervolume results of A-NSGA III using ENS and SS as non-dominated sorting algorithms

Problem	N	M	D	ANSGAIII _{SS}	ANSGAIII _{ENS}
DTLZ1	200	3	7	3.4846e-1 (3.21e-1) \approx	3.9379e-1 (3.26e-1)
DTLZ2	200	3	12	5.4795e-1 (3.45e-3) \approx	5.4838e-1 (3.11e-3)
DTLZ3	200	3	12	0.0000e+0 (0.00e+0) \approx	0.0000e+0 (0.00e+0)
DTLZ4	200	3	12	5.1223e-1 (9.53e-2) \approx	5.0266e-1 (8.97e-2)
DTLZ5	200	3	12	1.9477e-1 (8.47e-4) \approx	1.9478e-1 (7.83e-4)
DTLZ6	200	3	12	1.8950e-1 (2.33e-2) \approx	1.9224e-1 (1.12e-2)
DTLZ7	200	3	22	2.4546e-1 (6.85e-3) \approx	2.4564e-1 (7.92e-3)
WFG1	200	3	12	6.3209e-1 (4.22e-2) \approx	6.3202e-1 (4.42e-2)
WFG2	200	3	12	9.0284e-1 (1.94e-2) \approx	9.0513e-1 (6.03e-3)
WFG3	200	3	12	3.4916e-1 (1.07e-2) \approx	3.4912e-1 (1.08e-2)
WFG4	200	3	12	5.1007e-1 (5.41e-3) \approx	5.1001e-1 (5.20e-3)
WFG5	200	3	12	4.8924e-1 (4.16e-3) \approx	4.8955e-1 (4.11e-3)
WFG6	200	3	12	4.5924e-1 (1.72e-2) \approx	4.6154e-1 (1.53e-2)
WFG7	200	3	12	5.1958e-1 (4.11e-3) \approx	5.1961e-1 (4.42e-3)
WFG8	200	3	12	4.3066e-1 (6.92e-3) \approx	4.2964e-1 (6.47e-3)
WFG9	200	3	12	4.8734e-1 (2.45e-2) \approx	4.8977e-1 (2.18e-2)
ZDT1	200	2	30	5.8139e-1 (1.97e-2) \approx	5.8353e-1 (2.02e-2)
ZDT2	200	2	30	3.7758e-1 (3.48e-2) \approx	3.6982e-1 (5.51e-2)
ZDT3	200	2	30	5.8593e-1 (1.84e-2) \approx	5.8583e-1 (1.75e-2)
ZDT4	200	2	10	1.3921e-1 (1.46e-1) \approx	1.6663e-1 (1.51e-1)
ZDT6	200	2	10	1.1808e-1 (6.07e-2) \approx	1.3638e-1 (6.60e-2)
+ / - / \approx				0/0/21	

Appendix B

Non-dominated Sorting Algorithm MATLAB[®] Code

The MATLAB code for the four non-dominated sorting algorithms tested in this research are provided below.

B.1 Code for Deductive Sort

```

function out = DeductiveSort(pop,nSort)
    [N,M] = size(pop);
    pop = [(1:N)' pop zeros(N,1)];
    x = 1;
    count = 0;
    while min(pop(:,M+2)) == 0
        D = false(N,1);
        for i = 1:N
            if ~D(i) && pop(i,M+2) == 0
                for j = i+1:N
                    if ~D(j) && pop(j,M+2) == 0
                        d = dominates(pop(j,2:M+1),pop(i,2:M+1));
                        if d == 1
                            D(j) = true;
                        elseif d == 3
                            D(i) = true;
                            break
                        end
                    end
                end
            end
            if ~D(i)
                pop(i,M+2) = x;
                count = count + 1;
            end
        end
        end
        if count ≥ nSort
            break
        end
        x = x+1;
    end

    out = sortrows(pop(:,2:M+2),1);

function out = dominates(a, b)
    a_dom_b = false;
    b_dom_a = false;
    for m = 1:size(a,2)
        if b(m) < a(m)
            if a_dom_b
                out = 2;
            end
        end
        if ~b_dom_a
            b_dom_a = true;
        end
    end
end

```

B.1 Code for Deductive Sort

```
elseif a(m) < b(m)
    if b_dom_a
        out = 2;
    end
    if ¬a_dom_b
        a_dom_b = true;
    end
end
end
if a_dom_b && ¬b_dom_a
    out = 3;
elseif b_dom_a && ¬a_dom_b
    out = 1;
else
    out = 2;
end
end
end
```

B.2 Code for the Efficient Non-dominated Sort

B.2 Code for the Efficient Non-dominated Sort

```

function [FrontNo,MaxFNo] = ENS(varargin)
%NDSort - Perform non-dominated sorting by using efficient non-dominated
%sort.
%----- Reference -----
% [1] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, An efficient approach to
% nondominated sorting for evolutionary multiobjective optimization, IEEE
% Transactions on Evolutionary Computation, 2015, 19(2): 201-213.
% [2] X. Zhang, Y. Tian, R. Cheng, and Y. Jin, A decision variable
% clustering based evolutionary algorithm for large-scale many-objective
% optimization, IEEE Transactions on Evolutionary Computation, 2018, 22(1):
% 97-112.
%----- Copyright -----
% Copyright (c) 2021 BIMK Group. You are free to use the PlatEMO for
% research purposes. All publications which use this platform or any code
% in the platform should acknowledge the use of "PlatEMO" and reference "Ye
% Tian, Ran Cheng, Xingyi Zhang, and Yaochu Jin, PlatEMO: A MATLAB platform
% for evolutionary multi-objective optimization [educational forum], IEEE
% Computational Intelligence Magazine, 2017, 12(4): 73-87".
%-----

PopObj = varargin{1};
[N,M] = size(PopObj);
if nargin == 2
    nSort = varargin{2};
else
    PopCon = varargin{2};
    nSort = varargin{3};
    Infeasible = any(PopCon>0,2);
    PopObj(Infeasible,:) = repmat(max(PopObj,[],1),sum(Infeasible),1) + ...
        repmat(sum(max(0,PopCon(Infeasible,:)),2),1,M);
end

[FrontNo,MaxFNo] = ENS_SS(PopObj,nSort);

end

function [FrontNo,MaxFNo] = ENS_SS(PopObj,nSort)
[PopObj,~,Loc] = unique(PopObj,'rows');
Table = hist(Loc,1:max(Loc));
[N,M] = size(PopObj);
FrontNo = inf(1,N);
MaxFNo = 0;
while sum(Table(FrontNo<inf)) < min(nSort,length(Loc))
    MaxFNo = MaxFNo + 1;
    for i = 1 : N
        if FrontNo(i) == inf
            Dominated = false;
            for j = i-1 : -1 : 1
                if FrontNo(j) == MaxFNo
                    m = 2;
                    while m ≤ M && PopObj(i,m) ≥ PopObj(j,m)
                        m = m + 1;
                    end
                    Dominated = m > M;
                    if Dominated || M == 2
                        break;
                    end
                end
            end
            if ~Dominated
                FrontNo(i) = MaxFNo;
            end
        end
    end
end
FrontNo = FrontNo(:,Loc);
end

```


B.3 Code for the Best Order Sort

```

function Pareto = BOS(pop)

    % Algorithm 1: Initialisation
    [N,M] = size(pop);
    L = cell(N,M);
    C = cell(N,1);
    for i = 1:N
        C{i} = [1:M];
    end
    isRanked = false(N,1);
    SC = 0;
    RC = 0;
    R = zeros(N,1);
    pop = sortrows(pop,1);
    pop = [(1:N)' pop];
    Q = [(1:N)' zeros(N,M-1)];
    for q = 2:M
        pop = sortrows(pop,q+1);
        Q(:,q) = pop(:,1);
    end
    Q = sortrows(Q,1);
    pop = sortrows(pop(:,2:M+1),1);

    % Algorithm 2: Main Loop
    for i = 1:N
        for j = 1:M
            s = Q(i,j);
            C{s} = C{s}(C{s}≠j);
            if isRanked(s) == true
                R(s);
                L{R(s),j} = union(L{R(s),j},s);
            else
                FindRank(s,j);
                isRanked(s) = true;
                SC = SC + 1;
            end
        end
        if SC == N
            break
        end
    end
    Pareto = [pop R];

    % Algorithm 3: FindRank
    function FindRank(s,j)
        done = false;
        for k = 1:RC
            check = false;
            for t = L{k,j}
                check = DominationCheck(s,t);
                if check == true
                    break
                end
            end
            if check == false
                R(s) = k;
                done = true;
                L{R(s),j} = union(L{R(s),j},s);
                break
            end
        end
        if done == false
            RC = RC + 1;
            R(s) = RC;
            L{R(s),j} = union(L{R(s),j},s);
        end
    end
end

```

B.3 Code for the Best Order Sort

```
    end
end

% Algorithm 4: DominationCheck
function result = DominationCheck(s,t) % Algorithm 4
    flag = true;
    for d = C{t}
        if pop(s,d) < pop(t,d)
            flag = false;
        end
    end
    if flag == false
        result = false;
    else
        result = true;
    end
end
end
```

B.4 Code for the Shadow Sort

```

function [FrontNo, MaxFNo] = ShadowSort(varargin)
    pop = varargin{1};
    [N,M] = size(pop);
    count = 0; % Counting how many solutions have been ranked
    if nargin == 2
        nSort = varargin{2};
    else
        PopCon = varargin{2};
        nSort = varargin{3};
        Infeasible = any(PopCon>0,2);
        pop(Infeasible,:) = repmat(max(pop,[],1),sum(Infeasible),1) + ...
            repmat(sum(max(0,PopCon(Infeasible,:)),2),1,M);
    end

    if isnan(nSort) || nSort > N
        nSort = N;
    end

    if M == 2
        [FrontNo, MaxFNo] = CaseI(pop, nSort);
    elseif M > 2 || M < 15
        [FrontNo, MaxFNo] = CaseII(pop, nSort);
    else
        [FrontNo, MaxFNo] = CaseIII(pop, nSort);
    end

% Case I: If the problem is bi-objective

function [FrontNo, MaxFNo] = CaseI(pop, nSort)
    [pop,~,Loc] = unique(pop,'rows');
    [N,M] = size(pop);

    FrontNo = [[1:N]' pop, zeros(N,1)];
    k = 0; % Assign k as front number
    while count < nSort
        index = Inf; % Create index vector for each front with ...
        a large number
        k = k + 1;
        Key = FrontNo(FrontNo(:,M+2)==0,1);
        for p = 1:size(Key) % For p = 1 to N
            L = Key(p);
            if FrontNo(L, 3) < index % Compare p to the front index ...
                (best 2nd objective so far)
                index = FrontNo(L, 3); % If p's 2nd obj < index, set ...
                p's 2nd obj value = new index
                FrontNo(L, M+2) = k; % Set p's front value to the ...
                applicable number
                count = count + 1;
            end
        end
    end

    FrontNo = FrontNo(:,2:4);
    MaxFNo = max(FrontNo(:,3));
    FrontNo(FrontNo(:,3)==0,3)=Inf;
    FrontNo(:,1:2)=[];
    FrontNo = FrontNo(Loc);
    FrontNo = FrontNo';
end

% Case II: If the problem has more than 2 but less than 15 objectives

function [FrontNo, MaxFNo] = CaseII(pop, nSort)
    [pop,~,Loc] = unique(pop,'rows');
    [N,M] = size(pop);
    Tie_Break = 2:M; % Initialise tie-break vector

```

B.4 Code for the Shadow Sort

```

FrontNo = zeros(N,1); % out = pop with Inf column ...
vector for rank number
Key = pop; % Create an array 'Key' containing ...
only the objective values
Key(:,1) = [1:N]'; % Replace actual values of Obj 1 with ...
ID vector
for i = 2:M % For the remaining objectives
for j = 0:M-2
Tie.Break(j+1) = mod(i+j,M)+1;
end
Key = sortrows(Key,[i, Tie.Break]);
Key(:,i) = [1:N]'; % Replace actual values of Obj i with ...
ID vector
end
Key = sortrows(Key,1); % Sort 'Key' according to Obj 1
k = 0;

while true
k = k + 1;
Flag = false(N,1);
KeyM = Key(FrontNo==0,1);
for i = 1:size(KeyM,1) % For all solutions in the population
p = KeyM(i,1);
if Flag(p) == true % If point i's flag is already 1, skip.
continue
end
FrontNo(p) = k;
count = count + 1;
Bitmat = Key > Key(p,:);
Flag = bitor(Flag,all(Bitmat,2));
end
if count ≥ nSort
break
end
end

MaxFNo = max(FrontNo);
FrontNo(:,1:end-1)=[];
FrontNo(FrontNo(:)==0) = Inf;
FrontNo = FrontNo(Loc);
FrontNo = FrontNo';
end

% Case III: If the problem has 15 or more objectives

function [FrontNo, MaxFNo] = CaseIII(pop, nSort)
ID = [1:N]'; % Creating point ID vector
[pop,~,Loc] = unique(pop,'rows');
FrontNo = zeros(N,1); % out = pop with Inf column vector for rank ...
number
Key = pop; % Create an array 'Key' containing only the objective ...
values
Standing = zeros(N,M);
Key(:,1) = ID; % Replace actual values of Obj 1 with ID vector
Standing(:,1) = ID;
for i = 2:M % For the remaining objectives
Key = sortrows(Key,i);
Key(:,i) = ID; % Replace actual values of Obj i with ID vector
Standing(:,i) = Key(:,1); % Set all rows, column 2, page i to Obj 1 ID
end
Key = sortrows(Key,1); % Sort 'Key' according to Obj 1
Standing = sortrows(Standing,1);

while true
Flag = false(N,1);
KeyM = Key(FrontNo==0,1);
for i = 1:size(KeyM,1) % For all solutions in the population
p = KeyM(i,1);
if Flag(p) == true % If point i's flag is already 1, skip.
continue
end
Dominated = [];
Potential = [];

```

B.4 Code for the Shadow Sort

```

    FrontNo(p) = k;
    count = count + 1;
    [CheckBelow, Objective] = max(Key(p,:));
    Potential = Standing(CheckBelow+1:end,Objective);
    Potential(Potential<p)=[];
    if ~isempty(Potential)
        for x = 1:size(Potential)
            if DominationCheck(Key(p,:), Key(Potential(x,:),:)) == true
                Flag(Potential(x),2) = Flag(Potential(x),2) + 1;
            end
        end
    end
end
end
if count ≥ nSort
    break
end
end

MaxFNo = max(FrontNo);
FrontNo(FrontNo(:)==0) = Inf;
FrontNo = FrontNo(Loc);
FrontNo = FrontNo';
end

% Domination Check: Classic solution domination check used in Case III

function result = DominationCheck(s,t)
    r = size(s,2);
    for y = 2:r
        % for all objectives in C-t
        % if t is better than s
        if t(y) < s(y)
            result = false; % s cannot dominate t
            break
        end
    end
    result = true; % s dominates t
end
end
end
end

```

B.4 Code for the Shadow Sort
