NTNU
Norwegian University of
Science and Technology

Multi-Objective Animal Migration Optimization

A Metaheuristic Optimization Algorithm

Eirik Bertelsen
Christian Duvholt

Master of Science in Informatics
Submission date: June 2018
Supervisor: Anders Kofod-Petersen, IDI

Norwegian University of Science and Technology
Department of Computer Science

Abstract

Metaheuristic optimization has received a lot of attention over the last couple of decades. Some optimization problems are just too computationally heavy to solve with traditional search techniques, especially when there is more than one objective to optimize for. There have been many promising metaheuristic algorithms for solving multi-objective problems, but algorithms have different strengths and weaknesses, and no algorithm can be the best at solving every problem. New metaheuristic approaches are, therefore, an interesting topic to study. In this thesis, a structured literature review of state of the art single and multi-objective metaheuristic algorithms was performed. A framework was created to implement and evaluate promising algorithms. Moreover, a comparison study of single-objective algorithms was performed, where the most suitable single-objective algorithm was extended to handle multi-objective problems. The extended algorithm's performance was compared with other well-performing algorithms from the literature. The resulting algorithm is called Multi-Objective Animal Migration Algorithm (MOAMO) and showed competitive results when compared with eight other algorithms on 22 test functions using three performance metrics.

Sammendrag

Metaheuristisk optimalisering har fått stor oppmerksomhet de siste par tiårene. Noen optimeringsproblemer er for beregningsmessige tunge å løse med tradisjonelle søketeknikker, spesielt når det er mer enn ett mål å optimalisere for. Det har vært mange lovende metaheuristiske algoritmer for å løse problemer med flere mål, men algoritmer har forskjellige styrker og svakheter, og ingen algoritme er best til å løse alle problemer. Nye metaheuristiske tilnærminger er derfor et interessant emne å studere. I denne masteroppgaven ble det gjennomført en strukturert litteraturgjennomgang av nåtidens mest aktuelle metaheuristiske algoritmer for optimalisering av enten ett eller flere mål. Et rammeverk ble laget for å implementere og evaluere lovende algoritmer. Videre ble det utført en undersøkelse av optimaliseringsalgoritmer for problemer med ett mål, hvor den mest hensiktsmessige av dem ble videreutviklet til å kunne håndtere problemer med flere mål. Den videreutviklede algoritmens ytelse ble så sammenlignet med andre vellykkede algoritmer fra litteraturen. Den endelige algoritmen kalles Multi-Objective Animal Migration Algorithm (MOAMO) og får gode resultater når den testes med åtte andre algoritmer på 22 testfunksjoner ved hjelp av tre måter å måle ytelse på.

Preface

This thesis was written during the fall of 2017 and the spring of 2018 at the Department of Computer Science (IDI), Norwegian University of Science and Technology (NTNU).

We would like to thank our supervisor Anders Kofod-Petersen for the valuable guidance and feedback during this project. We would also like to thank family and friends for the moral support and assistance they have provided.

Eirik Bertelsen and Christian Duvholt

Trondheim, June 25, 2018

Contents

List of Figures

List of Tables

Acronyms

AMO Animal Migration Optimization.

AMOSA Archived Multi-Objective Simulated Annealing.

CEC Congress on Evolutionary Computation.

CLI Command Line Interface.

DA Dandelion Algorithm.

DFA Dragonfly Algorithm.

DPP Dual-Population Paradigm.

EA Evolutionary Algorithm.

EWA Earthworm Optimization Algorithm.

GA Genetic Algorithm.

GD Generational Distance.

HV Hypervolume.

IGD Inverted Generational Distance.

LOA Lion Optimization Algorithm.

MOALO Multi-Objective Ant Lion Optimizer.

MOAMO Multi-Objective Animal Migration Algorithm.

MODFA Multi-Objective Dragonfly Algorithm.

MOEA Multi-Objective Evolutionary Algorithm.

Acronyms

MOEA/D Multi-Objective Evolutionary Algorithm Based On Decomposition.

MOEA/DD Multi-Objective Evolutionary Algorithm based on Dominance and Decomposition.

MOGWO Multi-Objective Grey Wolf Optimizer.

MOP Multi-objective Optimization Problem.

MOPSO Multi-Objective Particle Swarm Optimization.

MOWOA Multi-Objective Whale Optimization Algorithm.

MS Moth Search.

NS-MFO Non-dominated Sorting Moth-Flame Optimization.

NSGA Non-dominated Sorting Genetic Algorithm.

NSIWO Non-dominated Sorting Invasive Weed Optimization.

PAES Pareto Archived Evolution Strategy.

PSO Particle Swarm Optimization.

RQ Research Question.

SA Simulated Annealing.

SI Swarm Intelligence.

SLR Structured Literature Review.

SMPSO Speed-constrained Multi-objective Particle Swarm Optimization.

SPEA Strength Pareto Evolutionary Algorithm.

SRQ Search Research Question.

VEGA Vector Evaluated Genetic Algorithm.

1. Introduction

This chapter gives an introduction to this thesis. First, background and motivation are presented, then the goal and research questions are defined. The research method to answer research questions is outlined and lastly, the thesis structure is presented.

1.1. Background and Motivation

Computer-aided optimization is a popular area of research for solving problems in several fields like biology, physics, chemistry, economy, and sociology [Coello and Lamont, 2004]. Some of the problems encountered in these fields are just too computationally heavy to find optimal solutions by calculating all possible solutions. However, finding optimal solutions is not always necessary. Sometimes a close to optimal solution can be good enough. In cases like this, metaheuristic optimization algorithms are perfect [Coello et al., 2007, p.62]. A metaheuristic is a high-level procedure that is used to guide a search, allowing for near optimal results with acceptable computational cost. Metaheuristics have been applied to many interesting real-world problems. For example when NASA needed a satellite antenna for their 2006 mission "Space Technology 5" they used a metaheuristic algorithm to find the best possible shape of the antenna [Hornby et al., 2006]. Metaheuristics have also been used to solve other problems like aerodynamic design [Hasenjäger et al., 2005], treatment planning optimization in radiation therapy [Yu, 1997], designing a mobile telecommunication network [Meunier et al., 2000], and many more. As most real-world optimization problems have multiple objectives [Konak et al., 2006; Coello, 2015] having good multi-objective metaheuristic algorithms is very important. Most modern multi-objective metaheuristic algorithms use the *a posteriori* approach [Mirjalili, 2016], where a set of solutions are found, and presented after optimizing. *A posteriori* algorithms are often nature-inspired. The field of nature-inspired multi-objective algorithms began in the mid-80's with Vector Evaluated Genetic Algorithm (VEGA), a multi-objective algorithm based on the single-objective Genetic Algorithm (GA). Since then many approaches to multi-objective optimization have been tried, and new algorithms are developed all the time. However, it is hard to solve multi-objective optimization problems,

and even today there is room for improvement. According to the No Free Lunch theorem [Wolpert and Macready, 1997], an algorithm that can perfectly solve all classes of optimization problems cannot exist since an improvement in solving one class of problems will reduce the performance of other classes. Consequently, it is worthwhile to create new multi-objective algorithms. Lately, many new single-objective algorithms have been extended to handle multiple objectives in hopes of creating new multi-objective algorithms with improved convergence and diversity [Savsani and Tawhid, 2017].

1.2. Goal and Research Questions

The following goal was set for this thesis:

Goal *Create a multi-objective metaheuristic algorithm based on a single-objective algorithm from the literature.*

The goal is to find an algorithm that was created for single-objective optimization and extend this algorithm to handle multi-objective problems. The resulting algorithm should be able to handle real value problems that are unconstrained. It should be competitive with other state of the art multi-objective optimization algorithms. It should be general, that is, it should able to perform well on many different problems with different characteristics. The algorithm will be optimized for problems with two or three objectives.

To achieve the goal three research questions (RQs) were created.

RQ1 *Which single-objective algorithm has the best potential for multi-objective extension?*

Promising new single-objective algorithms are developed all the time. But just because an algorithm performs well on single-objective problems, does not necessarily mean that it will do well when extended to handle more objectives. Some algorithms use techniques that make it hard to extend them to multi-objective without abandoning the core concepts of the algorithm, making it less of an extension and instead just a new algorithm that uses some techniques from the old one. The aim of this question is to find an algorithm that performs well and can be extended to handle multiple objectives without changing the core principles of the algorithm.

RQ2 *Which multi-objective techniques are most suitable for extending the selected algorithm to multi-objective?*

There are several techniques for multi-objective optimization that are used by state of the art algorithms. As most of the current multi-objective algorithms in the literature use the *a posteriori* approach, the aim of this question is to get an overview of the most promising *a posteriori* techniques, learn how they work and find out which seems to be the most suitable for extending the selected single-objective algorithm.

RQ3 *How does the proposed algorithm's performance compare to other competitive algorithms from the literature?*

Since the final version of the algorithm should be competitive, its performance should be compared with competitive algorithms. The algorithms should be tested on a range of multi-objective optimization problems with different characteristics.

1.3. Research Method

To help answer the research questions related work was studied to get a better understanding of the field. A structured literature review (SLR) [Kofod-Petersen, 2014] was performed to gather information about the state of the art. In the literature review, the focus was on finding single-objective algorithms that are suitable for multi-objective extension, multi-objective techniques used by the state of the art, competitive multi-objective algorithms, and test functions and performance metrics that are used when testing multi-objective algorithms.

A framework for metaheuristic optimization was created. This framework includes a range of single and multi-objective test functions for testing the algorithms. A set of commonly used multi-objective performance metrics was implemented. Statistical measures like mean and standard deviation are implemented to assess the performance over several runs.

Using this framework some of the more promising single-objective algorithms were implemented and compared against each other in an experiment to answer **RQ1**. Based on the results from this experiment, one of the algorithms was chosen for multi-objective extension.

The selected algorithm was extended to handle problems with multiple objectives using different techniques to answer **RQ2**. The resulting algorithms were compared with each other in an experiment. The best performing algorithm was selected as the final version of the algorithm.

To answer **RQ3** the algorithm was compared with other competitive algorithms in a more extensive experiment. The results of this experiment have been analyzed,

to evaluate the quality of the final version of the algorithm.

1.4. Thesis Structure

This thesis has three experiments. Two smaller ones are part of Chapters 5 and 6, while a larger experiment covers all of Chapter 7. They all follow the same structure: experimental setup, results and evaluation, discussion, and conclusion.

In Chapter 2 the background theory that is necessary to understand this thesis is explained. Chapter 3 is a structured literature review covering the state of the art with a summary and a discussion of the related work. In Chapter 4 the framework architecture is explained. Chapter 5 answers **RQ1** with an experiment comparing single-objective algorithms from the literature review where one algorithm is selected for multi-objective extension and explained in detail. Chapter 6 answers **RQ2** with several extensions of the selected algorithm to multi-objective using different techniques, an experiment comparing the different versions, and the presentation of the final version of the algorithm. Chapter 7 answers **RQ3** with a multi-objective comparison experiment where the final version of the algorithm is compared with other competitive algorithms. Lastly, the thesis is concluded in Chapter 8.

2. Background Theory

In this chapter the relevant background theory that is necessary to understand this thesis is presented.

2.1. Optimization

The goal of optimization problems is to find the optimal solution(s) out of a set of possible solutions. This is usually done by minimizing (or maximizing depending on the problem) the output by tweaking the input variables. The optimal solution is not known before-hand and might require exhaustive search techniques to be found. Optimization of a minimization problem with a single objective can be broken down to the mathematical statement in Equation (2.1) [Coello et al., 2007, p.4].

$$\text{minimize } f(\boldsymbol{x}),\ \boldsymbol{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n \tag{2.1}$$

Here the output of the objective function $f(\boldsymbol{x})$ is minimized for all possible solutions $\boldsymbol{x}$ with n decision variables. The output of $f(\boldsymbol{x})$ is referred to as the fitness of $\boldsymbol{x}$ and the evaluation itself is referred to as a fitness evaluation in this thesis. Optimization problems can have more than one objective (criterion) which decides the number of output variables. Real-life optimization problems usually involves constraints which have to be upheld for the answer to be valid. Examples of constraints are maximum time usage or physical limitations. Problems can have equality and inequality constraints, as seen in Equation (2.2) [Coello et al., 2007, p.6].

$$\begin{aligned} g_i(\boldsymbol{x}) &= 0 \quad \text{for } i = 1, \ldots, p \quad \text{Equality constraints} \\ h_j(\boldsymbol{x}) &\leq 0 \quad \text{for } j = 1, \ldots, m \quad \text{Inequality constraints} \end{aligned} \tag{2.2}$$

For a solution to be considered valid, its decision variables $\boldsymbol{x}$ have to satisfy p equality constraints and m inequality constraints.

Combinatorial optimization problems are a subset of optimization problems where the goal is to the find the optimal order of a set of items. A classic example is the Traveling Salesman Problem (TSP) where the goal is the find the shortest route between cities by only visiting each city once and returning to the origin city [Schrijver, 2005]. For this example, $\boldsymbol{x}$ is the order in which the cities should be visited and $f(\boldsymbol{x})$ is the total distance traveled. To solve problems like this some traditional mathematical techniques like linear programming, non-linear programming, and dynamic programming have been used. They guarantee that an optimal solution is found, but they all have difficulties with computationally heavy problems, like TSP, when the search space grows. In other words, doing an exhaustive search is impractical for many optimization problems.

Optimization problems can be broken down into classes based on their computational complexity. A class which has gotten much research is the NP-complete class [Michael and David, 1979]. NP stands for nondeterministic polynomial time and is a class of problems where the answer can be verified in polynomial time. What makes NP-complete problems so interesting is that even though solutions to these problems cannot be found in polynomial time (assuming $P \neq NP$), they are as hard as the hardest problems in NP. If one of the NP-complete problems can be solved in polynomial time, then all NP-complete problems can too [Michael and David, 1979]. When the search space of these problems grows, it becomes exceptionally computationally expensive and, in some cases, impossible to solve these problems. Good solutions to these problems can still be found using heuristic approaches.

The term *heuristic* comes from the Greek verb *heuriskein* which means "to find" [Blum and Roli, 2003, p.270]. Heuristic methods focus on finding solutions that are not necessarily optimal, but good enough. For problems where finding an optimal solution is impractical or not possible, a heuristic approach is a good alternative. Mathematical functions are usually used to express optimization problems where the goal is to either minimize or maximize the output. The function is not known to the solver and acts as a black box where only input and output is known. Heuristic approaches like this can reduce the computation time drastically. Heuristics are usually problem specific meaning that a heuristic for one NP-complete problem may not work as well on another [Stützle, 1999, p.23]. The prefix *meta* means "beyond, in an upper level" [Blum and Roli, 2003, p.270]. A metaheuristic is a high-level procedure that is used to guide other heuristics that can achieve near optimal results on an optimization problem with acceptable computational cost. Metaheuristics are usually non-deterministic and are not problem-specific Blum and Roli [2003].

2.2. Characteristics of Metaheuristics

Metaheuristics have long been an interesting field of study, and many different approaches have been tried to create metaheuristics that perform well. The goal is to have an algorithm that finds high-quality solutions within an acceptable time frame. There are some common characteristics that can be seen in the vast majority of metaheuristics.

Most metaheuristics use a combination of exploration and exploitation techniques [Luke, 2009, p.22]. Exploration aims to find promising solutions far away from each other using a form of global search. A simple form of an exploration technique is random search which searches by selecting random solutions [Luke, 2009, p.7]. When talking about exploitation in regards to search algorithms, it means focusing on searching for better solutions in areas where promising solutions have been found, also known as local search. Local search techniques alone are not suitable for most problems, as they tend to get stuck in local optima. An example of a simple local search technique is hill climbing, which selects the best neighboring solution to the current solution until it reaches an optimum [Luke, 2009, p.8]. The effectiveness of exploration and exploitation depends on the optimization problem as some problems can be solved using only exploitation techniques, while other problems have several local optima and therefore need a degree of exploration. An example of a metaheuristic using both exploration and exploitation is Simulated Annealing (SA) which uses exploitation by always selecting a newly found solution if it is better than the current, but will also sometimes select a worse solution to promote exploration of the search space.

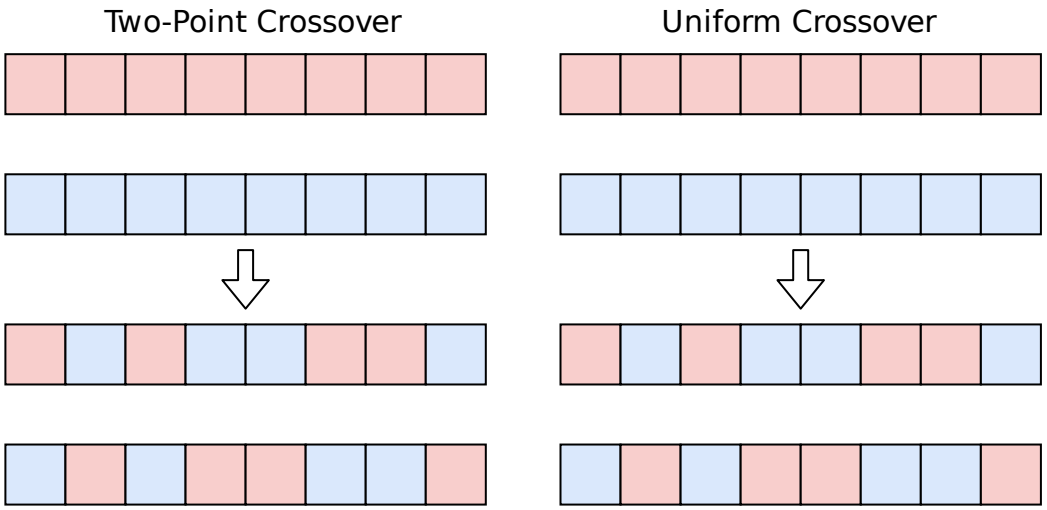
Metaheuristics have either a single-state or a population [Luke, 2009, p.29]. The single-state method only uses the current solution to search and replaces this solution when meeting a set of conditions. SA is an example of an algorithm using this method. Population-based algorithms, on the other hand, use several solutions to search. The initial population usually consists of randomly selected solutions from the search space. These solutions can then explore by sharing information to find new improved solutions. An example of such an algorithm is Particle Swarm Optimization (PSO) [Eberhart and Kennedy, 1995] which uses a combination of the current solution, the personally best-known solution so far, and the globally best-known solution to guide the population as a form of exploitation and exploration combined.

2.3. Nature Inspired Metaheuristics

Many metaheuristic algorithms are nature inspired [Birattari et al., 2001]. In this section, the focus is on two subsets of metaheuristic algorithms that are inspired by different phenomena found in nature. Evolutionary algorithms are inspired by biological evolution, while swarm intelligence algorithms are inspired by animal movement.

2.3.1. Evolutionary Algorithms

Evolutionary Algorithms (EAs) are a subset of metaheuristic algorithms that are inspired by Darwin's theory of evolution. In this section, some of the operators used by EAs are explained.



(a) Two-point crossover (b) Uniform crossover

Figure 2.1.: Two common types of crossover

Crossover

Crossover is a population-based technique inspired by inheritance of genes in the natural world, where offspring will get some of the genes from each of their parents. Likewise, when creating a new solution for the next generation, some of the data from each parent will be combined into a new solution. There is no guarantee that this new solution is as good or better than the parents, but it can help keep the population diverse, especially if the parents are dissimilar.

Single-point crossover picks a point in the solution where the information on each

side of this point will go to different offspring. This can be done with more than one point. An example of two-point crossover can be seen in Figure 2.1a. Uniform crossover is a type of crossover where each bit of information has a percentage chance of being passed on to the offspring based on a mixing ratio. This mixing ratio is typically 0.5, meaning that the offspring inherits approximately half from each parent. An example of uniform crossover can be seen in Figure 2.1b.

Mutation

Mutation is a small (often random) change to a solution. Mutation is usually applied randomly to solutions and only changes some characteristics of the solution, for example, changing one decision variable. Mutation operators help keep diversity within a population. This is important to make sure the algorithm does not get stuck with a population of extremely similar or even equal solutions.

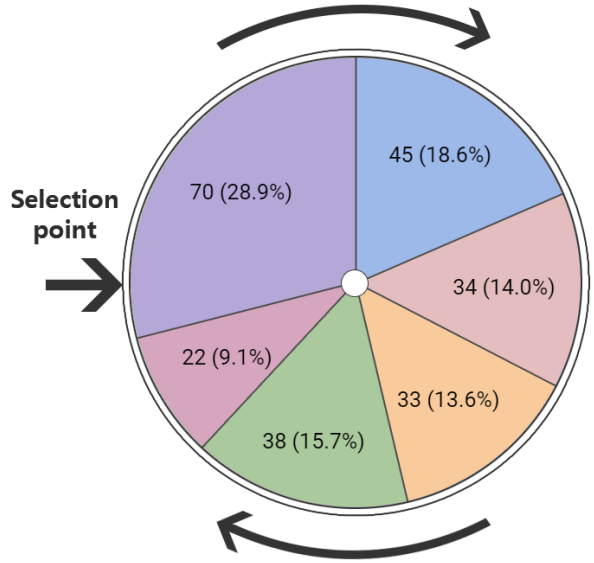


Figure 2.2.: Example of fitness proportionate selection on a maximization problem. Fitness is shown on the labels with chance of selection in parenthesis. The section the arrow points at after the wheel is done spinning gets selected.

Selection Strategy

When creating a new generation in an EA, one or more solutions (parents) are chosen to create the next generation. This can be tricky, as it is important to keep solutions similar to the best known current solution, while also exploring other areas of the search space. There are many ways of doing this.

Tournament selection chooses a set of potential parents at random and holds a tournament where the solution with the best fitness is chosen. Tournament selection with few solutions makes it more likely that bad solutions are selected, helping with the diversity of the population. Tournament selection with many solutions makes it more likely that a good solution is chosen, but can lead to a lack of diversity in the population, increasing the chance of getting stuck in a local optimum.

Another common way of selecting parents is fitness proportionate selection. This gives every solution a probability of being chosen based on their fitness value, which means that the best solutions have a higher chance of being chosen compared to bad ones. This is also referred to as roulette wheel selection, as it can be visualized as a roulette wheel where each fitness value gets a section, and the size of this section is based on how good the fitness is compared to the rest. When the wheel spins the larger sections have a higher chance of being chosen. An example of fitness proportionate selection on a maximization problem can be seen in Figure 2.2.

Elitism

To avoid reducing the quality of the solutions in the population, elitism is commonly used. Algorithms with elitism always keep some of the best solutions from the previous generation. Although commonly associated with GA, elitism is not something that happens naturally during evolution.

2.3.2. Swarm Intelligence

Swarm Intelligence (SI) is the collective behavior exhibited by self-organized beings that act as one system [Parpinelli and Lopes, 2011]. While SI algorithms do not necessarily have to be nature inspired, many of the most well-known algorithms are inspired by how groups of animals like birds, fish, and ants move. Unlike EA techniques the techniques used by SI algorithms vary greatly from algorithm to algorithm. Parpinelli and Lopes [2011] have categorized the behavior of SI algorithms into three categories which are explained with some examples.

Mechanism of Exploitation

In many SI algorithms, a leader is used to guide the search. The leader is usually a high-quality solution from the population. Moving towards high-quality solutions lets the population exploit areas of the search space where the most promising solutions were found. If the leader has a too big impact on the movement of the population, it can lead to the population converging to a local optimum. Cuckoo Search [Yang and Deb, 2009], Flower Pollination Algorithm [Yang, 2012], and PSO are examples of algorithms that use a global leader for exploitation. Bees Algorithm [Pham et al., 2006] does not use a global leader, but instead, it selects

the n best sites explored by bees for further neighborhood search.

Mechanism of Exploration

A common way for SI algorithms to explore the search space is using random search. Artificial Bee Colony [Karaboga and Basturk, 2007] has three types of bees where one, scout bees, visit random positions in the search space. Another common technique is to add a random factor to the calculation of new positions as done by Firefly Algorithm [Yang, 2010] and PSO. The random factor is commonly based on random numbers from the uniform or Gaussian distributions.

Communication Model

The communication model dictates the way individuals in the swarm communicate with each other. This can either be done in a broadcast-like manner where a greater part of the population is communicated with or by directly transmitting information from one individual to another. Ant Colony Optimization [Dorigo et al., 1996] use a broadcast model where the whole swarm shares pheromone to indicate where they have traveled before. While PSO broadcasts the position of the best leader to the whole swarm. Examples of direct communication are swarm algorithms that use mating. Marriage in Honey Bees Optimization [Abbass, 2001] is a swarm algorithm that uses mating between the queens and several drones to create new workers as a form of communication.

2.4. Multi-Objective Optimization

Many optimization algorithms focus on optimizing a single objective. For example, maximizing performance. However, some problems have more than one objective to optimize. For example, minimizing fuel usage while also maximizing engine power. Most real-world optimization problems are more similar to multi-objective problems than single-objective [Konak et al., 2006].

This section explains the basic concepts of multi-objective optimization and some of the most common techniques used.

2.4.1. Multi-Objective Optimization Problem

Multi-objective optimization is more complicated than single-objective optimization. When optimizing for one objective, a solution can be objectively compared solely be checking if the fitness value is lower or higher than the other. With several objectives, the comparison is more complicated as different solutions can be superior at solving different objectives. A definition of the multi-objective optimization problem is given in Definition 2.4.1 [Coello et al., 2007, p.8].

Definition 2.4.1 (Multi-objective optimization problem (MOP))
Given a vector function $\vec{\boldsymbol{f}}(\vec{\boldsymbol{u}}) = [\boldsymbol{f_1}(\vec{\boldsymbol{u}}), \boldsymbol{f_2}(\vec{\boldsymbol{u}}), ..., \boldsymbol{f_k}(\vec{\boldsymbol{u}})]$ *with* $\boldsymbol{k}$ *objectives and its feasible search space* $\boldsymbol{\Omega}$*, the MOP consists of finding a vector* $\vec{\boldsymbol{u}} \in \boldsymbol{\Omega}$ *that optimizes the vector function* $\vec{\boldsymbol{f}}(\vec{\boldsymbol{u}})$*. Without loss of generality we will assume only minimization functions.*

Depending on the problem, objectives in multi-objective optimization may be independent of each other or dependent. Often objectives conflict with each other, meaning that optimizing one objective can have adverse effects on the other objectives. Multi-objective problems are hard to solve. In fact, finding the global optimum of a general MOP is NP-Complete [Veldhuizen, 1999, p.2-2]. It is therefore common to use metaheuristic algorithms on multi-objective optimization problems.

2.4.2. Decision Making in Multi-Objective Optimization

Since there is no single optimal solution to a multi-objective problem, a solution has to be selected based on preferences. The instance responsible for selecting solutions is referred to as the Decision Maker in this section [Coello et al., 2007]. Multi-objective algorithms can be split into three categories based on their use of the Decision Maker.

A Priori Preference Articulation

With the *A priori* approach the Decision Maker combines the objectives into one objective, effectively making the MOP single-objective prior to running the optimization. One example of a simple *a priori* technique is the linear fitness combination seen in Equation (2.3) [Coello et al., 2007].

$$\text{fitness} = \min \sum_{i=1}^{m} w_i f_i(x) \tag{2.3}$$

Where $w_i \geq 0$ and w_i is the weight assigned to objective i by the Decision Maker, and m is the number of objectives. Another technique is lexicographic ordering, where the Decision Maker ranks the objectives in order of importance. The objectives are then minimized in sequence to produce the solution. Due to their simplicity these, and other *a priori* techniques, usually do not produce acceptable results.

Progressive Preference Articulation

Progressive preference articulation means that the Decision Maker makes decisions while the optimization process is still ongoing. Normally this is performed by

searching for a non-dominated solution, present it to the Decision Maker, modify the objective preferences based on the reaction before continuing the search. This process continues until the Decision Maker is satisfied with the result or until the search has converged. Some examples of progressive preference articulation are Biased Crowding Distance [Deb and Algorithms, 1999] and ϵ-MOEA [Laumanns et al., 2002]. Progressive preference articulation requires that the Decision Maker invest more time into the optimization process compared to other techniques.

A Posteriori Preference Articulation

The goal of *a posteriori* is to find a wide range of solutions covering all the objectives while at the same time optimizing them all. At the end of the search, the solution set is presented to the Decision Maker which then selects a final solution. Some examples of *a posteriori* techniques are criterion selection, aggregation selection and Pareto sampling [Coello et al., 2007]. The first implementation of a multi-objective evolutionary algorithm (MOEA), VEGA, used criterion selection [Coello et al., 2007]. Most of the current multi-objective algorithms in the literature are *a posteriori* [Mirjalili, 2016] because the technique is the most convenient since it requires the least amount of interaction from the Decision Maker [Branke et al., 2001]. Some examples of Pareto techniques are presented in Section 2.4.4.

2.4.3. Pareto Definitions

To make it easier to explain some of the coming sections some definitions are made [Veldhuizen 1999, p.2-1 - p.2-4; Santiago et al. 2014, p.455]:

Definition 2.4.2 (Pareto dominance)
A vector $\vec{\boldsymbol{u}}$ *dominates* $\vec{\boldsymbol{v}}$ *(denoted* $\vec{\boldsymbol{u}} \preceq \vec{\boldsymbol{v}}$*) if and only if* $\boldsymbol{f}_i(\vec{\boldsymbol{u}}) \leq \boldsymbol{f}_i(\vec{\boldsymbol{v}})$ *for all functions* $\boldsymbol{i}$ *in* $\boldsymbol{f}$ *and there is at least one* $\boldsymbol{i}$ *such that* $\boldsymbol{f}_i(\vec{\boldsymbol{u}}) < \boldsymbol{f}_i(\vec{\boldsymbol{v}})$

Definition 2.4.3 (Pareto optimality)
A vector $\vec{\boldsymbol{u}} \in \boldsymbol{\Omega}$ *is Pareto optimal if there is no* $\vec{\boldsymbol{v}} \in \boldsymbol{\Omega}$ *for which* $\vec{\boldsymbol{v}} \preceq \vec{\boldsymbol{u}}$*.*

Definition 2.4.4 (Pareto set)
Given a MOP, the Pareto set $\boldsymbol{\mathcal{P}}$ *is defined as:*

$$\boldsymbol{\mathcal{P}} = \{\vec{\boldsymbol{u}} \in \boldsymbol{\mathcal{S}} | \neg \exists \vec{\boldsymbol{v}} \in \boldsymbol{\mathcal{S}}\ \vec{\boldsymbol{v}} \preceq \vec{\boldsymbol{u}}\}$$

A Pareto set $\boldsymbol{\mathcal{P}}$ *is the set of solutions that are not dominated by any other solutions in* $\boldsymbol{\mathcal{S}}$*, where* $\boldsymbol{\mathcal{S}}$ *is a subset of* $\boldsymbol{\Omega}$*.*

Definition 2.4.5 (Pareto front)
Given a MOP and the Pareto set $\mathcal{P}$, the Pareto front $\mathcal{PF}$ is defined as:

$$\mathcal{PF} = \{\vec{f}(\vec{u}) | \vec{u} \in \mathcal{P}\}$$

In other words the Pareto front is the output from mapping the Pareto set over the $\vec{f}$ function.

Definition 2.4.6 (Pareto optimal set)
Given a MOP, the Pareto optimal set $\mathcal{P}^*$ is defined as:

$$\mathcal{P}^* = \{\vec{u} \in \Omega | \neg \exists \vec{v} \in \Omega \ \vec{v} \preceq \vec{u}\}$$

In other words the set of solutions that are not dominated by any other solutions in the search space.

Definition 2.4.7 (Pareto optimal front)
Given a MOP and its Pareto optimal set $\mathcal{P}^*$, the Pareto optimal front $\mathcal{PF}^*$ is defined as:

$$\mathcal{PF}^* = \{\vec{f}(\vec{u}) | \vec{u} \in \mathcal{P}^*\}$$

The optimal Pareto front is the best possible answer a multi-objective problem can have.

2.4.4. Pareto Techniques

Some common Pareto techniques used to solve *a posteriori* multi-objective problems are explained in this section. All the following techniques uses Pareto sampling. Pareto sampling works by generating a variety of solutions in a single run. Pareto dominance is used to find the best solutions among them. A problem with multi-objective algorithms that use Pareto concepts is that there is no efficient way to check for non-dominance [Coello et al., 2007, p.111]. The set of solutions is usually visualized as a trade-off curve where each dimension corresponds to an objective as seen in Figure 2.3

Crowding Distance

Crowding distance [Deb et al., 2000] is a measurement of how close a solution is to its nearest neighbors along a Pareto front. A solution with a low crowding distance is close to its neighbors and vice verse. Crowding distance is calculated by individually sorting the solution set for each objective and calculating distance between the fitness values of the neighbors. A visualization of the crowding distance calculation is shown in Figure 2.4, where the solution i's crowding distance is calculated using

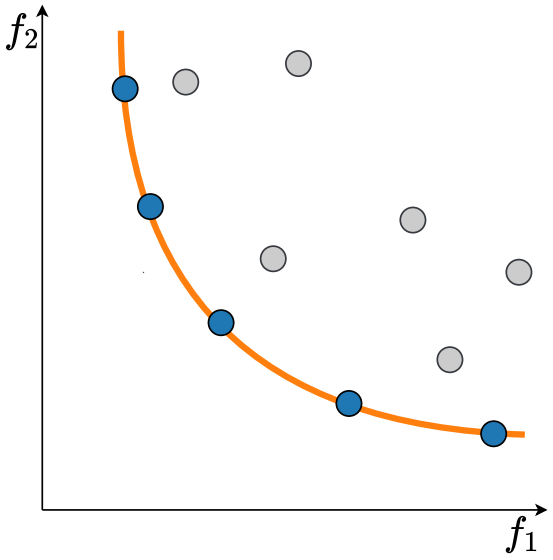


Figure 2.3.: Visualization of solutions for a problem with two objectives. The orange line represents the Pareto optimal front, the blue circles are the best known Pareto set, and the grey circles are dominated solutions.

the two nearest neighbors. The first and last element, the objective bounds, always have a crowding distance of infinity. Calculating the actual crowding distance for a solution is done by summing the normalized crowding distance for each objective as formulated in Equation (2.4).

$$\sum_{j=1}^{m} \frac{{d_j}^i}{f_j^{max} - f_j^{min}} \tag{2.4}$$

To normalize the crowding distance the minimum and maximum fitness of each objective is used. This is to avoid one objective dominating the calculated crowding distance. Crowding distance is usually used to promote a uniformly spread Pareto front by removing solutions with low crowding distance.

Non-dominated Sorting

Non-dominated sorting is a technique to sort multi-objective solutions based on ranking and crowding distance [Deb et al., 2000]. The population is first split into fronts based on their level of non-domination. The first front, consisting of the non-dominated solutions, is created by comparing each member in the population against all others and only including the members that are non-dominated. The members of the first front are then discounted. This procedure repeats until all

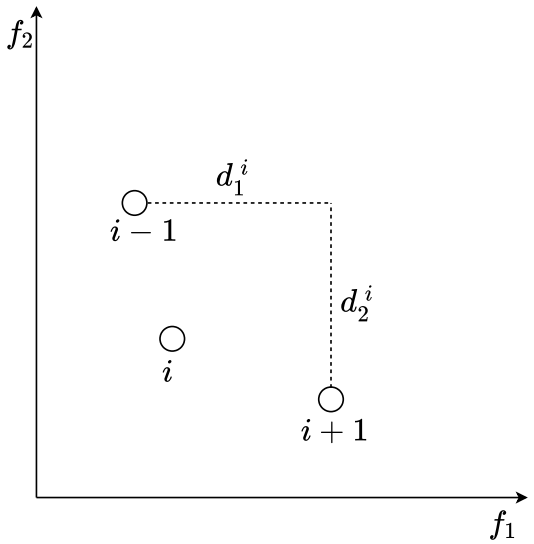


Figure 2.4.: Visualization of crowding distance for solution i

solutions are assigned to fronts. A new set of solutions is formed by including fronts until there is no space left. If the last front included is bigger than the size limit, it is sorted based on crowding distance, from high to low, and the solutions with the highest crowding distance are kept. A graphical example of non-dominated sorting is shown in Figure 2.5 which shows how two populations are combined, sorted into fronts and finally merged back by keeping the least dominated solutions with the highest crowding distance.

Archiving

Archiving is an essential part of many metaheuristic algorithms. It is a broad term, but in general, it means to store solutions in an archive external to the population used in the algorithm. This external archive is then used to guide the solution search and make sure the diversity remains high. Most archive implementations limit the number of solutions stored, and use pruning techniques when the archive has too many solutions. Some common pruning techniques are hypercube fitness [Coello et al., 2004], crowding distance [Sierra and Coello, 2005], ϵ-dominance [Sierra and Coello, 2005] and strength [Zitzler and Thiele, 1999]. Pruning works by removing the worst solution until the archive's size is no longer above the archive limit.

2.4.5. Decomposition

Decomposition decomposes the multi-objective optimization problem into smaller scalar optimization problems. The subproblems can be optimized like single-objective optimization problems and are therefore easier to solve. This way decom-

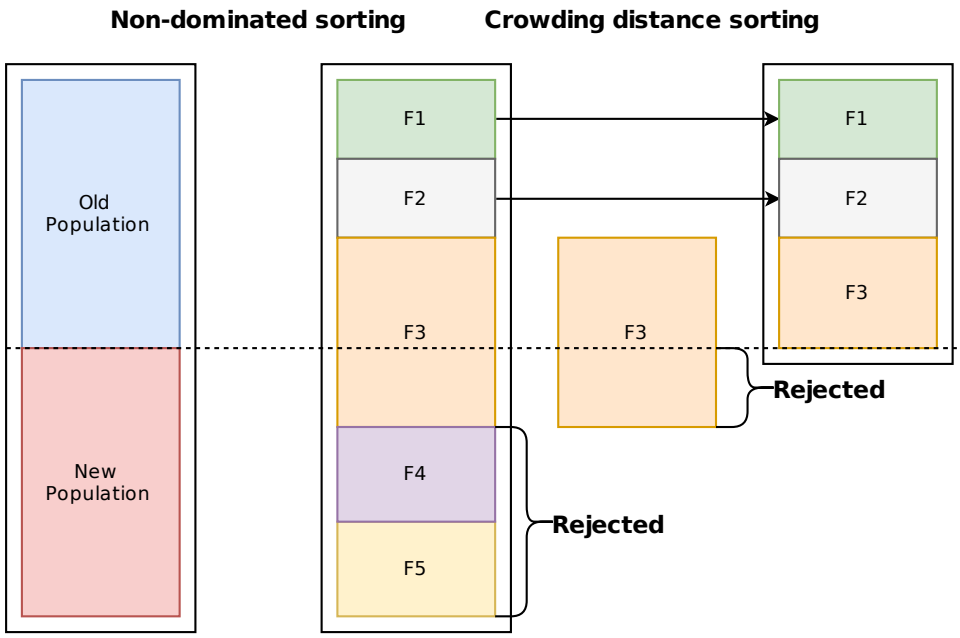


Figure 2.5.: Non-dominated sorting performed on a merged population. A new population is created based on the non-dominated sorted population.

position can handle far more objectives than Pareto based approaches can. Some common decomposition techniques are Tchybycheff and Penalty-based Boundary Intersection [Zhang and Li, 2007]. The strength of decomposition is that it often gives a more diverse set of solutions compared to non-dominated Pareto-based approaches as techniques like crowding distance might not lead to high diversity between all the solutions.

2.5. Single-Objective Algorithms

In this section, some tried and true single-objective algorithms from the literature are described. These algorithms have stood the test of time and many of the techniques used by these algorithms are used by modern single-objective algorithms.

2.5.1. Genetic Algorithm

The Genetic Algorithm [Goldberg and Holland, 1988] is one of the earliest and most well-known metaheuristic algorithms. It is inspired by the process of natural selection in the real world. It is an EA and uses the crossover, mutation, elitism, and selection operators discussed in Section 2.3.1. The solution representation is usually referred to as a chromosome.

2.5.2. Simulated Annealing

Simulated Annealing (SA) [Kirkpatrick et al., 1983] is a single objective optimization algorithm that takes inspiration from the controlled cooling methods used when cooling down metals. By simulating a temperature, that slowly decreases, the algorithm decides if the next solution should be chosen or not. If the next solution is better than the current it is always chosen, but based on the current temperature it might also decide to select a worse solution to improve exploration. With a low temperature, the algorithm behaves more like a Hill Climbing as it only chooses better solutions.

2.5.3. Particle Swarm Optimization

Particle Swarm Optimization (PSO) [Eberhart and Kennedy, 1995] is a population-based metaheuristic algorithm. The solutions in the population are known as particles. Every particle stores a position, velocity, and the personal best-known solution found so far. The algorithm uses this to move the particles around by calculating a new position based on each particle's stored information and the globally best-known position. The algorithm was inspired by how schools of fish and flocks of birds move. PSO has been a popular algorithm ever since its release. Many researchers have built upon the PSO algorithm, creating new and improved versions of it. A multi-objective approach to PSO is covered in Section 2.6.4.

2.6. Multi-Objective Algorithms

This section gives a brief description of some tried and true *a posteriori* multi-objective algorithms. Many of these algorithms introduced techniques that are still used by many of the recently developed algorithms.

2.6.1. Non-dominated Sorting Genetic Algorithm

The original Non-dominated Sorting Genetic Algorithm (NSGA) was proposed by Srinivas and Deb in 1994 [Srinivas and Deb, 1994]. It introduced non-dominated sorting and combined it with a genetic algorithm to create an MOEA. It was quite good for its time but had some shortcomings. It was computationally heavy with a complexity of $O(mn^3)$, where m is the number of objectives and n is the population size. It also did not utilize elitism and required a specified sharing parameter. A new and improved version of the algorithm called NSGA-II [Deb et al., 2000] was created that solves all these issues. Instead of naively finding the rank by checking every solution with all other solutions NSGA-II finds the rank in two steps. In the first step, the domination count and the set of solutions the solution dominates is calculated for each solution. The fronts are then calculated by adding all the

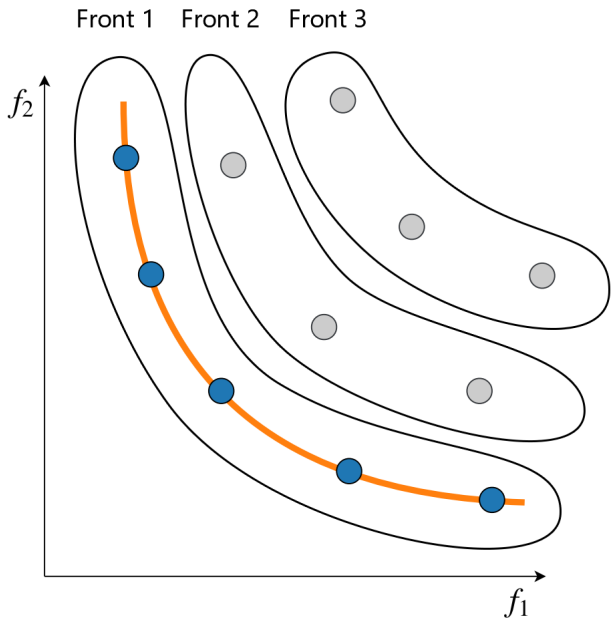


Figure 2.6.: Solutions divided into fronts based on ranks after non-dominated sorting.

solutions that have a domination count of zero, but at the same time decrease the domination counter for all the solutions they dominate. This way the next front is easily found by once again adding the new non-dominated solutions which allow the fronts to be calculated in $O(mn^2)$ time instead. A visualization of the fronts obtained with this procedure is shown in Figure 2.6. Instead of using the sharing function from NSGA-I a new crowding distance operator was adopted which does not require any parameters. NSGA-II is one of the most used multi-objective metaheuristics to this day.

2.6.2. Strength Pareto Evolutionary Algorithm

The Strength Pareto Evolutionary Algorithm (SPEA) [Zitzler and Thiele, 1999] was one of the first stable MOEAs. The algorithm has a population and an archive set. The non-dominated solutions from the population are stored in the archive during each iteration. During this step, solutions that become dominated or are equal to the new solutions will be removed from the archive. Each solution in the archive is assigned a strength value based on how many solutions in the population it dominates or are equal to. This strength value is also used as the solution's fitness value. When calculating the fitness of the solutions in the population, the strength values of the archive solutions that dominate or have equal fitness values

as it are summed up, and then one is added to it. Binary tournaments are used to select parents for the next population. The parents can be chosen from the archive or the population, but there is a higher chance that a solution from the archive will be chosen.

A few years after SPEA was created the original authors created an improved version called SPEA2 [Zitzler et al., 2001]. In the original SPEA solutions that were dominated by the same archive-solutions could have the same fitness. In some cases this could lead to the majority of the population having the same fitness, decreasing the selection pressure. To solve this SPEA2 incorporated a new fitness strategy, based on k-nearest neighbor method, that takes density information into account. If there are too few non-dominated solutions to fill the archive SPEA2 adds dominated solutions to make sure the archive is always the same size. The pruning method that was used to cut solutions when the archive had too many solutions was changed in SPEA2 so that it would not cut boundary points. Another change from SPEA is that it only chooses solutions from the archive during mating selection.

2.6.3. Pareto Archived Evolution Strategy

The Pareto Archived Evolution Strategy (PAES) [Knowles and Corne, 1999] algorithm is a simple and effective algorithm for solving multi-objective problems. Unlike the majority of multi-objective optimization algorithms, it works with a single solution at the time instead of a population. The algorithm uses local search similar to hill climbing to generate new solutions, and stores the non-dominated solutions in an archive. If neither the current solution nor the new candidate solution dominates the other the archive is used to aid the selection. The solution which belongs to the grid-location with the lowest number of solutions is selected to promote diversity over clustered solutions. The archive has a maximum size and if the number of non-dominated solutions in the archive goes above this limit the archive is pruned using a d-dimensional grid, where d is the number of objectives. The solutions are mapped to this grid and when new solutions are added, a solution from the grid-location with the highest number of solutions is removed. It serves as a good baseline for comparison for new algorithms [Knowles and Corne, 1999].

2.6.4. Multi-Objective Particle Swarm Optimization

PSO is one of the most used and well known single objective optimization algorithms. Because of this, there are many multi-objective algorithms based on PSO, one of them being Multi-Objective Particle Swarm Optimization (MOPSO). The MOPSO version initially proposed by Coello et al. [2004] uses an archive similar to the adaptive grid used in PAES. Instead of using a global leader, like in PSO, MOPSO

selects a leader from the archive using hypercubes and fitness proportionate selection. First, the archive is split into hypercubes and then fitness proportionate selection is used to select a hypercube. This hypercube is selected using a fitness calculated based on the number of particles inside that hypercube, favoring hypercubes with few solutions in them. Once a hypercube is selected a random particle is then selected from this cube, which is used as the leader. The archive is updated by only storing the non-dominated solutions. When the archive goes over the size limit particles are pruned by removing a random solution from the hypercube with the worst fitness, which increases the chance that crowded areas of the archive are selected. A random solution from the selected hypercube is then removed and this process is repeated until the archive size is reached. In addition to the leader, PSO also keeps track of the personal best position a particle has had. MOPSO also does this, but since each particle has multiple fitness values domination checking is used instead of pure value comparison. To prevent the algorithm from prematurely converging a new mutation operator is introduced. The mutation operator is designed to mutate aggressively in the first iterations and then decrease rapidly during the execution. Each particle in the population has a chance for mutation calculated based on the current iteration. If a solution is selected for mutation only a randomly selected decision variable is mutated. The new decision variable is calculated by using the lower and upper bound of the search together with the mutation chance. Since the mutation chance decreases during the run the range of the new decision variable also decreases during the run.

2.6.5. Multi-Objective Evolutionary Algorithm Based On Decomposition

The Multi-Objective Evolutionary Algorithm Based On Decomposition (MOEA/D) [Zhang and Li, 2007] focuses on decomposition. Decomposition is something that has been used a lot in mathematical programming when solving multi-objective problems but had not been utilized to the same degree for MOEAs at the time. The authors wanted to change this and proposed an MOEA that used decomposition as an alternative to the more common Pareto-based MOEAs. The algorithm decomposes the multi-objective problem into many scalar optimization problems that it can solve simultaneously. Every solution in the population is associated with one of these subproblems, and a neighborhood relationship is created based on the distance between the weight vectors. When optimizing a subproblem, the algorithm only uses information from the subproblem's neighbors, to reduce the computational complexity. MOEA/D uses genetic operators like selection, crossover, and mutation to produce new solutions.

2.7. Test Functions

When testing optimization algorithms, test functions are often used. These test functions provide a good way of finding the strengths and weaknesses of an algorithm in a controlled setting. Test functions are implemented as mathematical functions which are often computationally inexpensive, and the global and local optima are known before starting. Properties like these make them ideal for testing, as it is easier to analyze the results. However, solving real problems is more challenging than solving test functions [Mirjalili et al., 2017]. There are test functions for both single-objective and multi-objective algorithms. Test functions are used as black box problems, which means that the algorithm does not know anything about the problem beforehand and should not be guided based on knowledge about the test function.

2.7.1. Visualization

Single-objective test functions are usually visualized with a three-dimensional plot, using two decision variables and the fitness value. An example of this can be seen in Figure 2.7 where the fitness is visualized using a colormap to show height differences. Most test functions have a scalable number of decision variables, and often 30 variables are used when testing. A large number of decision variables makes it a lot harder to visualize the problem as only two decision variables can be plotted at a time.

For multi-objective optimization, the Pareto front found by the algorithm is usually visualized together with the Pareto optimal front for the test function. This can be seen in Figure 2.3. Unlike single-objective test functions, the input of the test functions is rarely plotted as the front represents the fitness values obtained. The Pareto optimal front for multi-objective test functions is calculated using high-performance parallel computers by computing every possible combination of decision variables [Coello et al., 2007, p.179].

2.7.2. Types of Test Functions

To test different types of problems test functions emulate different problem characteristics. Typical characteristics used are unimodal vs. multimodal, convex vs. concave, and differentiable vs. non-differentiable [Coello et al., 2007, p.177]. Test functions are often combined into composite functions to test a combination of problems. Additionally, since many single-objective test functions reaches the optimal fitness value when all decision variables are 0, it is common to shift and rotate the input to avoid bias.

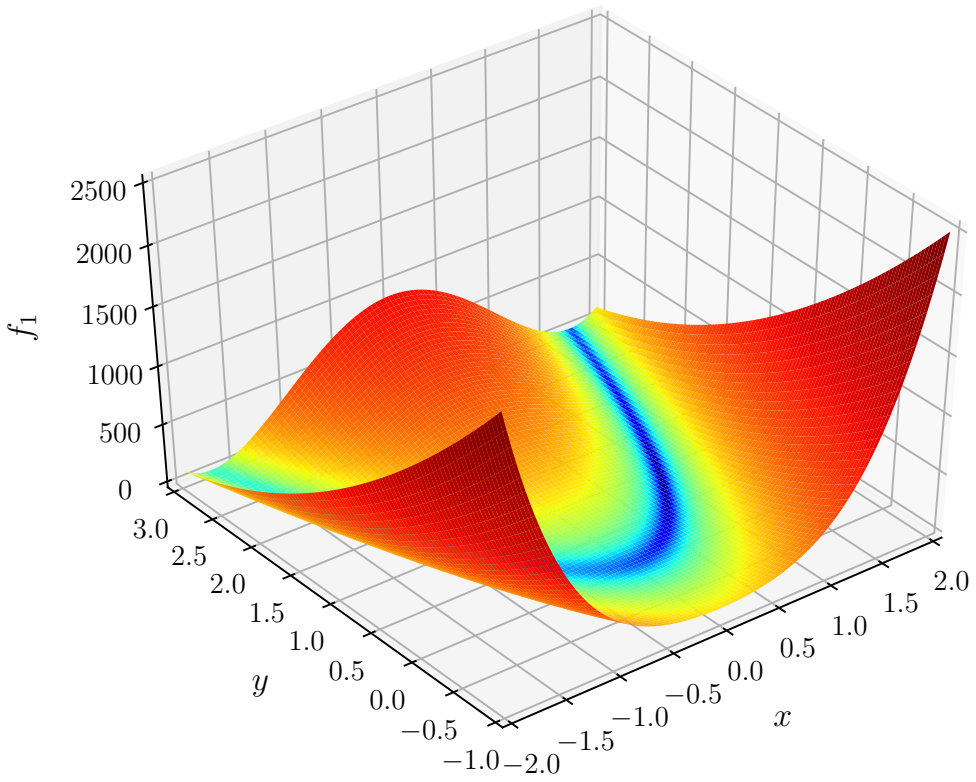


Figure 2.7.: The Rosenbrock test function with two decision variables. The x and y axis represents positions in the search space. The z axis is the fitness value

2.7.3. Test Suites

To ensure that testing is performed on a variety of test functions with different characteristics it is common to use a set of test functions referred to as a test suite. The suite will commonly include a mix of different characteristics to make it easier to see what types of problems the algorithm handles well and where there is room for improvement. The following section covers some of the most used test suites from the literature.

ZDT

The ZDT test suite [Zitzler et al., 2000] was one of the first multi-objective test suites. The name is based on the first letter of the last name of the creators. The test suite has six test functions, all with two objectives. ZDT5 stands out, as it is a binary problem, and is therefore often not included when using the test suite.

DTLZ

DTLZ [Deb et al., 2005] is a scalable test suite. It was created when the authors felt that MOEAs performed well enough on problems with two objectives, and wanted to create a test suite where the number of objectives was scalable. The three authors of ZDT were also involved in the creation of DTLZ, and it follows the same naming pattern. The original test suite consists of 9 test functions where DTLZ1-7 are unconstrained, while DTLZ8-9 are constrained.

UF

The UF test suite is a set of 13 test functions that was created for the IEEE Congress on Evolutionary Computation in 2009 (CEC2009) [Zhang et al., 2008]. UF stands for Unconstrained Functions since CEC2009 also contains a set of constrained functions. The test suite was created to resemble complicated real-life problems. UF1-7 have two objectives, UF8-10 have three objectives and UF11-13 has five objectives.

CEC2014

The CEC2014 test suite [Liang et al., 2013] is based on a collection of well known single-objective test functions from the literature. The test suite contains three unimodal functions and 13 simple multimodal functions. Based on these functions there are six hybrid functions and eight composition functions. The test functions are sometimes rotated or shifted or both.

2.8. Multi-Objective Performance Metrics

A posteriori multi-objective algorithms produce a set of solutions. To access the quality of these solutions, performance metrics are used. These performance metrics give a numerical value representing the quality of the solution. This allows whole solution sets to be compared with each other. The two essential measurements in multi-objective optimization are convergence and diversity. Some metrics that are used to measure convergence against the Pareto optimal front are Generational Distance (GD), and γ (Convergence) [Deb et al., 2000]. Metrics for measuring diversity are Spacing [Veldhuizen and Lamont, 2000] and Δ (Spread) [Deb et al., 2000]. Additionally, Hypervolume (HV) and Inverted Generational Distance (IGD) measures both convergence and diversity as a single metric. Some of the most common performance metrics are described in detail below.

2.8.1. Generational Distance

Generational Distance (GD) [Veldhuizen and Lamont, 2000] works by measuring the distance from all the solutions in the known Pareto front to the Pareto optimal front. The resulting value represents how well the known Pareto front has converged. In the literature, there are two versions of GD used. One version sums the distances and divides by the number of solutions in the known Pareto front. The second version, displayed in Equation (2.5), adds the distances using Euclidean norm by summing the squared distances and then takes the square root of this sum. The values produced by the different versions are not directly comparable.

$$GD(\mathcal{A}, \mathcal{PF}^*) = \frac{\sqrt{\sum_{p \in \mathcal{A}} d(p, \mathcal{PF}^*)^2}}{|\mathcal{A}|} \tag{2.5}$$

Where $\mathcal{A}$ is the Pareto front found by the algorithm and $\mathcal{PF}^*$ is the Pareto optimal front on the problem. d is the Euclidean distance between the solution in the known Pareto front and the closest solution in the Pareto optimal front. The optimal value of GD is 0 when all the solutions in $\mathcal{PF}^*$ overlap with $\mathcal{A}$. It measures how far the known Pareto front is from the Pareto optimal front.

2.8.2. Inverted Generational Distance

Inverted Generation Distance [Coello and Sierra, 2004], as its name implies, is just an inverted version of Generational Distance. By measuring the Euclidean distance from the Pareto optimal front IGD also measures whether the known Pareto front covers the whole Pareto optimal front. To obtain a measure of 0 the known Pareto front has to be evenly spread out over the Pareto optimal front and measures in this way both convergence and diversity. Precisely like the GD performance metric, there are two versions of IGD as well, which affects the calculated value. The Euclidean norm version is shown in Equation (2.6).

$$IGD(\mathcal{A}, \mathcal{PF}^*) = \frac{\sqrt{\sum_{p \in \mathcal{PF}^*} d(p, \mathcal{A})^2}}{|\mathcal{PF}^*|} \tag{2.6}$$

2.8.3. Hypervolume

The hypervolume indicator [Zitzler and Thiele, 1999] measures the volume of the subspace that is dominated by the Pareto front obtained using Equation (2.7).

$$HV(\mathcal{A}) = \mathrm{VOL}\Big(\bigcup_{p \in \mathcal{A}} [f_1(p), z_1^r] \times \ldots \times [f_m(p), z_m^r] \Big) \tag{2.7}$$

A reference point $z^r = (z_1^r, \ldots, z_m^r)^T$, where m is the number of objectives, is used to bound the objective space. Unlike IGD and GD hypervolume is defined for maximization problems, but works on minimization problems if the Pareto front is inverted prior to hypervolume calculation. VOL is the Lebesgue measure, which is an n-dimensional volume measure. A value of one indicates that the front dominates the entire search space. The volume obtained is bounded by the Pareto optimal front, and a measure of 1 is unlikely to be obtained. If both the known Pareto front and the Pareto optimal front measures the same volume the known Pareto front is as close as possible to the Pareto optimal front.

3. Structured Literature Review

To help answer the research questions a structured literature review was performed. Research questions explicitly made for the literature review search, called search research questions, are presented. Following is a summary of the related work and a discussion of the related work in light of the search research questions.

3.1. Structured Literature Review Protocol

A structured literature review has been performed to find the state of the art. The review follows the structure described in Kofod-Petersen [2014]. Using a structured literature review leads to a more thorough selection process, which results in higher quality related work.

3.1.1. Search Research Questions

A set of research questions for guiding the literature search was created. To differentiate between the main research questions and the research questions used here they will be referred to as Search Research Question (SRQ). **SRQ1-3** are related to **RQ1-3**, while **SRQ4** and **SRQ5** have been created for this literature review.

SRQ1 *Which single-objective algorithms exist that are suitable for multi-objective extension?*

Promising new single-objective algorithms are created all the time. Finding several state of the art single-objective algorithms is, therefore, necessary to make a solid choice of an algorithm to extend. For an algorithm to be considered suitable for multi-objective extension it should perform well on a range of optimization problems and use techniques that allows it to be extended without changing the core functionality of the algorithm.

SRQ2 *Which multi-objective techniques are used when extending a single-objective algorithm?*

The aim of this question is to learn more about which techniques are used when extending algorithms to handle multiple objectives, and learn how these techniques work. Getting an overview of these techniques is important to be able to make an informed decision when deciding which technique(s) to use when extending the selected single-objective algorithm.

SRQ3 *Which multi-objective algorithms should be compared with when evaluating an algorithm?*

Getting an overview of the state of the art multi-objective algorithms is essential to be able to assess the quality of the final version of the algorithm that will be created. Details like which algorithms they compare their results with and how the experiment is defined are important to research before determining the experimental setup for the multi-objective comparison experiment.

SRQ4 *Which test functions are used to test multi-objective algorithms?*

This question aims to give an overview of which test functions are most commonly used when running experiments with multi-objective algorithms. When running experiments, it is important to use test functions that test the algorithms thoroughly. Additionally, using well-known test functions can make it easier to compare results with other articles.

SRQ5 *Which performance metrics are used to measure multi-objective algorithms?*

To numerically compare multi-objective algorithms performance metrics are needed. It is therefore essential to know which performance metric or combination of performance metrics can most accurately assess the performance of the algorithms. It is also important to find which performance metrics are most used in the literature, as this can allow results to be compared with other articles.

The process of finding the related work using SLR is explained in Appendix A.

3.2. Related Work

In this section, the algorithms from the 16 articles that were gathered in the SLR are categorized and summarized.

3.2.1. Single-Objective Algorithms

Some of the articles present new single-objective metaheuristics for solving optimization problems. A summary of each algorithm is presented here.

Animal Migration Optimization

Animal Migration Optimization (AMO), is a swarm algorithm inspired by animal migration behavior, is presented in Li et al. [2014]. It consists of two phases: the migration phase and population updating phase. The migration phase simulates how animals move together in groups. It is based on three rules observed in migrating animals: they move in the same direction as their neighbors, remain close to their neighbors and avoid collision with their neighbors. The animals are structured in a ring topology, and during the migration phase, they move based on the position of the animals in their neighborhood. An animal's neighborhood contains itself as well as the two closest neighbors on each side. The neighborhood size is, therefore, always five. Once every animal has been moved, the fitness values of every animal's old and new position are compared, and the animal keeps the position with the best fitness score.

The population updating phase simulates how animals leave and join the group during migration. In this phase, new animals are created using the position of the existing population. Like in the migration phase, each new animal is based on one of the animals in the population. When creating a new animal, a probability is used to decide if a decision variable should be copied from the animal it is based on. Otherwise, it is created using the best solution, the animal it is based on, and two other animals selected randomly from the population. This probability is based on how good the original animal's fitness is compared to the rest of the population, giving animals with good fitness lower probability of creating new decision variables. As in the migration phase, the new animal's fitness is compared with the old one. The animal with the best fitness stays in the population. For every iteration the migration phase is ran first, followed by the population updating phase.

Dandelion Algorithm

The Dandelion Algorithm (DA) [Gong et al., 2017] is inspired by the sowing of dandelions. The algorithm has a mother dandelion that spreads seeds in an area around it. When calculating the radius of the seeds, the algorithm checks if the fitness of the mother dandelion from the current generation is better than the fitness from the previous generation. If it does not find a better solution, the radius decreases and if it finds a better solution, the radius increases. To keep the diversity high among the seeds the algorithm generates a small number of self-learning seeds each generation to avoid local optima. For selection, the algorithm always keeps

the seed with the best fitness as the mother dandelion for the next generation. When creating a seed, DA only changes one decision variable for each seed.

Earthworm Optimization Algorithm

Earthworm Optimization Algorithm (EWA) [Wang et al., 2015] is inspired by earthworm behavior in nature. The algorithm's structure is similar to GA, but has some significant changes to the reproduction method. The reproduction method is based on two different types of reproduction used by earthworms. Reproduction type 1 generates only one offspring by itself, while reproduction type 2 generates one or more offspring at the same time. The paper presents nine different crossover operators to perform reproduction type 2. A weighted summation of the offspring is used to generate the final earthworm for the next generation. To escape from local optima Cauchy Mutation is used. EWA uses generations with elitism by passing on the individuals with the highest fitness.

Lion Optimization Algorithm

The Lion Optimization Algorithm (LOA) [Yazdani and Jolai, 2016] is based on lions' social organization into residents and nomads, and their mating patterns. The initial population consists of randomly generated solutions referred to as lions. A given percentage of the lions are selected as nomad lions while the remaining lions are partitioned into subsets called prides. A given sex rate decides the percentage of female lions in the prides, while for nomad lions it decides the percentage of male lions. Based on lion hunting patterns opposition-based learning is utilized to simulate hunting. Hunters use a dummy prey (center of lions) to hunt, and by encircling the prey lions can escape local optima. The remaining female lions in the pride move towards what is called a safe place. This safe place is selected by performing tournament selection on the set of best solutions found by the members of the pride. Male lions in the pride perform a local search by roaming towards a randomly selected part of the territory. Nomad lions roam randomly in the search space by mutating decision variables. LOA uses mating to exchange information between lions similar to crossover in EAs. In prides a percentage of female lions mate with one or several males in the pride, while for nomad lions the females only mate with one male which is selected randomly. Offspring are created using linear combination with mutation. LOA also has a mechanism to move lions in and out of prides based on fighting between lions and migration.

Moth Search

Wang [2016] presents Moth Search (MS), an algorithm based on how moths tend to fly closer to a light source with paths resembling spirals. The current best moth is considered as the light source to guide the other moths closer to the current best

solution. It also uses Lévy-flights, a class of random walk, as moths have shown to use this pattern when flying close to the light source. By using Lévy-flights, the algorithm performs exploration and exploitation in the search space. Elitism is also employed to accelerate convergence.

Symbiotic Organisms Search

The article by Cheng and Prayogo [2014] describes a new single-objective algorithm called Symbiotic Organisms Search (SOS) which simulates the symbiotic interaction strategies adopted by organisms to survive and propagate in the ecosystem. SOS is a population-based algorithm that uses three different symbiosis phases to guide the population during optimization: mutualism, commensalism, and parasitism. The first phase is the mutualism phase which selects two random individuals from the population. The mutual vector between the individual and the helping individual are calculated, and then the difference between the globally best-known individual and the mutual vector is used to calculate new positions. The new position is only used if the fitness improved, a form of exploitation. The next phase is commensalism; this stage is similar to mutualism except that only the first individual selected benefits from the interaction. The last stage of SOS is parasitism in which one individual benefits from the interaction and another suffers. By randomly selecting decision variables to mutate a parasite vector is created by duplicating the first individual. If the new parasite vector has better fitness than the second individual, which is also randomly selected, it is killed off and replaced by the parasite and vice verse.

3.2.2. Non-dominated Sorting Based Algorithms

Some of the multi-objective algorithms in the literature review are based on non-dominated sorting used by NSGA and are described in this section.

Non-dominated Sorting Invasive Weed Optimization

Nikoofard et al. [2012] present Non-dominated Sorting Invasive Weed Optimization (NSIWO). NSIWO is based on Invasive Weed Opimization (IWO) [Mehrabian and Lucas, 2006] which is a single-objective algorithm inspired by colonizing and reproduction in weeds. IWO uses reproduction, spatial dispersal, and competitive exclusion. Like many other nature-inspired algorithms, it is population and generational based, and starts with a randomly generated population. A new generation is created based on seeds from the members of the population. Each member produces seeds relative to their fitness with the worst member producing fewer seeds than the best member. The seeds are then scattered randomly in the search space around the members. The newly produced seeds and their parents are then combined into a new generation using competitive exclusion where the

fittest members survive. NSIWSO extends IWO with concepts from NSGA-II. Binary tournament selection is used to select candidate parents from the population which it stores in an archive. A new population is then generated by using seed reproduction and seed dispersal. This new population is combined with the old population and then non-dominated sorting is performed. Like NSGA-II they also assign crowding distance to all individuals in the combined population. Weakness, a value opposite of fitness, for all individuals is calculated using the rank and crowding distance which is used to eliminate the weakest individuals. This process repeats for a set amount of iterations.

Non-dominated Sorting Moth-Flame Optimization

The algorithm extends an existing single-objective algorithm called Moth-Flame Optimization (MFO) [Mirjalili, 2015b]. The single-objective algorithm is, like Moth Search, inspired by moths' spiral movement around light sources. A light source, or flame, is one of the best solutions found so far by the moths. These flames are continuously updated when the moths find better solutions. Moths move by spiraling around the closest flame. During execution, the number of flames is gradually decreased. Non-dominated Sorting Moth Flame Optimization (NS-MFO) [Savsani and Tawhid, 2017] uses, as its name implies, non-dominated sorting and crowding distance from NSGA-II to handle multiple objectives. First, it generates a random population, then rank and crowding distance are assigned to the population. NS-MFO uses MFO to create a new generation based on the previous generation, which merges with the previous population. Non-dominated sorting and crowding distance is again applied to the combined population, and then the worst half of the population is killed off. This process repeats for a given number of generations. In other words, NS-MFO is like NSGA-II except that the GA is replaced with MFO.

3.2.3. Archive Based Algorithms

Many of the multi-objective algorithms in the literature review are based on some kind of archive technique for handling multiple objectives. The different implementations are described in this section.

Archived Multi-Objective Simulated Annealing

Archived Multi-Objective Simulated Annealing (AMOSA) [Bandyopadhyay et al., 2008] is based on Simulated Annealing and extends it with an archive to solve multi-objective optimization problems. The algorithm stores new solutions in the archive until a set soft limit (SL) is reached and will then reduce them down to a hard limit (HL) if they exceed the SL. The SL acts as a buffer before the population is reduced to the HL to avoid performing excessive pruning. On initialization, the number

of solutions created is relative to the SL. Hill climbing using domination is then performed as a greedy optimization. Only the solutions that are non-dominated are added to the archive. If the archive is larger than the HL, it is reduced using a clustering technique called single linkage algorithm [Jain and Dubes, 1988]. The acceptance of a new solution is extended from the temperature based method used in SA. AMOSA also considers the amount of domination when calculating the probability of accepting a new solution. Like SA, the temperature cools down during the AMOSA process which lowers the chance of accepting suboptimal solutions. In other words, AMOSA can, unlike many other multi-objective algorithms, select dominated solutions to search with.

Multi-Objective Ant Lion Optimizer

Multi-Objective Ant Lion Optimizer (MOALO) [Mirjalili et al., 2017] is based on Ant Lion Optimizer (ALO) [Mirjalili, 2015a], an algorithm that mimics the hunting mechanics of antlions and their interaction with ants. ALO operates on two populations: a set of ants and a set of antlions. The ALO algorithm uses several concepts from how ants and ant lions behave in real life: random walk of ants, entrapment in antlion pit, constructing a pit, sliding ant towards ant lions, catching prey, and reconstructing the pit. ALO does this by pairing each ant with an antlion which keeps track of the currently best solution found by that ant. A pit is constructed which slides the ant towards the best-found solution. The pit is reconstructed by the antlion when an improved solution is found. Roulette wheel selection based on fitness is used to pair ants with antlions (pits) to mimic how bigger ant lions produce bigger pits in nature. Additionally, an elite antlion (ant lion with the best fitness) will always affect all ants to converge ants towards the globally best-known solution. To extend ALO to multi-objective MOALO uses archive maintenance and leader selection from MOPSO. It also uses niching to measure the distribution of the solutions in the archive. Antlions are selected from the solutions with the least populated neighborhood to encourage searching of non-crowded areas. When the archive is full, the solutions with the most populated neighborhood are removed from the archive.

Multi-Objective Dragonfly Algorithm

Inspired by the static and dynamic swarming behaviors of dragonflies the Dragonfly Algorithm (DFA) [Mirjalili, 2016] was created. In the same article, the algorithm is extended to handle multiple objectives (MODFA) and binary problems. Static swarming is used while hunting and is performed by flying over a small area in abrupt movements. The dragonflies create sub swarms and as a result of hunting for prey, which are considered promising solutions, perform exploration of the search space. Dynamic swarming, on the other hand, is a massive number of

dragonflies whom all move in the same direction which results in exploitation. The general behavior of dragonflies in the DFA comes from five patterns: separation, alignment, cohesion, attraction, and distraction. These are all mathematically modeled and are ultimately used to update the dragonfly's position using a step vector. The transition from exploration to exploitation is performed by increasing the search radius proportional to the number of iterations. The neighborhood for each dragonfly also increases, and at the end of the optimization all dragonflies will end up in the same neighborhood which converges against the globally best-known solution. As mitigation against local optima dragonflies use Lévy flights when there are no neighboring solutions. To support multiple objectives a couple of changes were needed. Finding neighbors is extended to check multiple objectives, and food sources are selected from the least populated area of the obtained Pareto front in the archive. Enemies are also selected from the archive, but this time from the most populated part to discourage searching crowded areas.

Multi-Objective Grey Wolf Optimizer

The Multi-Objective Grey Wolf Optimizer (MOGWO) [Mirjalili et al., 2016] takes a single-objective optimization algorithm named Grey Wolf Optimizer (GWO) and extends it to handle multi-objective problems. The hunting behavior of grey wolves inspires the GWO algorithm. The three best wolves are referred to as the alpha, beta, and delta wolf. The rest of the candidate solutions follow these three leaders and are referred to as the omega wolves. When searching the algorithm simulates the encircling behavior grey wolves use around its prey. The conversion of GWO to multi-objective was done by adding two new components. The first was a hypercube based archive of non-dominated solutions directly based on the archive MOPSO uses. The second component they added was a new selection strategy. The single-objective algorithm uses the alpha, beta and delta wolf to guide the other wolves towards promising regions of the search space. Since the three best wolves no longer can be selected based on fitness alone when there are multiple objectives, the algorithm chooses three solutions from the least crowded segments of the archive using the roulette-wheel selection method.

Multi-Objective Whale Optimization Algorithm

Multi-Objective Whale Optimization Algorithm (MOWOA) [Kumawat et al., 2017] is based on the humpback whale inspired Whale Optimization Algorithm (WOA) [Mirjalili and Lewis, 2016]. WOA uses principles such as encircling prey, bubble net attacking, and searching for prey. Bubble net attacking exploits the globally best-known solution by using two phases: a shrinking encircling mechanism and spiral updating position. Since they are just different kinds of movement towards the prey, the phase is selected randomly. A sort of random walk is performed

while searching for the prey to explore the search space. MOWOA uses these same techniques but also uses an archive to solve multi-objective problems. Non-dominated solutions are identified by an archive controller, and the best solutions are stored in an archive repository. The archive is from MOPSO.

Speed-constrained Multi-objective PSO

Speed-constrained Multi-objective PSO (SMPSO) [Nebro et al., 2009a] is an improvement of OMOPSO, a PSO based multi-objective algorithm, by Sierra and Coello [2005]. The main differences between OMOPSO and SMPSO are speed constraint using a constriction coefficient, different ranges for the constants C_1 and C_2, and a polynomial mutation operator instead of the novel mutation operator in MOPSO. The polynomial mutation operator has a high chance of producing values close to the origin position with a user-defined chance of mutating one or more decision variables. SMPSO uses one of the two archives from OMOPSO. This archive stores non-dominated solutions. When the archive has too many solutions, it prunes based on crowding distance. When selecting a leader from the archive, binary tournament selection is used based on crowding distance.

3.2.4. Other Multi-Objective Algorithms

The rest of the multi-objective algorithms based on neither non-dominated sorting nor archive are presented here.

Modified Dual-Population Paradigm

This article modifies the Dual-population Paradigm (DPP) algorithm, creating a new version called DPP2 [Bui et al., 2017]. The goal of the authors was to create an algorithm that focuses both on diversity and convergence. To achieve this, the algorithm keeps two populations, one Pareto-based for convergence and one decomposition-based for diversity. DPP2 always chooses one parent from each population when creating offspring to keep both diversity and convergence for the next generation. By using Restricted Mating Selection (RMS), the two populations can cooperate to find solutions. DPP2 introduces a new type of RMS.

Multi-Objective Evolutionary Algorithm based on Dominance and Decomposition

The Multi-Objective Evolutionary Algorithm based on Dominance and Decomposition (MOEA/DD) [Li et al., 2015] focuses on solving optimization problems with many objectives. When there are a large number of objectives, the majority of solutions within a population becomes non-dominated with each other. The amount of non-dominated solutions makes it hard to use standard non-domination

approaches when there are many objectives. The algorithm focuses on both decomposition and dominance and is inspired by MOEA/D (decomposition) and NSGA-III (dominance). Similar to MOEA/D the authors used weight vectors to help guide the selection procedure. It also used the same neighborhood concept and applied a scalarization function when measuring the fitness of solutions. Similar to NSGA-III the authors divide the solution into non-domination levels based on their Pareto dominance relation.

3.3. Discussion

In this section, the related work is discussed in light of the SRQs. Each SRQ has its own section where relevant points are discussed before a conclusion for the SRQ is made.

3.3.1. Single-Objective Algorithms

SRQ1 *Which single-objective algorithms exist that are suitable for multi-objective extension?*

When evaluating which single-objective algorithm is the most suitable for multi-objective extension several attributes needs to be taken into consideration. The following list contains some of the attributes that are essential to evaluate when considering an algorithm for extension:

- **Performance:** The algorithm must perform well when compared with other state of the art algorithms. To be able to assess the quality of an algorithm it is important that it was tested thoroughly against other competitive algorithms on a wide variety of test functions.
- **Tuning parameters:** Multi-objective techniques can introduce new tuning parameters, and if the algorithm already has many tuning parameters it might make the algorithm hard to tune.
- **Convergence:** If a single objective algorithm converges rapidly it might converge to a local optimum when solving MOPs.

Performance

To assess the performance of the algorithms all the articles use test functions that are known from the literature. As none of the single-objective algorithms in this literature review are directly compared with each other in any of the articles, a "best" algorithm cannot be decided purely based on the experiment results. Some of the algorithms are tested on the same test functions, but settings like the number

of decision variables, upper and lower bound, and maximum function evaluations are different from article to article. The majority of the articles compare with some tried and true algorithms, as well as some state of the art algorithms. All the algorithms in the literature review do well in their own experimental testing, but there are some differences in how thoroughly the algorithms are tested. DA is only tested on 12 different test functions. This is significantly lower than the other articles, the second lowest being AMO with 23. The majority of the algorithms are tested on between 25 and 30 test functions, with EWA having vastly more than the other articles with 48. The majority of the articles use the number of function evaluations as a stopping criterion, ranging from 50,000 to 500,000 function evaluations.

Parameters

Looking at the algorithms in this literature review, AMO and SOS are the only algorithms with only one parameter. They only need the population size to be specified. To be able to still achieve good results on a wide variety of problems, these algorithms are created to handle general problems well without changing anything. This makes these algorithms a lot easier to use. EWA and DA are in the middle with (3-4) tunable parameters, allowing for some tuning. MS and LOA have a lot of tuning parameters (5-7). There are both positives and negatives with having many parameters in an algorithm. On one hand, it allows for fine tuning for a specific problem. This can lead to better results for this problem. On the other hand, this kind of tuning can lead to a loss of generality, making the algorithm require a lot of tuning work for each problem.

Having many parameters is also not ideal when extending the algorithm to multi-objective, as this can add additional parameters to the algorithm, making the tuning job even more complicated. A risk with having too few tuning parameters is that the algorithm might not work as well on multi-objective problems. One can then try to change some of the inner workings of the algorithm to perform better on multi-objective problems, but as the algorithm was not designed to be flexible in this way it might not perform well.

Convergence

Fast convergence towards the global optimum is a good characteristic of single-objective algorithms, but if the algorithm converges too fast, it might prematurely converge and end up in a local optimum. When extending an algorithm to handle multiple objectives it might lead to a local Pareto optimal front if the convergence speed is too high [Coello et al., 2004, p. 261].

Most of the algorithms in the literature review have included convergence graphs, with the exception of LOA. AMO, EWA, and DA show a fast convergence without getting stuck in a local optimum, but MS sometimes converges prematurely. Unlike the other algorithms, SOS only shows a convergence graph for an engineering problem, but the convergence seems similar to the compared algorithms. It might be necessary to introduce additional mutation to ensure exploration if an algorithm converges to a local Pareto optimal front [Coello et al., 2004, p. 261].

Conclusion

Because none of the algorithms can be directly compared based on the results from the articles as discussed in the performance section, there is not enough information to conclude which algorithm is best suited for extension based on the articles alone. Because performance is an important part of the selection process, the most promising algorithms are implemented and evaluated in Chapter 5.

3.3.2. Multi-Objective Techniques

SRQ2 *Which multi-objective techniques are used when extending a single-objective algorithm?*

When researching multi-objective techniques used for extension it is important to look at what challenges they might present during implementation and how they perform. The following points are considered when evaluating the different techniques:

- **Convergence:** How the Pareto front obtained converges against the Pareto optimal front.
- **Diversity:** How good the spread of solutions is compared to the Pareto optimal front.
- **Difficulty of implementation:** A more complex algorithm increases the chance of implementation errors and makes it harder to understand what might have gone wrong.
- **Computational and space complexity:** Techniques with high complexity will slow down as the number of objectives and the population size increase.

Non-dominated Sorting

Non-dominated sorting uses techniques that can help it achieve good convergence as well as good diversity. Crowding distance ensures that the population remains

diverse by preferring non-crowded solutions, while sorting into fronts based on rank ensures that the least dominated solutions survive to encourage convergence towards the Pareto optimal front. The implementation of non-dominated sorting is not very complicated as it requires no specific tuning and often can be added on top of an algorithm without any specific tweaks to it. The complexity of non-dominated sorting depends on the implementation. Zhang et al. [2015] describe an implementation with a best case of $O(mn \log n)$, average case of $O(mn^2)$ and space complexity of $O(1)$. NSGA-I and NSGA-II respectively have a best case of $O(mn^3)$ and $O(mn^2)$. Crowding distance has a computational complexity of $O(mn \log n)$ for assignment and $O(2n \log(2n))$ for sorting [Deb et al., 2000].

External Archive

Common for most archives is that they are used to guide the search by selecting different leaders for each iteration. As most archives store non-dominated solutions, selecting leaders from this archive will help the algorithm converge. Selection techniques are often used to help the algorithm pick solutions in the least populated areas to encourage diversity. The most common type of external archive used in the articles from the literature review is an archive first described by Coello et al. [2004] and was used in MOPSO. See Section 2.6.4 for more details about MOPSO. One of the reasons it is so popular is because it is easy to extend a single-objective population-based algorithm with this type of archive. The algorithms from this literature review that are based on MOPSO's archive are MODFA, MOWOA, MOGWO, and MOALO. This type of archive has a complexity of $O(mn^2)$ where m is the number of objectives, and n is the archive size. It is common to limit the archive size, mainly due to practical reasons related to scaling [Reyes-Sierra et al., 2006, p.293]. SMPSO's archive is one of the two archives used by OMOPSO. This archive uses crowding distance and recalculates it every time a solution is pruned, so removing one solution has a computational complexity of $O(mn \log n)$. AMOSA's archive, on the other hand, does not use the archive to guide the search. The archive is only used to keep track of all the non-dominated solutions and is pruned using a clustering technique called single-linkage. The complexity of the clustering technique is $O(SL^2 log(SL))$ where SL is the soft limit used by the archive.

Decomposition

Two of the papers in the literature review use decomposition. Unlike archives and non-dominated sorting, decomposition is more tightly integrated into the algorithm. DPP2 and MOEA/DD are both based on the decomposition techniques described in MOEA/D which have a computational complexity of $O(mnt)$ where t is the number of neighbors considered when optimizing a subproblem, m is the number

of objectives, and n is the number of weight vectors (population size) [Zhang and Li, 2007]. By splitting the MOP into subproblems diversity is encouraged as each subproblem is regarded as a solution. Convergence is handled by minimizing each subproblem. While DPP2 uses the same decomposition method as MOEA/D, weighted Tchebycheff, MOEA/DD uses Penalty-based Boundary Intersection. Both the decomposition based algorithms can be modified to use any decomposition technique used for classic multi-objective optimization.

Conclusion

Both non-dominated sorting and an external archive are techniques that can relatively easily be added on top of the single-objective algorithm to guide the process. Decomposition, on the other hand, is more involved and requires more significant changes to the single-objective algorithm. The average computational complexities of non-dominated sorting and MOPSO's external archive are the same, but the decomposition technique used by MOEA/D is better depending on the population size and number of neighboring sub-problems.

3.3.3. Multi-Objective Algorithms for Comparison

SRQ3 *Which multi-objective algorithms should be compared with when evaluating an algorithm?*

Getting an overview of the state of the art algorithms, and how their experiments are carried out is important. There are too many multi-objective algorithms in the literature to compare with them all, and therefore it is important to see which algorithms and how many algorithms the state of the art compares with. Interesting aspects related to the experimental comparison are:

- **Performance:** how well the algorithm performs compared to other algorithms.
- **Convergence and diversity:** for multi-objective algorithms a balance between convergence and diversity is important.
- **Computational complexity:** when comparing the performance of algorithms it is useful to know their computational complexity to ensure a fair comparison is performed.
- **Which algorithms are used to compare results with:** knowing which algorithms are considered competitive by the community is useful for the experiment performed later in this thesis.

Performance

Because of the stochastic nature of multi-objective optimization results are usually gathered over a set number of runs. Ten or more runs are common, with one paper using as many as 100. Two common stopping criteria are running the algorithm for a set number of fitness evaluations or iterations, where the number of function evaluations is the most common. The number of function evaluations varies, but most seem to use a number between 25,000 and 300,000.

It is hard to directly compare the performance of the multi-objective algorithms in the literature review since the number of function evaluations, test functions and comparison algorithms differ significantly. Since the typical way to present test results is by listing the mean performance metric of the end result, comparing results after 25,000 evaluations with results after 300,000 evaluations will most likely not provide any useful information. In general, all the multi-objective algorithms conclude that they are better than or competitive with the algorithms in their experiments. However, it is interesting to compare how the obtained Pareto fronts look when plotted with the Pareto optimal front as this clearly shows the convergence and diversity of the obtained front. The most common test functions plotted are the ZDT test functions. SMPSO (ZDT4), NS-MFO (ZDT1-3) and MODFA (ZDT1-3) show good results both in convergence and diversity of the generated fronts, while MOALO (ZDT1-2) lacks diversity and NSIWO (ZDT1-4) lacks convergence. It should be noted that NSIWO is tested only with 25,000 function evaluations and might improve by letting it run longer like most of the other algorithms do. MOWOA and MOGWO also plot solutions for some of the test functions from CEC2009, and they both have problems with low diversity, but that might be an indication that CEC2009 is harder to solve than the ZDT functions.

Algorithms Used for Comparison

Selection of comparison algorithms and the number of algorithms selected differs significantly between the multi-objective algorithms. Some only compare with the algorithm they are trying to improve, while NS-MFO compares with as many as 26 other algorithms. Comparing with 2-4 other algorithms is most common. The most common algorithms to compare with are MOEA/D, MOPSO, NSGA-II, PAES, and SPEA. A common factor for these algorithms is that they have been used for a while and have proven their worth.

Computational Complexity

Common for all the algorithms is that the multi-objective technique is the largest factor in the computational complexity and therefore decide the complexity. Since

most of the algorithms are based on either MOPSO-like archive or NSGA-II's non-dominated sorting their complexity is $O(mn^2)$. Outliers are AMOSA with a complexity of $O(n(m + \log n)$ and decomposition based algorithms with $O(mnt)$. m is the number of objectives, n is the archive/population size, and t is the number of neighboring solutions considered in decomposition. Note that these are the complexities of running a single generation of the algorithms.

Decomposition-based algorithms can handle more objectives as the computational complexity is lower while NSGA-II and MOPSO based algorithms are usually only used on problems with two or three objectives.

Conclusion

The experimental setup used by articles differs greatly, but a common factor is that the comparison is performed using the mean and standard deviation of several runs on test functions. The algorithms used for comparison of results are often well-known algorithms from the 90s and 00s. Most of the algorithms included in this literature review have a computational complexity of $O(mn^2)$, or comparable complexities like decomposition with $O(mnt)$.

3.3.4. Test Functions

SRQ4 *Which test functions are used to test multi-objective algorithms?*

When selecting which test functions to use it is essential to find functions that can test for both convergence and diversity. It is also favorable if the test functions have been used to test other state of the art algorithms, as it makes it easier to compare the performance of the algorithm with the state of the art.

Table 3.1 shows the test functions used for testing in the multi-objective articles that were part of the literature review. The three test suites that the majority of articles used are the DTLZ test functions [Deb et al., 2005], the ZDT test functions [Zitzler et al., 2000] and the UF test functions from CEC 2009 [Zhang et al., 2008]. Additionally, WFG [Huband et al., 2005] are common test functions for solving more than two objectives as they are designed to scale dynamically. All the test functions mentioned have scalable decision variables, but in the literature review, 30 decision variables are commonly used. They are also designed to cover a variety of different search spaces and fronts.

Table 3.1.: The test functions used by the articles covering multi-objective algorithms

Algorithm	Test Functions
AMOSA	DTLZ 1-6
DPP2	DTLZ 1-7, UF1-10, WFG1-9, ZDT 1-4,6
MOALO	ZDT 1-3
MODFA	ZDT 1-3
MOEA/DD	DTLZ 1-4, WFG1-9
MOGWO	UF1-10
MOWOA	UF1-4,6-7
NSIWO	ZDT1-4,6
NS-MFO	LZ, SCH, UF1-10, ZDT1-3
SMPSO	DTLZ1-7, ZDT1-4,6

3.3.5. Performance Metrics

SRQ5 *Which performance metrics are used to measure multi-objective algorithms?*

The performance metrics used by the multi-objective articles in the literature review are displayed in Table 3.2. The most commonly used performance metrics are IGD, GD and Δ (Spread), but a large variety of metrics are used. MODFA is the only algorithm that was only measured with one performance metric. While this is not ideal, the metric used is IGD which measures both convergence and diversity. The majority of the articles has a performance metric that measures both diversity and convergence at the same time.

Table 3.2.: The performance metrics used by the articles covering multi-objective algorithms

Algorithm	Performance metrics
AMOSA	MinimalSpacing, Purity, Spacing, γ (Convergence)
DPP2	GD, HV, IGD, IGD+
MOALO	GD, IGD, Spacing, Δ (Spread)
MODFA	IGD
MOEA/DD	HV, IGD
MOGWO	IGD, Maximum Spread, Spacing
MOWOA	GD, Δ (Spread)
NSIWO	Δ (Spread), γ (Convergence)
NS-MFO	GD, IGD, Spacing, Δ (Spread)
SMPSO	Additive Unary Epsilon indicator, HV, Δ (Spread),

4. Framework Architecture

This chapter will cover the framework that was built for implementing and testing algorithms. Following are the reasons why building a framework was necessary.

- **Gathering statistics:** A framework gave a reliable way of running algorithms and gathering statistics over multiple runs.
- **Experimentation:** Having a framework made it easier to experiment with different techniques, as algorithms can quickly be extended with techniques used by other algorithms implemented in the framework.
- **Reuse of implemented techniques:** Having all algorithms implemented in the same framework allows for reuse of techniques that have already been implemented. This reduces the chance of mistakes in implementation, as the same techniques will not have to be implemented multiple times and more time can be dedicated to checking the correctness of the implementation.

In the following section, the architectural overview is explained. Following that the different modules of the framework are explained in greater detail.

4.1. Architectural Overview

The framework was created using the Rust programming language[1]. The reason the framework is written in Rust is mostly because of its fast runtime. Rust is a modern systems programming language similar to C and C++. However, unlike C and C++, Rust has many zero-cost abstractions that make it easier to write safe code without affecting the runtime. The framework is accessed using a Command Line Interface (CLI) where the test function(s) and the algorithm are specified. Additionally, command line parameters such as population size and the number of iterations can be provided. Each algorithm can also have specific command line parameters like mutation rate specified.

[1]The Rust programming language: `https://www.rust-lang.org`

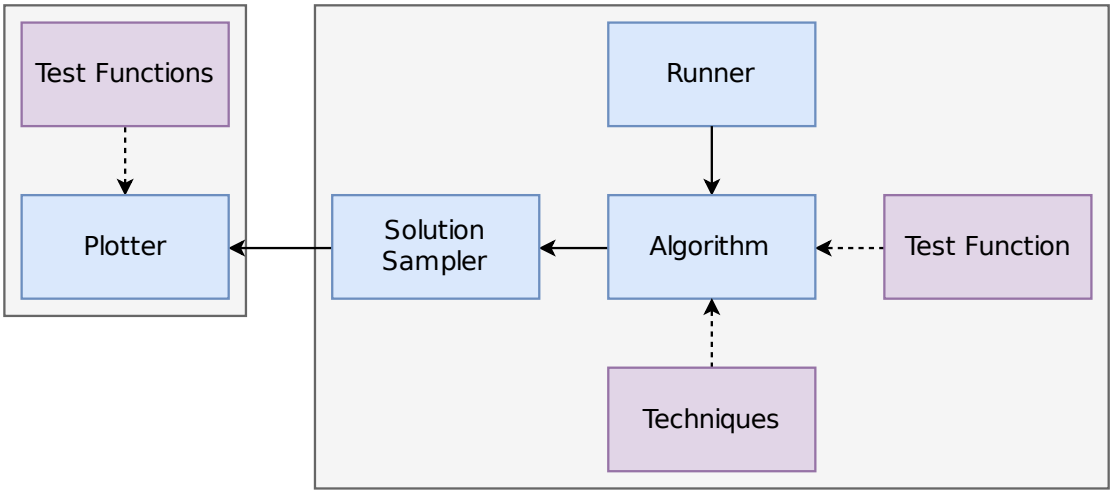


Figure 4.1.: Architectural overview of the framework. The dashed arrows connecting the modules show how the algorithm and plotter use other modules. The solid arrows show how the different modules communicate with each other. The gray boxes separate the plotter modules and optimization modules.

A diagram displaying the architectural overview of the framework can be seen in Figure 4.1. A runner is in charge of loading in all the test functions and test suites, as well as running algorithms for a set number of rounds. The algorithm can use generic techniques which are usually stateless functions like mutation and crossover or they can store solutions like an external archive. While the algorithm is running the solution sampler is called and stores the solutions for later use. To evaluate a solution the algorithm calls a test function which is selected before starting the run. When the algorithm has completed the solutions are collected from the solution sampler and statistics about the run(s) are shown. If the plotter is running the solutions stored in the solution sampler are plotted. If the test function plotted is single-objective the plotter first plots the whole search space, using test functions implemented in the plotter, and then the sampled solutions on top of the function plot. Some of the modules in the framework are described in greater detail in the following sections.

4.2. Test Functions

A range of both single and multi-objective test functions are implemented in the framework. An algorithm can use one or more test functions in the same execution by specifying the test functions using the CLI. It is also possible to use an entire

test suite. The single-objective test functions have support for shifting, rotating and scaling the input based on provided data sets. Additionally, the framework automatically keeps track of the number of fitness evaluations (calls to the test functions), which means that neither the algorithms nor test functions have to implement any functionality for this. The majority of the test functions have a scalable number of decision variables.

4.3. Solution Sampling

The solution sampler module was created to make it easier to analyze the behavior of the algorithm with different settings. The sampler has several purposes where the main ones are to show statistics about the quality of the solutions, like mean and standard deviation, and to sample solutions to plot. It has four sampling modes: sample final generation only, sampling over time of the whole population, sampling over time of the best solution only, and sampling all fitness calculations. Sampling of the final generation is the default behavior and shows the final result of the algorithm. Sampling over time shows how the algorithm converges, while sampling all fitness evaluations for use in plotting shows how the algorithm searches. The statistics printed behave a little different between single and multi-objective. For single-objective problems min, max, mean, and standard deviation are shown when more than one solution is sampled, while for multi-objective problems the implemented performance metrics GD, HV, and IGD are shown. If the number of runs is more than one, additional statistics of the sampled solutions are shown using mean and standard deviation. This is useful because of the many random elements often involved in metaheuristic algorithms.

4.4. Plotting

A plotter has been created to plot solutions. The plotter was created as a stand-alone program. It was created using the Python programming language[2]. The plotter listens for new output from the main program. When it receives new output it will automatically display a plot of it in a new window.

The plotter can plot single objective solutions with two decision variables. The plots are three dimensional, with the horizontal axes showing positions, and the vertical axis showing fitness value. The area that is plotted is adjusted automatically based on the values of the solutions that are plotted. It is possible to always plot the bounds using a CLI parameter. An example of a plot of the test function Rosenbrock generated using the plotter is shown in Figure 2.7. It is also possible

[2]The Python programming language: `https://www.python.org/`

to plot multi-objective Pareto fronts. The Pareto front is plotted together with the Pareto optimal front. Several examples of multi-objective plots are shown in Figure 7.1. The Pareto optimal fronts for all test functions are pre-generated and stored in the framework. These pre-generated fronts are the same files that are used when calculating the performance metrics that rely on the Pareto optimal front. The approximate Pareto set can also be plotted together with the Pareto optimal set. This plot is three dimensional and can give an overview of how well an algorithm managed to spread within the search space. Like with the Pareto optimal fronts, the input is pre-generated and stored in the framework. The plotter can be used to plot several Pareto fronts at the same time which makes it easier to compare the quality of solutions generated by different algorithms. While all plots of Pareto fronts and single-objective test functions in this thesis are generated by this plotter, some tweaks have been made to improve the visual quality of the plots.

5. Selection of Single-Objective Algorithm

In order to answer **RQ1** the single-objective algorithms from the literature review are evaluated and an empirical experiment is performed. The selected algorithm is presented in detail at the end of the chapter.

5.1. Evaluating Algorithms

The most promising algorithms from the literature review will be implemented and compared with each other more thoroughly in an empirical experiment. The choice of implementing the algorithms was for a few reasons:

- **To get an even better understanding of the algorithms:** When implementing an algorithm it has to be thoroughly analyzed to make sure every detail is correct. This gives a great understanding of every part of the algorithm.

- **To be able to compare the algorithms:** For testing it is ideal to have all the relevant algorithms implemented in the same framework to easily compare results on test functions and ensure that they are tested equivalently.

- **To verify the results in the articles:** Unfortunately, it has been shown that it can be hard to reproduce the results from artificial intelligence articles, in fact only 6% of articles presented on two top AI conferences shared the code [Hutson, 2018]. Implementing the algorithms is a decent way to confirm the quality of the results, but inconsistent results do not necessarily mean the algorithm is flawed as the reproduction of the algorithm can be flawed or the article is not explained thoroughly enough.

Six single-objective algorithms were gathered from the literature review. Because of limited time, it was not deemed feasible to implement six separate algorithms. One of the main points of the thesis is innovation, and therefore some algorithms were cut early and will not be a part of the experiment.

MS will not be implemented. MS and MFO are based on many of the same premises. Since MFO has already been extended to handle multiple objectives, using non-dominated sorting, it was decided that extending MS was not that innovative as it would likely consist of more or less the same work.

Additionally, SOS will not be implemented. There already exists a multi-objective version of SOS called Multiple Objective Symbiotic Organisms Search [Tran et al., 2016] that is based on non-dominated sorting. Since other promising single-objective algorithms, without a multi-objective counterpart, were gathered in the literature review it was decided that SOS would not be a part of the further process.

5.2. Implementation Summary

The following four algorithms were implemented and are part of the empirical experiment. Details about the implementation process for each algorithm is presented in this section.

Lion Optimization Algorithm

Lion Optimization Algorithm was by far the most complex single-objective algorithm included in the literature review. For each iteration of the algorithm ten different steps are performed and if one misbehaves it can ruin the rest of the search. The resulting implementation is likely to contain errors since no existing implementation of LOA was found and the article is somewhat vague on some implementation details.

Dandelion Algorithm

DA was simple to implement. The pseudocode of DA in the article gave a good overview of the implementation procedure. Because the algorithm always uses the solution with the best fitness value as the mother dandelion for the next generation the selection process was especially easy to implement.

Animal Migration Optimization

AMO was the most straightforward algorithm to implement. The article's explanations regarding the implementation process were detailed, and the algorithm itself has few tunable parameters, which made it easy to detect if something was implemented incorrectly.

Earthworm Optimization Algorithm

Implementing EWA was less complicated than LOA, but some steps described in the article were confusing and contained conflicting information. Luckily a

MATLAB implementation[1] was found from one of the authors of the article, which cleared up some uncertainty, but running the MATLAB code did not yield the same results as presented in the article. Since all the results in the EWA article are normalized, it is not possible to directly compare the results. However, by comparing the results of the MATLAB EWA implementation with the results of other algorithms, it became clear that EWA does not perform well.

5.3. Single-Objective Experiment

As a part of the evaluation of single-objective algorithms, an empirical experiment was conducted. The goal of the experiment is to see how well the implemented algorithms perform on a variety of test functions, which will make it easier to select an algorithm to extend to handle multi-objective problems. Additionally, the behavior of the algorithms and their strengths and weaknesses are discussed.

5.3.1. Experimental Setup

Since most algorithms can technically run forever, a stopping criterion is needed. For this experiment, fitness evaluations were used as the stopping criterion for all the algorithms. Alternatives would be setting the maximum number of iterations or use CPU time, but every algorithm differs in the amount of work done in each iteration and CPU time would need the same hardware and programming language to replicate which makes the number of fitness evaluations the most suitable stopping criterion. The experimental setup is based on the setup used in CEC2014. The algorithms ran until they hit a total of 300,000 fitness evaluations on all test functions except for High Conditioned Elliptic, Bent Cigar, and Discus. The number of fitness evaluations was set to 150,000 for these test functions, as these are quite simple unimodal problems, and multiple algorithms would find perfect solutions every run with 300,000 fitness evaluations. The maximum number of iterations was set to 10,000 to meet the fitness evaluation stopping criterion. Because all the algorithms implemented contain stochastic operations each algorithm was run 51 times for each test functions as done in CEC2014.

Each algorithm ran on the 14 test functions described in CEC2014 [Liang et al., 2013]. CEC2014 offers a variety of test functions to test different types of optimization problems, as explained in Section 2.7.3. Most test functions have a global optimum when every decision variable is 0, which can be exploited either intentionally or accidentally by algorithms. Using data files published for use with CEC2014 the input to the test functions are shifted and rotated to avoid

[1] Earth Worm Optimization MATLAB implementation: `https://se.mathworks.com/matlabcentral/fileexchange/53479-earthworm-optimization-algorithm--ewa-`

exploitation of common test functions. Lower and upper bound was set to -100 and 100 respectively. Additionally, three test functions were added: Axis Parallel Hyper-Ellipsoid, Moved-Axis Parallel Hyper-Ellipsoid, and Modified Easom. The extra test functions can showcase some edge cases and make it easier to identify misbehaving algorithms. The Hyper-Ellipsoid test functions used -500 and 500 as bounds instead since Moved-Axis Parallel Hyper-Ellipsoid has an optimum relative to the number of decision variables. Modified Easom is a test function that has a single global optimum that is very hard to find, so bounds were set to -10 and 10 to accommodate this. All the test functions used 30 decision variables and are described in Appendix D.

The parameters used for the algorithms are the same that were used in their original article and can be seen in Table 5.1. DA does not operate using a population like the other algorithms.

Table 5.1.: Tunable parameter settings of the algorithms

AMO		DA		EWA		LOA	
Population	50	Growth rate	1.1	Population	50	Population	50
		Wither rate	0.9	Similarity factor	0.98	Percent of nomad lions	0.2
		Normal seeds	100	Proportional factor	1.0	Roaming percent	0.2
		Self-learning seeds	1	Cooling factor	0.9	Mutate percent	0.2
						Sex rate	0.8
						Mating probability	0.3
						Immigrate rate	0.4
						Number of prides	4

5.3.2. Results and Evaluation

The mean and standard deviation (SD) of the fitness values obtained over 51 runs is shown in Table 5.2. LOA and EWA perform poorly on all test functions and do not provide the best solution on any of the test functions. The inadequate performance of LOA is likely because the implementation is incorrect. AMO has the overall best results with the best score on 10 out of 17 test functions while DA finds the best solution on 8 out of 17 test functions and performs reasonably well on all test functions except for Modified Easom and Rosenbrock. Both AMO and DA always obtain a perfect score on the Moved-Axis Parallel Hyper-Ellipsoid test function while EWA and LOA are not even close to achieving an optimal answer. AMO performs better than the other algorithms on the unimodal test functions High Conditioned Elliptic, Bent Cigar, Discus, and Axis-Parallel Hyper-Ellipsoid. None of the algorithms included in this experiment are able to find solutions very close to the global optimum on Rosenbrock, Ackley, HappyCat, and HGBat. These are all multi-modal test functions and many of them have a large number of local optima which make them hard to solve. AMO is also clearly best on Griewank and

Table 5.2.: Experiment results on the single-objective algorithms

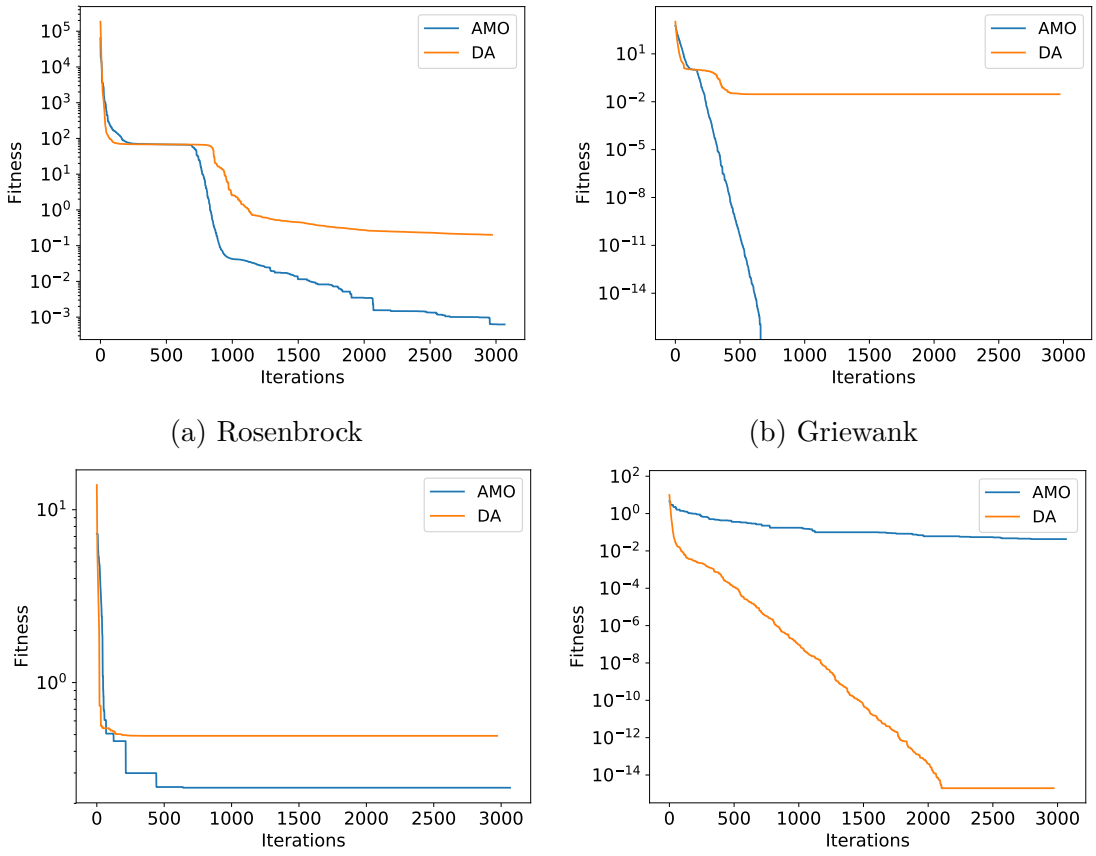
Test function	Measure	AMO	DA	EWA	LOA
High Conditioned	Mean	**0.00E+00**	9.13E-11	3.46E+08	1.08E+09
Elliptic[1] [4]	SD	0.00E+00	2.55E-10	1.36E+08	3.24E+08
Bent Cigar[1] [4]	Mean	**0.00E+00**	9.54E-10	4.78E+10	1.08E+11
	SD	0.00E+00	1.22E-09	1.49E+10	2.06E+10
Discus[1] [4]	Mean	**0.00E+00**	1.55E-11	1.29E+05	6.96E+06
	SD	0.00E+00	3.94E-11	6.28E+04	1.25E+07
Rosenbrock[1]	Mean	**1.57E-01**	3.41E+00	5.33E+03	5.42E+04
	SD	7.82E-01	1.33E+01	2.51E+03	1.95E+04
Ackley[1]	Mean	2.00E+01	**3.92E+00**	2.11E+01	2.14E+01
	SD	6.86E-03	8.02E+00	1.23E-01	6.12E-02
Weierstrass[1]	Mean	1.49E-02	**2.23E-14**	3.47E+01	4.23E+01
	SD	6.41E-02	1.52E-14	1.89E+00	1.91E+00
Griewank[1]	Mean	**0.00E+00**	4.36E-02	2.54E+02	4.70E+02
	SD	0.00E+00	5.98E-02	1.16E+02	9.68E+01
Rastrigin[1]	Mean	3.51E-01	**2.23E-13**	3.79E+02	5.80E+02
	SD	5.20E-01	1.32E-13	3.08E+01	6.92E+01
Modified Schwefel[1]	Mean	1.53E+01	**6.99E-01**	7.46E+03	9.10E+03
	SD	6.90E+00	8.86E-01	4.83E+02	5.33E+02
Katsuura[1]	Mean	4.01E-02	**4.20E-15**	2.65E+00	7.16E+00
	SD	5.97E-03	2.41E-15	8.03E-01	9.01E-01
HappyCat[1]	Mean	**2.53E-01**	4.81E-01	4.65E+00	9.11E+00
	SD	3.38E-02	1.23E-01	7.11E-01	1.10E+00
HGBat[1]	Mean	**2.04E-01**	3.52E-01	1.65E+02	2.91E+02
	SD	2.84E-02	1.68E-01	6.60E+01	4.56E+01
Griewank	Mean	3.11E+00	**1.01E+00**	2.00E+04	7.49E+06
+ Rosenbrock[1]	SD	2.69E-01	3.33E-01	1.99E+04	5.35E+06
Expanded	Mean	4.50E+00	**9.37E-01**	1.30E+01	1.45E+01
Schaffer[1]	SD	3.48E-01	4.26E-01	4.10E-01	1.71E-01
Axis Parallel	Mean	**1.06E-87**	1.61E-33	1.12E-04	2.09E+07
Hyper-Ellipsoid[2]	SD	1.43E-87	3.57E-33	4.42E-05	3.31E+06
Moved-Axis Parallel	Mean	**0.00E+00**	**0.00E+00**	7.25E+07	3.29E+08
Hyper-Ellipsoid[2]	SD	0.00E+00	0.00E+00	1.47E+07	5.10E+07
Modified Easom[3]	Mean	**-1.00E+00**	0.00E+00	-1.75E-87	-4.01E-139
	SD	0.00E+00	0.00E+00	1.25E-86	2.86E-138

[1] Shifted and rotated input using CEC2014 data files
[2] Lower and upper bound set to -500 and 500
[3] Lower and upper bound set to -10 and 10
[4] 150,000 fitness evaluations

Modified Easom where it finds the global optimum every time. However, DA is superior to AMO on the test functions Weierstrass, Rastrigin, and Katsuura with results many orders of magnitude closer to to the optimal value.

5.3.3. Discussion

This section contains a discussion about some of the more interesting behaviors of the algorithms on some test functions.



(a) Rosenbrock

(b) Griewank

(c) HappyCat

(d) Katsuura

Figure 5.1.: Fitness convergence of AMO and DA over iterations.

Fitness Convergence of AMO and DA

DA has superior performance on some test functions, and by plotting the best fitness value for each iteration, it is easier to see how AMO and DA performs. Figure 5.1 shows the fitness convergence on four functions. On Rosenbrock both algorithms hit a local optimum for a while before they find a more promising region

to explore. On Griewank DA quickly gets stuck in a local optimum, while AMO gradually improves the fitness until it finds the optimal fitness. DA obtains a much better fitness value than AMO on Katsuura, but ends up in a local optimum at the end of the run. AMO converges really slowly, leading to a poor result. On HappyCat DA converges really fast, but gets stuck in a local optimum that it never escapes. AMO has a similar start to DA, but it escapes several local optima and achieves a better fitness value. Overall, DA is more likely to end up in local optima that it can not escape than AMO.

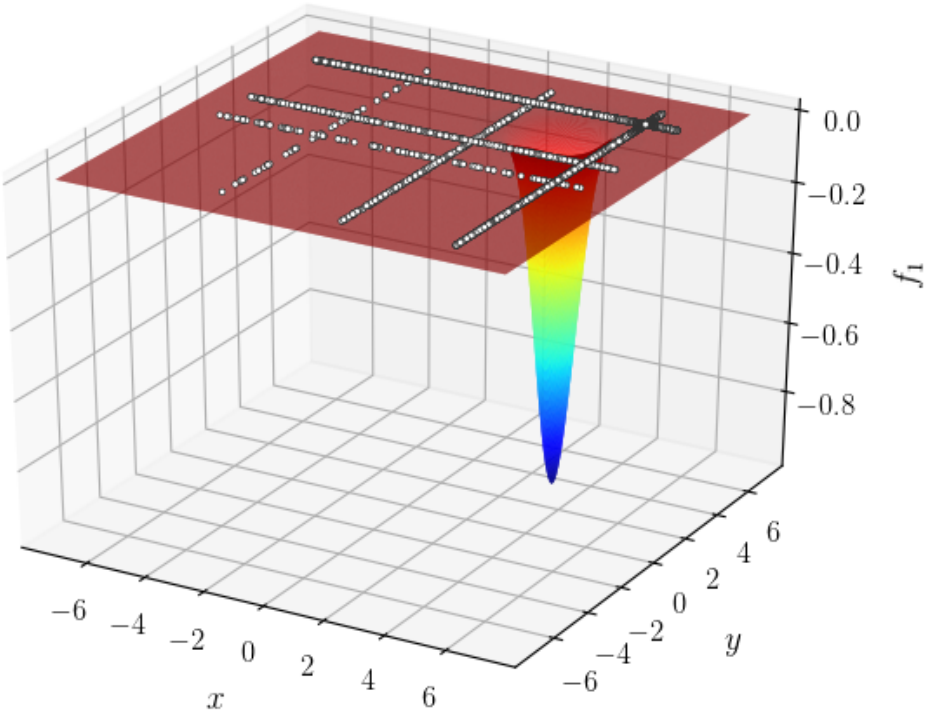


Figure 5.2.: Plot of Dandelion Algorithm after 300,000 fitness evaluations on Modified Easom using two decision variables

Limitations of Searching Using One Decision Variable with DA

The shape of several of the test functions is approximately like a uniform sphere, with a global optimum in the center for every decision variable. Since DA only searches using one decision variable at a time, it can usually find decent solutions for these types of problems, as moving towards the smallest value in the decision variable frequently moves the solution towards the global optimum. Moving like this, however, does not work well with non-uniform test functions like Modified Easom. The search space of Modified Easom is more or less a flat surface with a

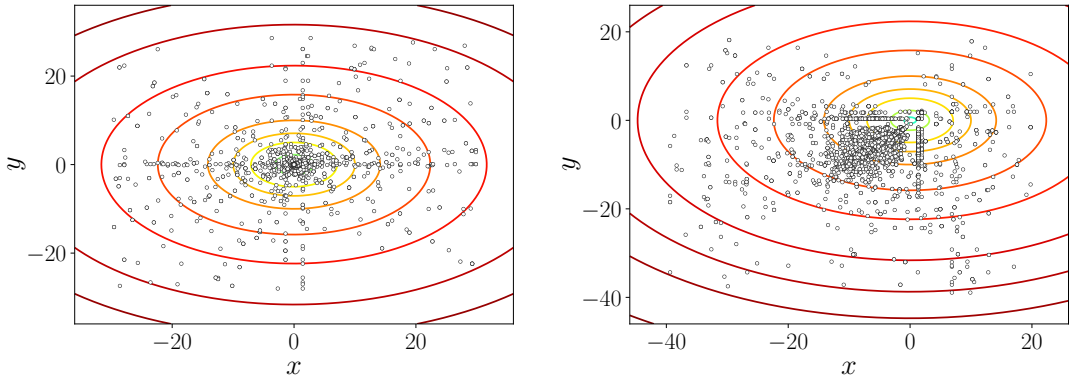
hole around π containing the global optimum. As seen in Figure 5.2 DA is stuck and unable to find the optimum even when only using two decision variables. The algorithm has to have a sound global search to find the hole in Modified Easom, and can not rely on local search techniques. As can be seen from the results AMO is the only algorithm that can reliably find the global optimum of -1 while the other algorithms are unable to find the global optimum consistently.

EWA's Convergence Against the Middle of Search Space

Plotting fitness evaluations of EWA showed that it frequently searched around the middle of the search space. For example with lower and upper bounds set to -30 and 30 respectively EWA searches mostly around 0. Coincidentally many of the unshifted test functions' global optimum input is 0. This behavior is especially noticeable when running EWA with γ, the cooling factor, set to 0.99 instead of 0.9. By shifting the upper and lower bounds to something where the middle of the search space does not equal 0, for example -28 and 32, shows that EWA performs worse, for no good reason other than accidentally finding decent solutions earlier. To clearly show the effect shifting of bounds has contour plots of EWA with and without shifted bounds on Axis Parallel Hyper-Ellipsoid is shown in Figure 5.3. The results from running EWA with different γ values and bounds changed to -28, 32 is shown in Table 5.3. EWA with $\gamma = 0.99$ performs excellently on test functions where the global optimum and the search space median are equal, but once the search space median changes a relatively small amount EWA performs poorly. Moved Axis Parallel Hyper-Ellipsoid, a test function equal to Axis Parallel Ellipsoid except with a decision variable optimum dependent on the decision variable number, shows this problem on all EWA versions as it as a global optimum far away from 0. Contours plots of EWA and AMO on Moved-Axis Parallel Hyper-Ellipsoid is shown in Figure 5.4 and it is clear that EWA searches mostly in the middle of the search space while AMO is mostly searching near the global optimum. Other algorithms can with ease find the global optimum. The MATLAB implementation of EWA mentioned in Section 5.2 has the same problems, and the algorithm should, therefore, be considered flawed.

Table 5.3.: Results on running EWA with different settings

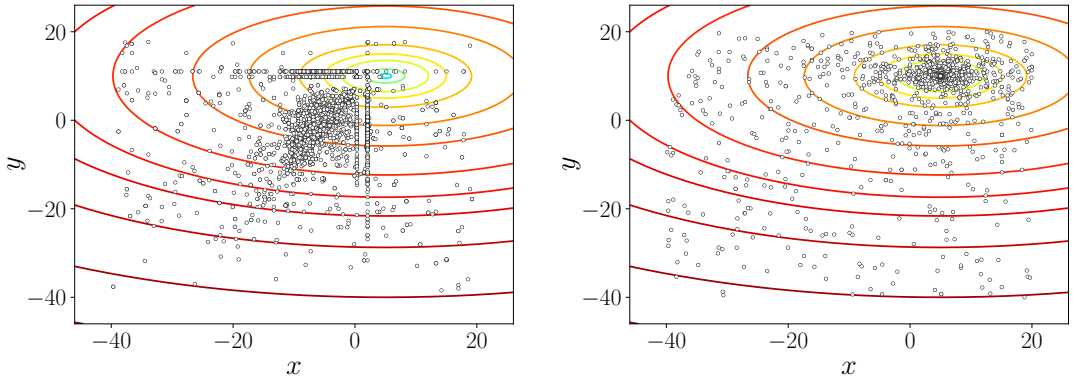
Test function	Measure	EWA	EWA 0.99	EWA Shifted	EWA Shifted 0.99
Axis Parallel	Mean	1.16E-04	**0.00E+00**	1.38E+03	1.02E+03
Hyper-Ellipsoid	StdDev	4.93E-05	0.00E+00	1.37E+02	2.68E+02
Moved-Axis Parallel	Mean	**6.94E+07**	7.85E+07	9.83E+07	9.52E+07
Hyper-Ellipsoid	StdDev	1.07E+07	1.47E+07	1.80E+06	2.22E+06
Modified Easom	Mean	-8.54E-101	-3.31E-92	-2.13E-24	**-7.36E-02**
	StdDev	3.80E-100	1.66E-91	1.06E-23	1.16E-01



(a) EWA with the global optimum in the middle of the search space

(b) EWA with shifted bounds

Figure 5.3.: Plots of 3,000 fitness evaluations on Axis Parallel Hyper-Ellipsoid using two decision variables as axes.



(a) EWA

(b) AMO

Figure 5.4.: Plots of 3,000 fitness evaluations on Moved-Axis Parallel Hyper-Ellipsoid using two decision variables as axes and shifted bounds.

5.3.4. Conclusion

Two of the implemented algorithms, EWA and LOA, do not perform well enough to be considered for selection. The implementation of EWA does not seem to search correctly and is unable to find any decent results. LOA cannot be used as no proper implementation of it is available, and based on the results the version implemented does not perform well at all. AMO and DA both obtain decent results on several test functions, but as discussed DA can quickly get stuck in local optima depending on the problem. Converging to local optima is likely to be a more

significant problem on multi-objective problems which makes DA less ideal than AMO. Overall the conclusion is that AMO performs and behaves best out of the algorithms in the experiment and is most suitable for multi-objective extension and is therefore chosen.

5.4. Presentation of Animal Migration Algorithm

Animal migration optimization [Li et al., 2014] has two phases: the animal migration phase and the population updating phase. In the animal migration phase solutions move based on the position of their neighbors. In the population updating phase solutions are changed with a probability based on how good their fitness is. When creating the first population, all solutions are initialized at random positions within the search space.

The pseudocode for the algorithm is presented in Algorithm 1. P is the main population and Q is the new population created in each phase. $P_{i,j}$ refers to decision variable j in solution i in population P. $f(P_i)$ is the fitness value of solution i in P. n is the population size, δ is a random number between 0 and 1 from the Gaussian distribution, and $rand$ is a random number between 0 and 1 from the uniform distribution. S are the probabilities used when deciding if a decision variable should be changed in the population updating phase.

5.4.1. Animal Migration

The animal migration phase of the algorithm can be seen from line 3 to line 16 in Algorithm 1. A ring topology is used for the solutions and a neighborhood refers to any five solutions that are next to each other in the ring topology. The algorithm loops over every solution, and for each decision variable, it moves the solution towards a random solution within its neighborhood. A new solution from the neighborhood is picked at random for each decision variable. The distance moved is calculated by taking the difference between the neighbor and the current position $P_{neighborhood,j} - P_{i,j}$, and multiplying it by δ as seen on line 6. After the solutions have been moved, it compares the fitness of the new and old position and keeps the one with the best fitness. This can be seen from line 12 to line 16.

5.4.2. Population Updating

The population updating phase can be seen from line 17 to 35 in Algorithm 1. A probability S_i is calculated for each solution in P and is used on line 20 when looping over every decision variable, deciding if it should be changed or not. The

Algorithm 1 Animal Migration Optimization

Initialize P with n random solutions
while stopping criterion is not met **do**
    for $i = 1$ to n **do** ▷ Animal migration
        Generate δ from the Gaussian distribution
        for $j = 1$ to number of decision variables **do**
            $Q_{i,j} = P_{i,j} + \delta \times (P_{neighborhood,j} - P_{i,j})$
        end for
    end for
    if solution is out of bounds **then**
        Place solution at a random position within the search space.
    end if
    for $i = 1$ to n **do**
        if $f(Q_i) < f(P_i)$ **then**
            $P_i = Q_i$
        end if
    end for
    Calculate probabilities S ▷ Population updating
    for $i = 1$ to n **do**
        for $j = 1$ to number of decision variables **do**
            if $rand > S_i$ **then**
                Generate two random population indices where $r1 \neq r2 \neq i$
                $Q_{i,j} = P_{r1,j} + rand \times (P_{best,j} - P_{i,j}) + rand \times (P_{r2,j} - P_{i,j})$
            else
                $Q_{i,j} = P_{i,j}$
            end if
        end for
    end for
    if solution is out of bounds **then**
        Place solution at a random position within the search space.
    end if
    for $i = 1$ to n **do**
        if $f(Q_i) < f(P_i)$ **then**
            $P_i = Q_i$
        end if
    end for
end while

probability is calculated using Equation (5.1).

$$S_i = \frac{n - D_i}{n} \tag{5.1}$$

D_i is the number of solutions in the population with better fitness than P_i, and n is the population size. This leads to solutions with good fitness having few or none of their decision variables changed, while bad solutions are almost replaced entirely. If decision variable $P_{i,j}$ is selected for replacement the new decision variable is calculated based on the position of the best-found solution P_{best_j} and the position of two other solutions selected randomly from the population $P_{r1,j}$ and $P_{r2,j}$ as seen on line 22. The first randomly selected solution's position $P_{r1,j}$ is used as a base for the new position. The base position is then changed by adding the difference between the best position and the current position $P_{best_j} - P_{i,j}$ multiplied by a random number *rand* sampled from the uniform distribution. Additionally, the difference between the second randomly selected solution's position and the current position $P_{r2_j} - P_{i,j}$ is also multiplied by another random number sampled from the uniform distribution and added to the new position. Like in the animal migration phase the new solutions are compared with the solution they are based on and the solution with the best fitness is kept.

5.4.3. Out of Bounds Check

The original article does not cover how AMO handles positions that go out of bounds. A MATLAB implementation of the algorithm created by one of the article's authors was found which covers this[2]. When any decision variable is out of bounds, the whole solution is placed at a random position within the search space. This check is shown on line 9-11 and 28-30.

[2]AMO implemented in MATLAB: `https://se.mathworks.com/matlabcentral/fileexchange/65846-animal-migration-optimizer`

6. Multi-Objective Extension

This chapter aims to answer **RQ2** by extending AMO to handle multi-objective problems using different techniques. The extended algorithms will then be evaluated against each other to find the best fit, and the best one is selected as the final version of the algorithm. At the end of the chapter, the selected version is presented.

6.1. Multi-Objective Techniques on the Algorithm

Some changes were required for AMO to handle multi-objective problems. After each phase AMO decides if a solution should be replaced with the new solution based on which solution has the best fitness. With one objective this is quite simple, as the comparison uses the single objective fitness value and selects the lowest value. However, with several fitness values, there is no simple way to decide whether a solution is better than another. Pareto dominance can be used if one solution is dominated by the other, but if they are both non-dominated with respect to each other then the selection has to be based on something else. Similarly, during the population updating phase, the probability that a decision variable of a solution is changed is based on how good the fitness value of the solution is compared with the rest of the population, which also has to be changed. The global best-known solution so far was used to calculate the new position, this also has to be selected in a different way.

In the literature review, the techniques that are used when extending single-objective algorithms to handle multi-objective problems were studied. The extension techniques that were found to be the most common are archiving and non-dominated sorting. These techniques can quite easily be added on top of an algorithm without changing the core functionality of the algorithm in most cases. Two multi-objective versions were therefore created based on these techniques. Additionally, a hybrid version of these algorithms was created which combines archive and non-dominated sorting. The following subsections will explain the process of extending AMO using these techniques.

6.1.1. Non-dominated Sorting Version

Efficient Non-dominated Sort [Zhang et al., 2015] was used to implement the non-dominated sorting version of AMO. This technique is interchangeable with the non-dominated sorting technique used by NSGA-II, except that is has a lower best case complexity. The version of AMO based on non-dominated sorting required a few changes to AMO. The selection at the end of each phase combines the old and new population, sorts the merged population using non-dominated sorting, and keeps the first half. During the population-updating phase, the best solution is chosen using binary tournament selection with crowding distance on all the non-dominated solutions in the population. The chance for replacement was unchanged as the population could be sorted using non-dominated sorting.

6.1.2. Archive Version

Two different archives were tested on AMO. The first was the MOPSO archive explained in Section 2.6.4. This archive was the most used archive type by the algorithms in the literature review. It worked well when used for extending single-objective algorithms. The second archive is from SMPSO and is explained in explained in Section 3.2.3. This archive worked well with SMPSO, and was simple to implement as it used crowding distance for pruning, something that had already been implemented for non-dominated sorting. The SMPSO archive performed slightly better and was used for the final version of archive AMO. To handle the comparison between the new and old solution that happens at the end of both phases a technique from MOPSO was used. A domination check is performed on the solutions, and if one solution's fitness values dominates the other's fitness values, it gets to stay in the population. If none of them dominate each other, one is selected randomly. Selection of the best solution was performed similar to leader selection in SMPSO using binary tournament selection with crowding distance on the archive. The probability of replacement is calculated based on the percentage of solutions the solution is dominated by. The archive is updated at the end of each phase.

6.1.3. Hybrid Version

In addition to the two versions described a version using both archive and non-dominated sorting was created. This version used the external archive to store solutions and to select the best solution during the population updating phase. Non-dominated sorting is used to calculate the probability for replacement like in the non-dominated sorting version. Additionally, the population is pruned using non-dominated sorting by combining the old and new population, sorting it and keeping the best half.

6.1.4. Additional Improvements

When first extending to multi-objective, the out of bounds handling from the single-objective algorithm was used. It placed solutions that went out of bounds for at least one decision variable at a completely random position in the search space. This did not perform well when extended to multi-objective, and therefore other methods were tried. Placing only the decision variable that went out of bounds at a random position was attempted and gave slightly better results than the original approach. Placing the decision variable at the bound where it went out of bounds was also attempted, and the algorithms performed a lot better. This ended up being used for every version.

To avoid prematurely converging a mutation operator was added. Two mutation operators were tested. One from MOPSO explained in Section 2.6.4, the other from SMPSO explained in Section 3.2.3. The MOPSO mutation performed better and was used for every version. The mutation happens between the animal migration and population updating phase.

6.2. Multi-Objective Extension Experiment

An experiment comparing the multi-objective versions of AMO was conducted. The experiment used the test suites DTLZ and UF described in Section 2.7.3. The goal of the experiment is to compare the performance of the three implemented multi-objective versions of AMO and find out which version has the best performance. The experiment's setup, results, and discussion are covered in the following subsections. Based on the experiment one of the implemented algorithm versions is selected for further study.

6.2.1. Experimental Setup

The test functions used in this experiment were DTLZ1-7 and UF1-10. While the DTLZ test functions have a scalable number of objectives they will all use three objectives for this experiment. UF11-13 were not included, as they were designed to run using five objectives and the focus of this thesis is two to three objectives. DTLZ8-9 are also not included, as they are constrained problems. As described in CEC2009, for UF1-7 there can be 100 solutions in the known Pareto set produced by the algorithms. This is because these test functions have two objectives. DTLZ1-7 and UF8-10 have three objectives and can have 150 solutions in the known Pareto set. Each test function uses 30 decision variables and is ran 30 times by each algorithm. All the test functions used are described in Appendix E. The parameters used by the three algorithms are listed in Table 6.1. The stopping criterion used is the number of fitness evaluations and is set to 300,000 which is

the most commonly used number of fitness evaluations used by the algorithms in the literature review and is also used in CEC2009. Additionally, the number of iterations is set to 10,000 which is necessary to ensure the mutation chance is high and ensures that the number of fitness evaluations is met. The results from the runs on DTLZ1-7 and UF1-10 for each of the three algorithms are measured using the performance metrics GD, HV, and IGD, which are explained in Section 2.8.

Table 6.1.: Parameter settings of the algorithms.

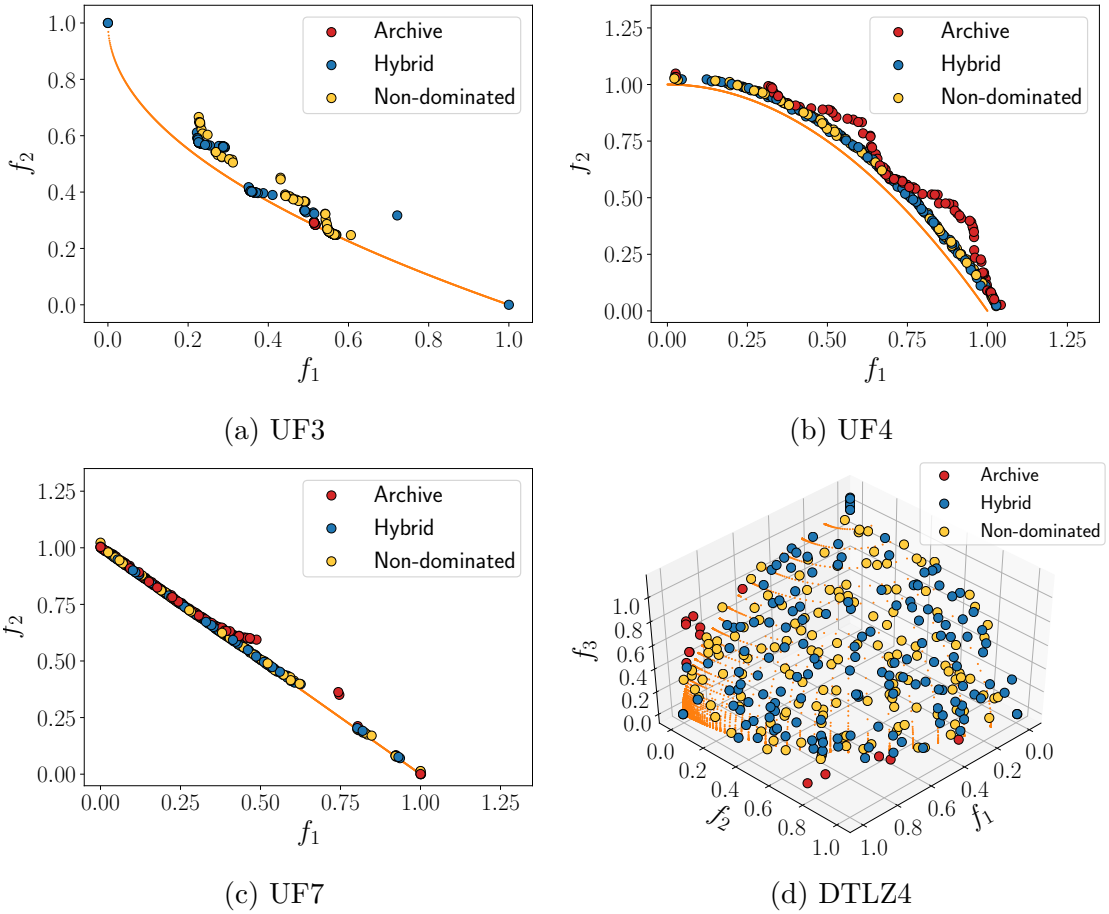
Archive		Non-dominated		Hybrid	
Population size	100 or 150	Population size	100 or 150	Population size	100 or 150
Mutation rate	0.1	Mutation rate	0.1	Mutation rate	0.1
Archive size	100 or 150			Archive size	100 or 150

6.2.2. Results and Evaluation

30 samples of the performance metrics are represented using mean and standard deviation in Tables B.1 to B.6 in Appendix B.

GD values of the non-dominated sorting version and hybrid version are similar, but overall the hybrid version has slightly better convergence. The archive version, however, clearly performs worse than the other versions on every test function except DTLZ6, UF8, and UF9. The HV values of the non-dominated sorting and hybrid versions are quite similar. The hybrid version generally performs a bit better, but the difference is negligible on most problems. However, on DTLZ3 and UF10 the hybrid version clearly performs better than the non-dominated sorting version. Non-dominated sorting version has the best mean score on DTLZ1, but the standard deviation is high on this test function which means that the results vary greatly from run to run which can skew the results. Again the archive version consistently performs worst of the three algorithms when measured using HV. The IGD scores are similar to the HV scores, with the exception that the archive version scores best on DTLZ6 with the HV performance metric, while it is beat by the hybrid version on the IGD performance metric. The only test functions where the archive performs best on several performance metrics are DTLZ6 and UF8, but the difference in scores compared to the hybrid and non-dominated sorting versions are small.

In Figures 6.1a to 6.1d the Pareto fronts obtained by running the algorithms on UF3, UF4, UF7, and DTLZ4 are shown. As can be seen in Figure 6.1a all multi-objective AMO versions are unable to spread out over the Pareto optimal front and seem to have problems converging entirely against the Pareto optimal front. The archive version, however, clearly has worse diversity than other versions. The hybrid version



(a) UF3

(b) UF4

(c) UF7

(d) DTLZ4

Figure 6.1.: Pareto fronts plotted for each algorithm on three test functions along with the Pareto optimal front.

is, in general, better at approximating the Pareto optimal front, but there is room for improvement. The Pareto fronts in Figures 6.1b and 6.1c show a similar pattern where the spread is decent, and the archive version is unable to converge as well as the other versions. As seen in Figure 6.1d the non-dominated sorting and hybrid versions are both able to find decent approximations of the Pareto optimal front, while the archive version is unable to find a diverse front.

A Friedman test [Friedman, 1937] was performed to obtain an overall ranking of the algorithms on all test functions. The Friedman test is a non-parametric statistical test that uses ranks on each test function to calculate the overall ranking. As seen in Table 6.2 the hybrid version is ranked first using HV and IGD as measurements, while the non-dominated sorting version is ranked first using GD. The difference in ranking between the hybrid version and the non-dominated sorting version is not

Table 6.2.: Friedman test ranks for the algorithm versions on all test functions

Rank	GD		HV		IGD	
	Algorithm	Value	Algorithm	Value	Algorithm	Value
1	Non-dominated	1.59	Hybrid	1.35	Hybrid	1.35
2	Hybrid	1.65	Non-dominated	1.88	Non-dominated	1.76
3	Archive	2.76	Archive	2.76	Archive	2.88

large, but noticable on HV and IGD. The archive, on the other hand, is ranked last on all metrics.

6.2.3. Discussion

One of the reasons the hybrid version can produce better solutions than the non-dominated sorting version is that the archive is used as the final Pareto set instead of the population. Over the course of the execution, the hybrid version can find a more extensive range of solutions without worrying about keeping promising solutions as they are stored in the external archive. The non-dominated sorting version, on the other hand, has to make sure that the population does not degrade in quality and therefore has a lower chance of exploring other solutions. Since the archive has a higher chance of storing a diverse set of solutions leader selection is likely better than only selecting from the non-dominated solutions in the population like the non-dominated sorting version does.

As shown in the results the archive version is with few exceptions worse on all the test functions used. One reason this can happen is the use of neighbors in the animal migration phase. Since the neighborhood is indirectly sorted as a consequence of using non-dominated sorting to combine the new and old population from the previous iteration the ring topology will use neighbors with similar qualities. This can act as a type of local search which might be favorable on some test functions. At the end of each phase, the selection between old and new solutions is also different and the method used by the archive is less extensive than using non-dominated sorting like the hybrid and non-dominated sorting versions do. The archive only compares two solutions at a time, while the other versions compare all the solutions in the merged population with each other which increases the chance of keeping decent solutions. Another reason the archive performs worse is the probability calculation. Unlike AMO the archive does not calculate probabilities for the population updating phase based on the population order but instead uses the domination count to calculate the probabilities.

6.2.4. Conclusion

Overall the hybrid version performs best out of the three algorithms. The difference between the non-dominated sorting and hybrid versions is not huge, but noticeable, mostly on the DTLZ test functions. The archive version, however, performs worst on almost all problems. Therefore, based on this experiment, the hybrid version is selected as the final version of multi-objective AMO.

6.3. Presentation of Multi-Objective Animal Migration Optimization

The Multi-Objective Animal Migration Algorithm (MOAMO) has a similar structure to the single-objective version explained in Section 5.4. The changes that have been made to the algorithm are presented in Section 6.1. In this section, the algorithm is explained in detail. The pseudocode for MOAMO can be seen in Algorithm 2. P is the main population, Q is the new population created in each phase and R is the merged population that has been sorted using non-dominated sorting. $P_{i,j}$ refers to decision variable j in solution i in population P. n is the population size, δ is a random number between 0 and 1 from the Gaussian distribution, and *rand* is a random number between 0 and 1 from the uniform distribution. S are the probabilities used when deciding if a decision variable should be changed in the population updating phase.

6.3.1. Animal Migration

The animal migration phase can be seen from line 3 to line 14 in Algorithm 2. This phase is similar to the single-objective version of this phase described in Section 5.4.1. In this phase, movement happens based on a solution's neighbors. The movement equation can be seen on line 6. When calculating the new decision variable $Q_{i,j}$, it uses $P_{i,j}$ as a starting point. The distance it will move is calculated by first selecting a decision variable from a random solution in the neighborhood $P_{neighborhood,j}$. A new neighbor is selected for each decision variable, but the same neighbor can be selected more than once. It is possible for it to select its own position from the neighborhood, leading to no change to that decision variable. The difference between $P_{neighborhood,j}$ and $P_{i,j}$ is calculated and used as the base movement distance. After the distance has been calculated, it is multiplied by δ. δ is the same for every decision variable in a solution, but a new one is generated for every solution. If a decision variable happens to go out of bounds as a result of the movement, it will be placed at the edge of the search space on the side where it went out of bounds. This can be seen on line 7-9. Because non-dominated sorting is used when selecting which solutions will make it to the next phase, the solutions

Algorithm 2 Multi-Objective Animal Migration Optimization

Initialize P with n random solutions
while stopping criterion is not met **do**
    for $i = 1$ to n **do** ▷ Animal migration
        Generate δ from the Gaussian distribution
        for $j = 1$ to number of decision variables **do**
            $Q_{i,j} = P_{i,j} + \delta \times (P_{neighborhood,j} - P_{i,j})$
            if $Q_{i,j}$ is out of bounds **then**
                Place $Q_{i,j}$ at the edge of the search space
            end if
        end for
    end for
    $R = non\_dominated\_sorting(P \cup Q)$
    $P = R[1 : n]$
    Update the archive
    Perform mutation operator
    Calculate probabilities S ▷ Population updating
    for $i = 1$ to n **do**
        for $j = 1$ to number of decision variables **do**
            if $rand > S_i$ **then**
                Generate two random population indices where $r1 \neq r2 \neq i$
                $Q_{i,j} = P_{r1,j} + rand \times (P_{best,j} - P_{i,j}) + rand \times (P_{r2,j} - P_{i,j})$
                if $Q_{i,j}$ is out of bounds **then**
                    Place $Q_{i,j}$ at the edge of the search space
                end if
            else
                $Q_{i,j} = P_{i,j}$
            end if
        end for
    end for
    $R = non\_dominated\_sorting(P \cup Q)$
    $P = R[1 : n]$
    Update the archive
end while

in $P_{neighbourhood}$ are likely to be of similar quality as P_i.

6.3.2. Mutation

The mutation step can be seen on line 15 in the pseudocode. This mutation is based on the mutation used by MOPSO explained in Section 2.6.4. The mutation step happens between the animal migration and the population updating phase. The difference between MOPSO's mutation operator and the one used in MOAMO is the mutation range calculation. Instead of using $(1 - \frac{\text{current iteration}}{\text{max iterations}})^{5/\text{mutation rate}}$ the mutation range is lowered to $(1 - \frac{\text{current iteration}}{\text{max iterations}})^{1/\text{mutation rate}}$.

6.3.3. Population Update

The population update phase can be seen from line 16 to 32 in Algorithm 2. This phase simulates how animals leave and join groups. Similarly to AMO, at of the start of the phase the probabilities S are calculated for the solutions in the main population P. This is done using Equation (6.1).

$$S_i = \frac{n - H_i}{n} \tag{6.1}$$

Here H_i is the number of solutions ahead of i after non-dominated sorting has been performed and n is the population size. This lets solutions in the lower fronts with the highest crowding distance have a lower chance of being changed. This probability is used when looping over the decision variables of a solution. This can be seen on line 19. The probability S_i that has been calculated is the chance that a decision variable $Q_{i,j}$ will be copied from $P_{i,j}$ with no changes. If this is not the case, a new decision variable will be created using the equation shown on line 21. It uses the current decision variable $P_{i,j}$, two decision variables that are selected randomly from the population $P_{r1,j}$ and $P_{r2,j}$, and $P_{best,j}$. $P_{best,j}$ is a solution selected from the archive using binary tournament selection based on the solution's crowding distance. The new position uses $P_{r1,j}$ as its starting point. This position is then changed twice. First, it multiplies the difference between $P_{best,j}$ and $P_{i,j}$ with a random number between 0 and 1 and adds that. Then it multiplies the difference between $P_{r2,j}$ and $P_{i,j}$ with a new random number between 0 and 1 and adds that as well. Like in the animal migration phase, out of bounds checking happens just after the decision variable has been calculated, and if it is out of bounds it is moved to the edge of the search space on the side where it went out of bounds.

6.3.4. End of Each Phase

At the end of each phase, non-dominated sorting and archiving are performed. This can be seen on lines 12-14 and 30-32 in Algorithm 2. Because the end of both phases is identical, only the pseudocode for the animal migration phase is referenced with line numbers in this section. On line 12, the main population P is combined with the new population Q that was created during the phase. This merged population is then sorted using non-dominated sorting, sorting it first by rank, and within each rank by crowding distance. The resulting population is represented as R in the pseudocode. On line 13 R is then cut in half, keeping only the best n solutions. This population replaces P as the new main population that will be used for the next phase. As a side effect of using non-dominated sorting the population always remained sorted after each phase.

Archiving happens on line 14 in the pseudocode. The archive stores all the non-dominated solutions. If the number of non-dominated solutions is higher than the max archive-size it removes the solutions with the lowest crowding distance. The one change that has been made to the archive, compared to the SMPSO archive it is based on, is that when pruning multiple solutions at once it does not recalculate the crowding distance after a solution is pruned. The reason for this was that recalculating crowding distance every time increased the computational complexity while having a marginal impact on the results.

6.4. Algorithm Analysis and Discussion

In this section, some aspects of MOAMO are analyzed and discussed.

6.4.1. Few Tuning Parameters

The only tuning parameter AMO has is the population size, and the only parameters added by the techniques used is the number of solution stored in the archive and the mutation rate. Since the archive size determines the number of the solutions in the known Pareto front the only parameters that can freely be changed are the population size and mutation rate. Archive size is by default the same as the population size. Few parameters make MOAMO easy to use on a variety of problems, but on the other hand, might make it harder to adjust to problems types MOAMO does not handle well.

6.4.2. Computational Complexity of MOAMO

MOAMO uses two multi-objective techniques for solving multi-objective problems unlike many other algorithms in the literature which only use one. Non-dominated

sorting and an external archive are both techniques that can be used alone so using both in a combination can be considered too computationally heavy. However, the computational complexity of both non-dominated sorting and the external archive is $O(mn^2)$, where m is the number of objectives and n is the population size, and since they are not used nested the overall complexity does not change. However, when the population increases the runtime difference might be noticeable when using MOAMO versus an algorithm like NSGA-II which only performs non-dominated sorting.

Non-dominated sorting and updating of the archive are run at the end of each phase. Non-dominated sorting is also used to calculate the probabilities used in the populating updating phase, so in total non-dominated sorting is ran three times for each iteration while the archive is updated two times. Unlike many other algorithms, MOAMO has two phases per iteration while other algorithms usually only have one phase. So even though MOAMO runs non-dominated sorting three times per iteration and updates the archive twice per iteration this would correspond to half of that in other algorithms.

6.4.3. Multi-Objective Extension of AMO

Extending AMO to handle optimizing of multiple objectives worked quite well partly because it generates a new population after each phase which can be used in non-dominated sorting and updating of the archive. Using non-dominated sorting and an external archive are common ways to extend population-based algorithms. From the literature review, several similar single-objective algorithms were extended with success and AMO did not pose any additional problems that were not solved by other algorithms. The algorithm itself was not radically changed and remained more or less intact. The changes implemented did not alter the idea behind the algorithm and were required for AMO to work on multi-objective problems. Calculating the population update probability using non-dominated sorting was a natural choice as the technique has been well proven and allows to order multi-objective solutions. Selecting the best solution from the archive, during the population updating phase, encourages the population to be more diverse in a similar way to how AMO encourages convergence by selecting the best-known solution. The largest difference between MOAMO and AMO is the evaluation of the new solutions, where formerly the new solutions were evaluated only against the solution used to create it, now the whole population is evaluated against each other in MOAMO using non-dominated sorting.

7. Multi-Objective Comparison Experiment

This chapter aims to answer **RQ3** by comparing MOAMO's performance with other competitive multi-objective algorithms in an experiment. This experiment will follow the same structure as the previous two experiments. The experimental setup is explained at the start of the chapter. Following this, the results are presented and evaluated. At the end of the chapter, some interesting results are discussed in depth, followed by a conclusion.

7.1. Experimental Setup

To generate test data jMetal[1] [Durillo and Nebro, 2011] was used. jMetal is an open source framework for multi-objective optimization written in Java. jMetal was used because it has a wide range of algorithms and test functions along with several common performance metrics. Additionally, using jMetal made it easier to run the experiment with the exact setup needed which also made it easier to verify that the comparison was valid. Since jMetal was used for all algorithms except MOAMO, which uses the framework described in Chapter 4, it was important to verify that every test function and performance metric implemented in the framework produced the same output as the ones in jMetal. To verify the correctness of the test functions 30 sets of input variables were generated and used to calculate fitness values in both jMetal and the framework. The fitness values were then compared to make sure they were identical. This was done for every test function used in the experiment. The performance metrics were also tested in a similar fashion. Pareto fronts were generated with jMetal and measured using both the implemented performance metrics and jMetal's. By adding these tests as unit tests, it could be ensured that there were no regressions related to performance metrics or test functions. Additionally, all the data produced by all the algorithms running in jMetal was verified with the test functions and performance metrics in the framework to ensures that there were no discrepancies between the implementations and that the results

[1] jMetal version used:
`https://github.com/jMetal/jMetal/tree/d2282523062c74d715a36a088bf7c531e0783068`

produced by MOAMO are comparable with results from jMetal's algorithm.

The experiment is based on the setup used in CEC2009 [Zhang et al., 2008]. Each test function uses 30 decision variables and is ran 30 times by each algorithm with 300,000 fitness evalutions as the stopping criterion.

7.1.1. Algorithms

MOAMO will be compared against eight other multi-objective algorithms which are all implemented in jMetal. Some tried and true algorithms that will be in the experiment are NSGA-II, PAES, and SPEA2. They are described in Section 2.6. Two algorithms from the literature review will be in the experiment: MOEA/DD (Section 3.2.4) and SMPSO (Section 3.2.3). Additionally, three novel algorithms that have not been previously described were included. AbySS [Nebro et al., 2008] is a hybrid multi-objective algorithm that combines scatter search structure with crossover and mutation operators. It uses Pareto dominance and an archive to store solutions. MOCell [Nebro et al., 2009b] is a cellular genetic algorithm with an external archive for solving multi-objective problems. The archive is, like SMPSO, using crowding distance as a metric for pruning. NSGA-III [Deb and Jain, 2014] is based on the well known NSGA-II, but with changes to the selection operator for diversity maintenance. NSGA-III uses a set of reference points that are updated and maintained to be evenly spread out over the search space. By associating these reference points with the population, it uses the number of solutions associated with reference points to select solutions instead of using the crowding operator like NSGA-II.

Unlike the other algorithms included in the experiment, the MOEA/DD version in jMetal does not only use the non-dominated solutions as the final solution set. Including dominated solutions can affect the calculation of the performance metrics in both positive and negative ways. Calculation of GD is likely to suffer since it keeps more solutions far away from the Pareto optimal front, while calculation of IGD can improve as the diversity is likely to be greater. HV, on the other hand, is unaffected as the dominated solutions are inside of the calculated volume and therefore do not contribute to it.

7.1.2. Test Suites

The test suites that are used to test the performance of the algorithms are ZDT, DTLZ, and UF. They are all explained in Section 2.7.3. The test functions that are used are DTLZ1-7, UF1-10 and ZDT1-4 and 6. DTLZ8-9 and UF11-13 are not included for the reasons explained in Section 6.2.1. ZDT5 is not included as MOAMO does not support binary problems. The ZDT and UF1-7 test functions

all have two objectives, while UF8-10 and DTLZ1-7 have three objectives.

7.1.3. Algorithm Parameters

The parameters used in the original articles of the algorithms were used except for the parameter that decided the number of solutions in the final Pareto set obtained by the algorithm. For most algorithms this is either the population size or archive size. Like in CEC2009 the number of solutions in the final Pareto front was set based on the number of objectives. 100 for two objectives and 150 for three objectives. Since MOCell only supports integers for the grid size 144 (12x12) was used instead of 150. Parameters used for all the algorithms are shown in Table 7.1.

The MOEA/DD version in jMetal did not have the exact number of weights needed for three objectives, so instead the closest number of weights above 150 was used which ended up being 300. The number of solutions returned is still 150. Since the NSGA-III version in jMetal bases the output on the number of divisions, the code was altered to use the specified population size instead.

Table 7.1.: Parameter settings of the algorithms. DV stands for number of decision variables.

AbYSS		MOCell		MOEA/DD	
Population size	20	Population size	100 or 144	Population size	100 or 150
Mutation rate	1/DV	Mutation rate	1/DV	Mutation rate	1/DV
Crossover rate	0.9	Crossover rate	0.9	Crossover rate	1
Subranges	4	Archive size	100 or 150	Neighborhood size	20
RefSets size	10 + 10			Neighborhood selection	0.9
Archive size	100 or 150			Weights	100 or 300
NSGA-III		NSGA-II		PAES	
Population size	100 or 150	Population size	100 or 150	Archive size	100 or 150
Mutation rate	1/DV	Mutation rate	1/DV	Mutation rate	1/DV
Crossover rate	0.9	Crossover rate	1	Bi sections	5
Divisions	99 or 16				
SMPSO		SPEA2		MOAMO	
Population size	100	Population size	100 or 150	Population size	100 or 150
Mutation rate	1/DV	Mutation rate	1/DV	Mutation rate	0.1
Crossover rate	1	Crossover rate	1	Archive size	100 or 150
Archive size	100 or 150				

7.1.4. Performance Metrics

The three performance metrics used in this experiment are generational distance (GD), inverted generational distance (IGD) and hypervolume (HV). See Section 2.8 for more details about the performance metrics. Since measuring multi-objective

problems using performance metrics like HV and IGD measures convergence and diversity in one metric, there was a need for a metric that only measures one. GD is a performance metric used solely for measuring convergence and was included to make it easier to differentiate between poor convergence or poor diversity. For example, a solution with poor HV and great GD will have poor diversity.

Table 7.2.: The rank of MOAMO on every test function measured with GD, HV, and IGD.

Test function	GD	HV	IGD
DTLZ1	8	9	9
DTLZ2	8	7	6
DTLZ3	6	7	8
DTLZ4	9	6	5
DTLZ5	6	5	6
DTLZ6	2	2	2
DTLZ7	4	4	3
UF1	4	1	1
UF2	1	1	1
UF3	6	2	2
UF4	2	1	1
UF5	5	1	1
UF6	4	2	1
UF7	2	1	2
UF8	5	4	3
UF9	4	4	3
UF10	4	6	6
ZDT1	7	7	7
ZDT2	6	8	7
ZDT3	5	4	4
ZDT4	7	8	8
ZDT6	6	6	7

7.1.5. Friedman Test Ranks

To compare the overall results non-parametric statistical tests were used. Based on recommendations in Derrac et al. [2011] the Friedman aligned ranks test [Hodges et al., 1962] was used for the individual test suites while the standard Friedman test [Friedman, 1937] was used for the overall results. The Friedman (aligned ranks) test uses the rank obtained on each test function to calculate the overall rank for a test suite or the overall results. Additionally, to check if MOAMO has significantly better or worse results compared to other algorithms, Holm's post hoc test [Holm,

1979] with a significance level of $p = 0.05$ was used. This comparison is useful to see if MOAMO is generally better than some algorithms or if they are better and worse on different test functions.

7.2. Results and Evaluation

In this section, the results are presented. The results of each test suite are presented separately, and at the end of the section, the overall results from the experiment are presented.

A Friedman aligned rank test is presented for every test suite in Tables 7.3 to 7.5, and a Friedman rank test of the overall results can be seen in Table 7.6. The tables are sorted according to the Friedman value where lower is better. The plots in this section show the Pareto optimal front in orange, while results from other algorithms use a predefined color. Tables showing the mean and standard deviation of 30 runs of every test function and performance metrics can be seen in Appendix C. Additionally, MOAMO's ranking on every test function is shown in Table 7.2.

Table 7.3.: Friedman aligned ranks on ZDT

Rank	GD		HV		IGD	
	Algorithm	Value	Algorithm	Value	Algorithm	Value
1	NSGA-III	14.00	SMPSO	13.80	SMPSO	15.80
2	MOCell	17.00	MOCell	15.80	MOCell	17.00
3	AbYSS	18.00	AbYSS	17.60	AbYSS	17.60
4	**MOAMO**	20.00	MOEA/DD	20.40	NSGA-III	21.20
5	MOEA/DD	20.40	NSGA-III	20.40	SPEA2	21.80
6	NSGA-II	20.80	SPEA2	23.20	**MOAMO**	22.80
7	SMPSO	22.60	**MOAMO**	24.80	NSGA-II	23.20
8	SPEA2	35.80	NSGA-II	28.00	MOEA/DD	24.60
9	PAES	38.40	PAES	43.00	PAES	43.00

7.2.1. Results on ZDT

As seen in Table 7.3 MOAMO is ranked fourth on GD, seventh on HV, and sixth on IGD. Except for PAES, all the algorithms in the experiment performed well on the ZDT test suite. The differences in values measured using HV and IGD are small, but overall SMPSO is the most consistent algorithm for finding near-optimal Pareto fronts. Except for on ZDT4, MOAMO has competitive results on all test functions, and on ZDT4 it is not far behind. The GD values differ a bit more which suggests that some algorithms convergences worse than others, but by looking at the plotted Pareto fronts in Figure 7.1, it is evident that all the solutions are close

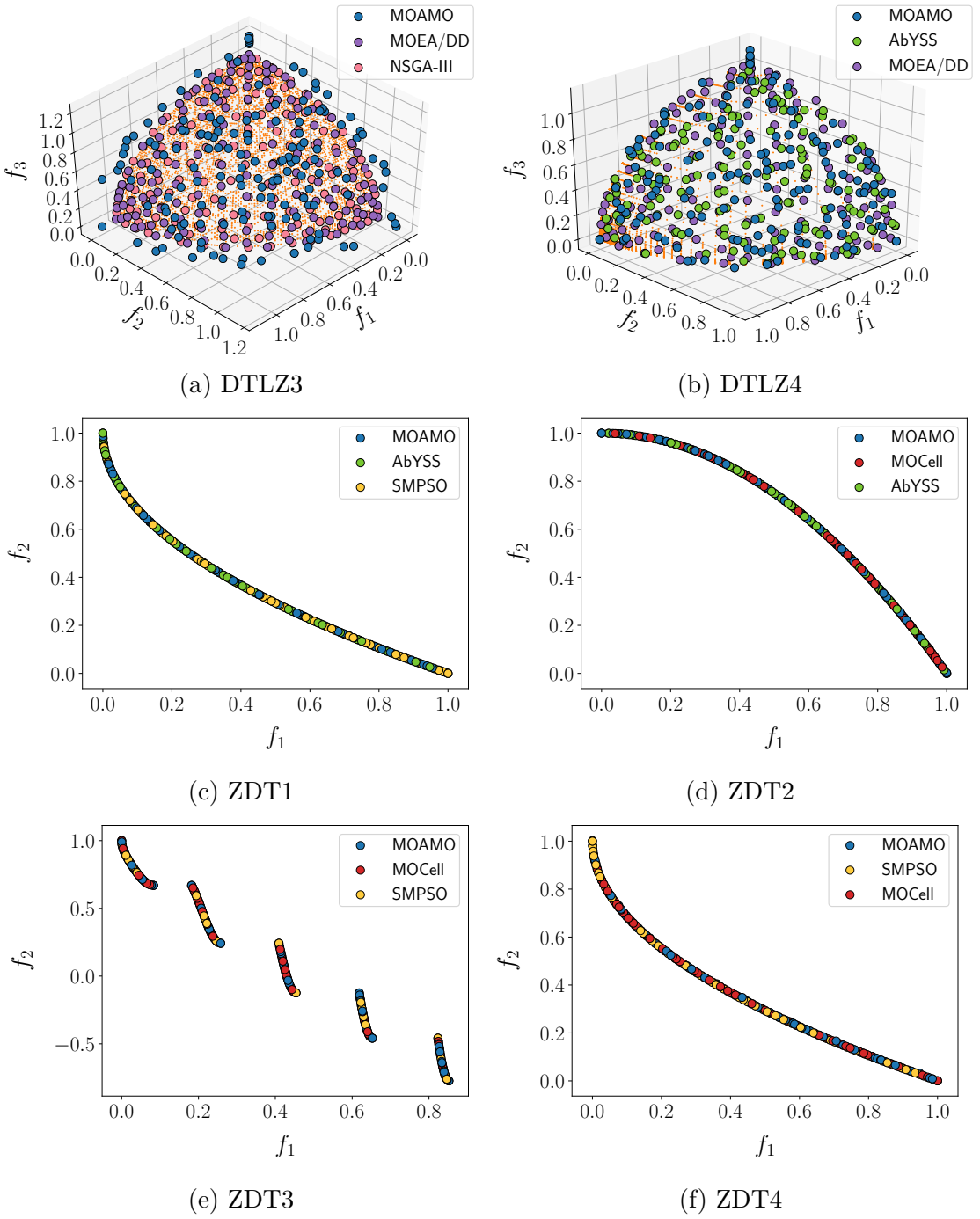
to the Pareto optimal front. The reason PAES performs worse than the other algorithms is that it is in many cases unable to find solutions spread out over the whole Pareto optimal front. Using Holm's post hoc test on the Friedman aligned ranks on each metric shows that there are no statistically significant differences between MOAMO and any of the other algorithms on the ZDT test suite.

7.2.2. Results on DTLZ

MOAMO performs below average on most of the DTLZ test suite, getting rank seven on GD and HV, and eight on IGD as shown in Table 7.4. The only test function where it performs well is DTLZ6 where it gets second best on every performance metric. By inspecting the obtained Pareto fronts of MOAMO, a clearer view of its performance can be seen. MOAMO's convergence and diversity on DTLZ2 and DTLZ4-5 are fairly good, but other algorithms usually provide better solutions. On DTLZ1 and DTLZ3 MOAMO is unable to consistently converge close enough to the Pareto optimal front. This is noticeable in Figure 7.1a which shows the plotted Pareto fronts of MOAMO, MOEA/DD, and NSGA-III on DTLZ3. Here it can be seen that while MOAMO achieves good diversity it is unable to converge as close to the Pareto optimal front as the other algorithms. MOEA/DD performs the best on DTLZ1-3 and the worst on DTLZ5-7. On DTLZ2, AbYSS gets a pretty good HV value, placing rank 3. However, its IGD is pretty bad compared to the rest, placing at rank 8. This can happen when the front has converged, but there are areas in the front where there are no solutions. As on ZDT, no statistically significant differences on the ranks were found using Holm's post hoc test.

Table 7.4.: Friedman aligned ranks on DTLZ

Rank	GD		HV		IGD	
	Algorithm	Value	Algorithm	Value	Algorithm	Value
1	AbYSS	24.86	NSGA-III	24.57	MOEA/DD	25.43
2	PAES	26.00	MOEA/DD	24.86	NSGA-II	25.86
3	NSGA-II	26.43	NSGA-II	25.29	SMPSO	27.00
4	NSGA-III	27.14	AbYSS	27.43	SPEA2	27.86
5	SPEA2	30.57	SMPSO	28.29	NSGA-III	29.71
6	MOCell	33.43	SPEA2	31.43	MOCell	30.00
7	**MOAMO**	35.71	**MOAMO**	35.57	AbYSS	34.71
8	MOEA/DD	40.00	MOCell	37.43	**MOAMO**	35.71
9	SMPSO	43.86	PAES	53.14	PAES	51.71



(a) DTLZ3

(b) DTLZ4

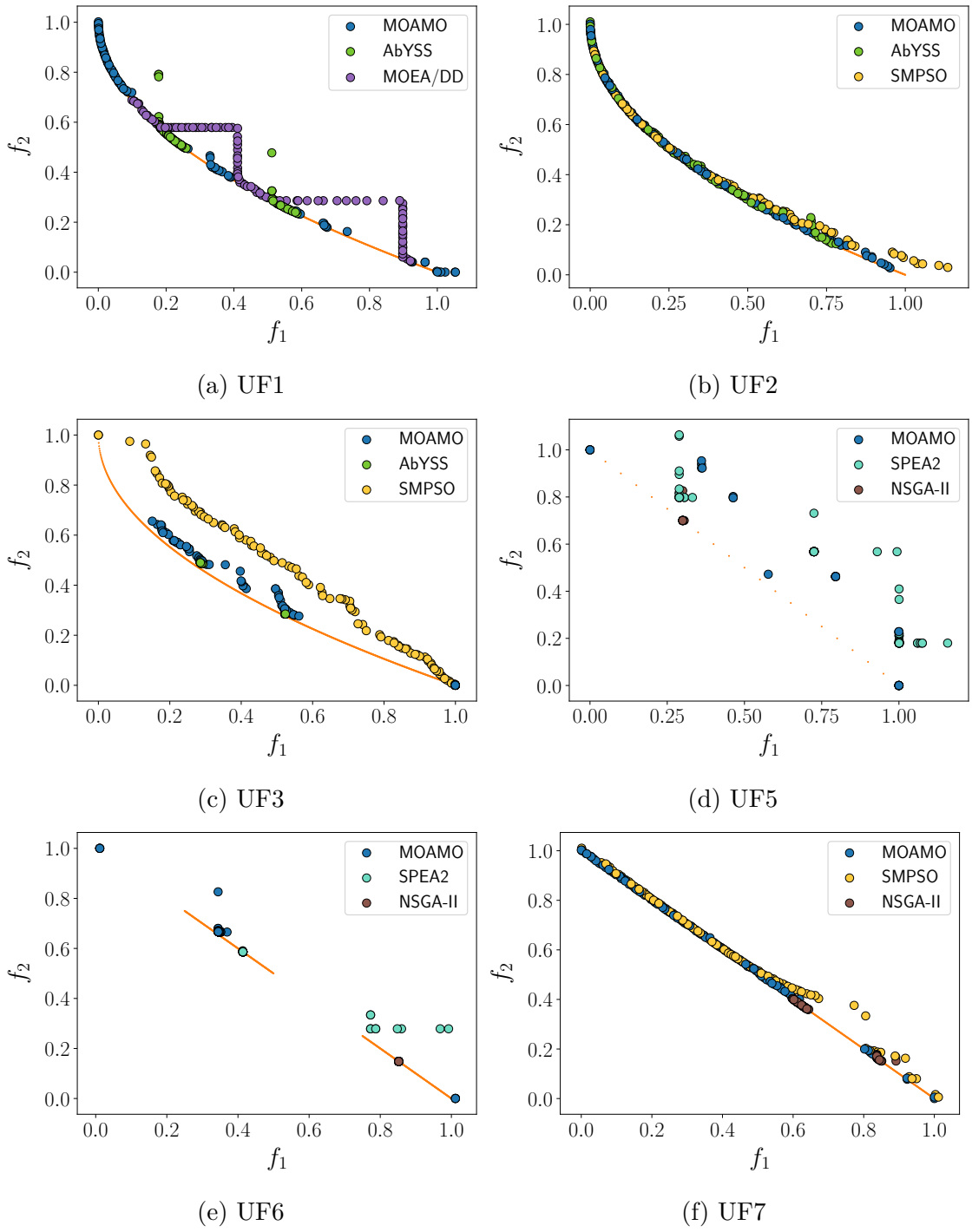
(c) ZDT1

(d) ZDT2

(e) ZDT3

(f) ZDT4

Figure 7.1.: Plotted Pareto fronts of MOAMO and the two best algorithms on DTLZ and ZDT test functions



(a) UF1 (b) UF2

(c) UF3 (d) UF5

(e) UF6 (f) UF7

Figure 7.2.: Plotted Pareto fronts of MOAMO and the two best algorithms on UF test functions

Table 7.5.: Friedman aligned ranks on UF

Rank	GD		HV		IGD	
	Algorithm	Value	Algorithm	Value	Algorithm	Value
1	NSGA-III	25.50	**MOAMO**	22.10	**MOAMO**	20.80
2	PAES	38.80	NSGA-III	35.10	NSGA-II	36.70
3	MOEA/DD	39.70	MOEA/DD	37.40	SPEA2	36.80
4	**MOAMO**	40.10	AbYSS	38.50	SMPSO	38.40
5	SPEA2	43.20	SPEA2	41.20	MOEA/DD	42.20
6	AbYSS	45.60	NSGA-II	43.90	AbYSS	47.60
7	NSGA-II	48.00	SMPSO	50.90	NSGA-III	50.90
8	MOCell	53.20	MOCell	60.90	MOCell	53.40
9	SMPSO	75.40	PAES	79.50	PAES	82.70

7.2.3. Results on UF

Overall MOAMO performs best on the UF test suite, getting rank one on HV and IGD, and rank four on GD as shown in Table 7.5. However, as can be seen from the plotted Pareto fronts shown in Figure 7.2 it does not find near-optimal Pareto fronts for the majority of the test functions. The results show that finding decent Pareto fronts on the UF test suite is much harder than both ZDT and DTLZ. A typical pattern displayed by the algorithms included in this experiment is that they are unable to find a diverse Pareto front, which either shows as gaps in the Pareto front (UF1, UF2, UF7) or small areas with clustered solutions (UF3, UF5, UF6). MOAMO shows many of the same patterns as other algorithms on UF1-7, but it usually has better diversity. On UF8, which has three objectives, MOEA/DD is the only algorithm able to consistently find Pareto fronts with both great convergence and diversity. MOAMO still produces decent results on UF8 compared to the other algorithms. Using Friedman aligned ranks test with Holm's post hoc shows that MOAMO has statistically significantly better results than PAES and MOCell on the UF test suite with HV and IGD metrics. Additionally, MOAMO has statistically better GD ranking than SMPSO.

7.2.4. Overall Results

The results from the Friedman test on all the test functions in the experiment is shown in Table 7.6. MOAMO is ranked third on the GD and HV performance metrics and ranked second using IGD. While MOAMO does not perform the best on any of the performance metrics overall, it is the only algorithm that ranks top three on every performance metric.

Table 7.6.: Overall algorithm Friedman rank on all test functions

Rank	GD		HV		IGD	
	Algorithm	Value	Algorithm	Value	Algorithm	Value
1	NSGA-III	3.27	AbYSS	3.77	SMPSO	3.45
2	AbYSS	4.05	MOEA/DD	4.18	**MOAMO**	4.18
3	**MOAMO**	4.73	**MOAMO**	4.32	SPEA2	4.36
4	NSGA-II	4.86	SMPSO	4.36	MOEA/DD	4.45
5	MOEA/DD	5.00	NSGA-III	4.50	AbYSS	4.59
6	MOCell	5.18	NSGA-II	4.95	NSGA-II	4.91
7	PAES	5.45	SPEA2	5.27	MOCell	5.14
8	SPEA2	5.73	MOCell	5.41	NSGA-III	5.14
9	SMPSO	6.73	PAES	8.23	PAES	8.77

7.3. Discussion

This section discusses some interesting aspects of MOAMO's results on the test functions used in the experiment.

7.3.1. ZDT Performance and MOAMO's Pruning Technique

The ZDT test suite is different from the other test suites in that it is a lot simpler to get decent results. Most algorithms find decent Pareto fronts with a lot fewer function evaluations than 300,000. MOAMO converges fast towards the optimal front, using only about 25,000 fitness evaluations on ZDT1-3. On ZDT6 it uses about 50,000 and on ZDT4 around 180,000. Knowing this, 300,000 evaluations might seem unnecessarily high for this test suite. But this was deliberate, as the algorithms that manage to converge quickly to the Pareto optimal front would have a lot of fitness evaluations left to get a great spread along the front.

MOAMO does not manage to find solutions as close to the Pareto optimal front as many of the other algorithms. It also does not manage to get as good spread between the solutions. The reason for this is that when MOAMO is pruning solutions from the archive, it does not recalculate crowding distance every time it removes a solution. This was a deliberate choice made during the creation process, as the crowding distance calculation has a complexity of $O(mn \log n)$, where m is the number of objectives and n is the archive size, and recalculating the crowding distance each time would give pruning a complexity of $O(mn^2 \log n)$. A comparison of calculating crowding distance one time vs calculating it each time a solution is pruned can be seen in Table 7.7. These runs used the settings

Table 7.7.: The left colum of every performance metric shows the normal version of MOAMO that only calculates crowding distance once before pruning. The right column shows a version of MOAMO that recalculates crowding distance every time a solution is removed from the archive.

Test function	Measure	GD		HV		IGD	
		Normal	Modified	Normal	Modified	Normal	Modified
ZDT1	Mean	1.52E-04	**1.44E-04**	6.60E-01	**6.61E-01**	1.78E-04	**1.44E-04**
	SD	2.26E-05	2.88E-05	2.51E-04	2.05E-04	1.41E-05	1.56E-06
	Rank	2	1	2	1	2	1
ZDT2	Mean	8.44E-05	**8.20E-05**	3.27E-01	**3.28E-01**	1.77E-04	**1.50E-04**
	SD	1.16E-05	1.03E-05	3.01E-04	1.82E-04	1.04E-05	1.97E-06
	Rank	2	1	2	1	2	1
ZDT3	Mean	1.05E-04	**1.02E-04**	5.16E-01	**5.16E-01**	1.28E-04	**1.05E-04**
	SD	8.50E-06	5.98E-06	6.48E-05	1.26E-05	7.96E-06	1.59E-06
	Rank	2	1	2	1	2	1
ZDT4	Mean	1.06E-03	**1.87E-04**	6.48E-01	**6.60E-01**	4.45E-04	**1.55E-04**
	SD	3.16E-03	1.19E-04	4.24E-02	1.74E-03	9.72E-04	2.66E-05
	Rank	2	1	2	1	2	1
ZDT6	Mean	1.23E-03	**8.04E-04**	4.00E-01	**4.01E-01**	1.77E-04	**1.43E-04**
	SD	5.03E-03	2.28E-03	7.01E-04	9.49E-04	1.52E-05	4.38E-06
	Rank	2	1	2	1	2	1

Table 7.8.: Time MOAMO needed to complete 30 runs with 300,000 fitness evaluations each on every test function in the test suites.

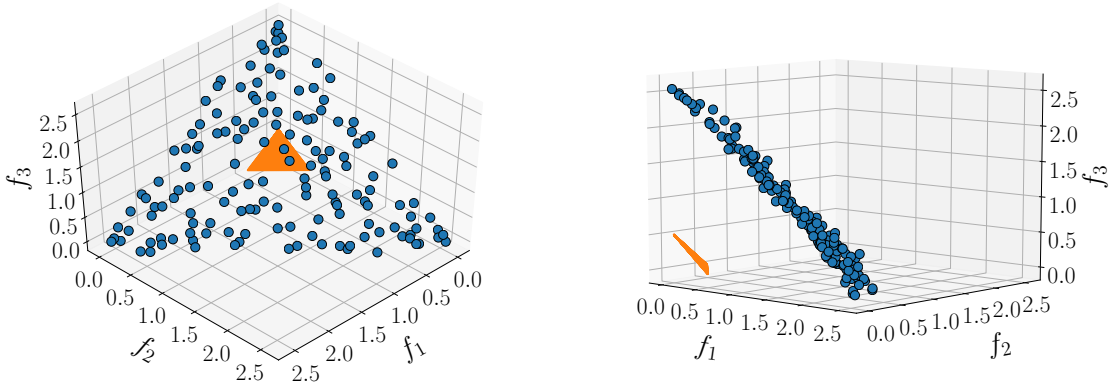
Test Suite	Normal	Modified
DTLZ	9 min 40 sec	62 min 30 sec
UF	10 min 50 sec	35 min 05 sec
ZDT	4 min 33 sec	36 min 41 sec

described in Section 7.1. The results show that recalculating crowding distance is better according to every performance metric on all the ZDT test functions. Additionally, the standard deviation is in almost all cases lower which means that the results are more consistent. This suggests that recalculating improves diversity, which makes sense as it would allow for more accurate pruning. However, the improved accuracy comes at the cost of the added complexity as mentioned earlier. Table 7.8 shows the time MOAMO needed to complete 30 runs with 300,000 fitness evaluations each on all the test suites with and without the pruning change. The time difference is massive, especially for DTLZ and ZDT. For DTLZ the archive size is 150 which cause the pruning calculation to take more time. It has the biggest impact on ZDT where the time usage is about nine times as long when recalculating crowding distance every time. This is because MOAMO finds a lot

of non-dominated solutions on ZDT and spends a lot of time pruning the archive. The difference in results was only substantial on the ZDT test suite. This is most likely because MOAMO does not have as much time to work really close to the Pareto optimal front on the other test suites. Overall, the slight improvement in results was not considered worth it, as the added computational complexity had a massive impact on the runtime of the algorithm.

7.3.2. Convergence on DTLZ1

MOAMO performs poorly with all performance metrics on DTLZ1. According to Deb et al. [2005] DTLZ1 is a good test function for testing convergence. DTLZ1 has $11^k - 1$ local Pareto optimal fronts in its search space where $k = \textit{decision variables} - \textit{objectives} + 1$. Results from an ordinary run on DTLZ1 can be seen in Figure 7.3. In Figure 7.3a it can be seen that it has quite good diversity, but as shown in Figure 7.3b it has not managed to converge close to the Pareto optimal front. On some runs MOAMO manages to converge close to the Pareto optimal front, getting similar results to other algorithms, but it can not do this consistently with 300,000 fitness evaluations. If the number of fitness evaluations is increased, it manages to converge close to the global Pareto optimal front consistently. This suggests that MOAMO does not get completely stuck at a local Pareto optimal front, but rather just converges slowly. The algorithm also performs better on this particular problem if the animal migration phase is turned off. This happens because the population updating phase creates new solutions with the help of non-dominated solutions in the archive, giving it a bigger chance to move towards solutions with better fitness values, converging faster.



(a) Diversity (b) Convergence

Figure 7.3.: MOAMO diversity and convergence on DTLZ1

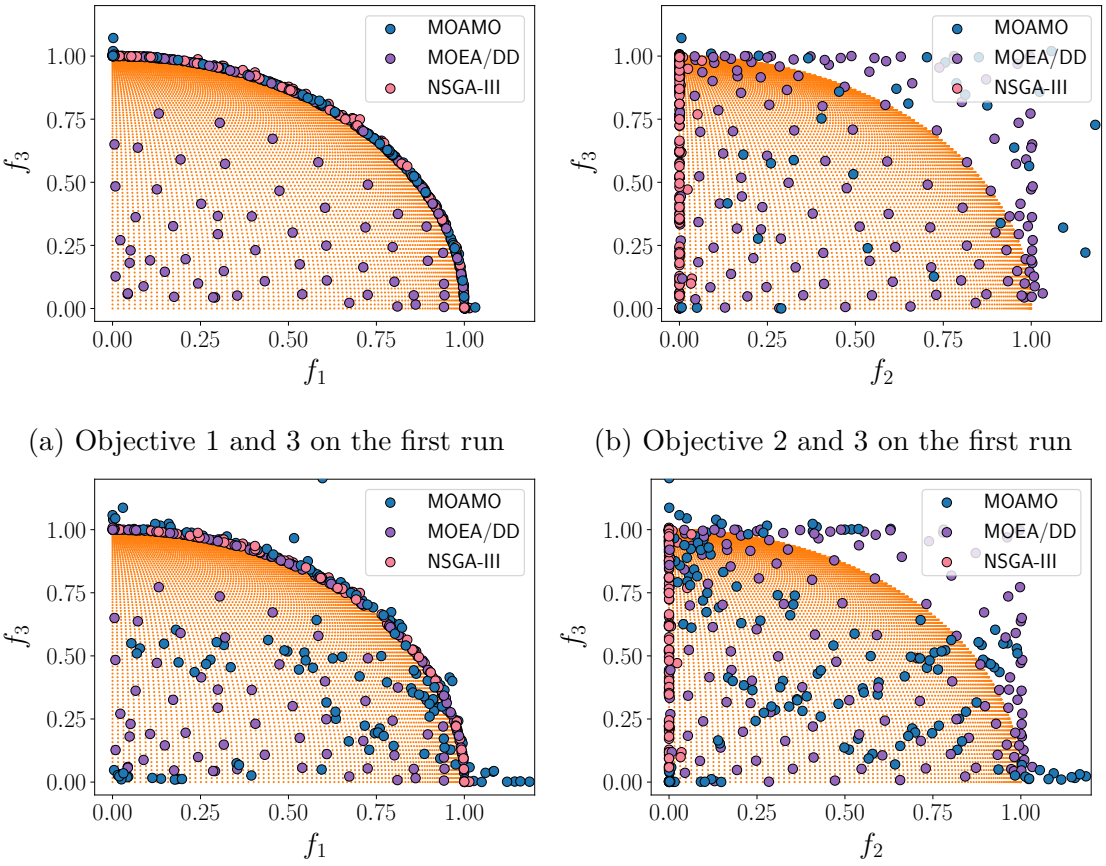
7.3.3. Excellent Performance on UF1-7

Using the HV and IGD performance metrics MOAMO scores either best or second best on UF1-7. As can be seen in Figure 7.2 MOAMO is able to consistently converge against the Pareto optimal front while maintaining a certain degree of diversity. The difference in converge on the left side of UF2 is marginal between the algorithms, but MOAMO is the only one able to find solutions close to the Pareto optimal front on the right side. While MOAMO is unable to find a perfect spread on most of the UF test functions the other algorithms included in the experiment are even more prone to only finding small clusters of solutions here and there and, therefore, produce worse results than MOAMO. In general, MOAMO is able to find the outer points of the Pareto optimal front when other algorithms are unable to. For example on UF6 MOAMO is the only algorithm that is able to reliably find solutions close to the optimal solution on the far left of the disconnected Pareto optimal front. Other algorithms are either unable to find solutions in this area of the objective space or only rarely do so. By looking at the solutions found by MOAMO while still optimizing it can be seen that it converges rapidly against the solutions seen in the final Pareto front. This can be advantageous since fewer fitness evaluations are spent searching less optimal parts of the search space, but also results in worse diversity on some problems. Calculating the IGD value for each iteration shows that most of the fitness evaluations are used to gradually improve an already fairly decent Pareto front. On many of the UF test functions, it is this relatively minor difference that differentiates MOAMO from the other algorithms.

7.3.4. Poor Diversity on UF8

In most cases, MOAMO is unable to spread out over the Pareto optimal front on UF8 properly. Figures 7.4a and 7.4b shows a plot of two pairs of objectives of MOAMO, NSGA-III, and MOEA/DD on UF8. MOAMO is only able to find solutions along the top edge of the Pareto optimal front (seen in Figure 7.4a) plotted in orange. NSGA-III has a similar pattern but has no variance in the fitness values for the second objective whereas MOAMO's fitness ranges from 0 to 6. On the other hand, MOEA/DD does not exhibit this pattern and is consistently able to find solutions spread out over the sphere-like Pareto optimal front. As can be seen in Figures 7.4c and 7.4d MOAMO is sometimes able to find some solutions along the bottom of the Pareto optimal front, but the diversity is still fairly bad. MOEA/DD is the only algorithm included in the experiment that is able to consistently find decent solutions for this problem which suggest that decomposition techniques are more fitting for this type of problem. Even if MOAMO is evaluated for 3,000,000 fitness evaluations, ten times the number used in the experiment in Section 7.2, it is unable to find a spread like MOEA/DD on UF8 which means that MOAMO is

not suitable for these types of problems. A similar pattern is noticeable on UF10, but in this case, MOEA/DD is also unable to spread out over the Pareto optimal front consistently. The Pareto optimal front of UF10 is of the same shape as UF8, but none of the algorithms included in the experiment can find solutions with a spread over the sphere-like Pareto optimal front and most only find solutions at the edge of the Pareto optimal front. Despite the fact that no other algorithms can solve this problem accurately MOAMO does not score particularly well.



(a) Objective 1 and 3 on the first run

(b) Objective 2 and 3 on the first run

(c) Objective 1 and 3 on the second run

(d) Objective 2 and 3 on the second run

Figure 7.4.: 2D objective plots of two different runs of MOAMO on UF8 with MOEA/DD and NSGA-III for comparison

7.4. Conclusion

Compared to the other algorithms included in the experiment MOAMO produces competitive results on most test functions and even performs the best on five test functions measured using the HV and IGD metric. Friedman aligned ranks show

that MOAMO performs best overall on UF measured with IGD and HV. While the results on the ZDT and DTLZ test suites are below average, the overall results are decent as indicated by the overall Friedman ranking in Table 7.6. MOAMO is ranked top three overall on all metrics, but there is still room for improvements especially on the DTLZ test suite.

8. Conclusion

This chapter will conclude the thesis. First the answers to the research questions are summarized. Then the contributions that have been made are presented. Lastly, the future work that can be done to improve upon the work in this thesis is covered.

The goal of the thesis was to create a multi-objective metaheuristic algorithm based on a single-objective algorithm from the literature. Following is a summary of how the research questions were answered.

RQ1 *Which single-objective algorithm has the best potential for multi-objective extension?*

Six algorithms were gathered from the structured literature review. Four of these algorithms were implemented and compared against each other in an experiment. Animal Migration Optimization (AMO) was evaluated to have the best potential for multi-objective extension.

RQ2 *Which multi-objective techniques are most suitable for extending the selected algorithm to multi-objective?*

In the structured literature review three multi-objective techniques were analyzed. Two of these techniques, non-dominated sorting and archiving, were used to create three multi-objective versions of AMO. One based on non-dominated sorting, one based on an external archive and one hybrid using both techniques. These three versions where compared in an experiment. A combination of non-dominated sorting and archiving was evaluated to be the most suitable way of extending AMO to multi-objective.

RQ3 *How does the proposed algorithm's performance compare to other competitive algorithms from the literature?*

The performance of Multi-Objective Animal Migration Optimization (MOAMO) was compared with eight other competitive multi-objective algorithms in an experi-

ment. MOAMO showed competitive results in the experiment.

8.1. Contributions

This thesis has three contributions that are presented in the following sections.

8.1.1. Multi-Objective Animal Migration Optimization Algorithm

The animal migration optimization algorithm has been extended to handle multiple objectives using both an external archive and non-dominated sorting. The multi-objective animal migration optimization algorithm is competitive with other state of the art multi-objective algorithms, performing well in an empirical experiment where it was compared with eight other algorithms on 22 test functions measuring results with three performance metrics.

8.1.2. Framework for Single- and Multi-Objective Optimization

A framework has been created that supports both single- and multi-objective optimization. The framework includes 21 commonly used single-objective test functions, and the multi-objective test functions DTLZ1-7, ZDT1-4 and 6, and UF1-10. Also implemented are the multi-objective performance metrics GD, HV, and IGD. The framework contains several single- and multi-objective algorithms. The single-objective algorithms included are AMO, DA, EWA, LOA, PSO, and SA. The multi-objective algorithms included are MOAMO, archive AMO, non-dominated sorting AMO, and MOPSO. Moreover, different solution sampling methods have been implemented to make it easier to study the behavior of the algorithms. The sampled solutions can be printed as statistical data or plotted. The framework is available as an open source project[1].

8.1.3. jMetal Experimental Bug

A bug was discovered in the jMetal [Durillo and Nebro, 2011] framework where some algorithms' state was kept between independent runs during experiments. This caused these algorithms to start runs with the same solutions in the archive as they had at the end of the previous run, giving different results than what they would have gotten if the runs were independent. The developers were notified about this problem and quickly resolved the issue.

[1]Metaheuristic framework: `https://github.com/duvholt/rust-metaheuristic-framework`

8.2. Future Work

Many interesting topics appeared during the development of MOAMO. Some of these were not explored in great detail because of limited time or because it would take the algorithm in another direction than what the goal of this thesis was. In this section, these topics will be discussed.

8.2.1. Extend AMO Using Decomposition

When deciding which techniques to use when extending AMO to multi-objective, non-dominated sorting, archive, and decomposition were considered as extension techniques. While decomposition was not pursued, it is still worth exploring a decomposition variant of AMO. Especially seeing how MOEA/DD is able to produce significantly better Pareto fronts on some test functions like DTLZ1 and UF8.

8.2.2. Many-Objective Optimization

Early on it was decided that this thesis was going to focus on multi-objective optimization with few objectives. On problems with many objectives, new difficulties might be encountered. For example, non-dominated sets usually become a lot bigger, making it even harder to decide which solutions are worth exploring. Lately, there has been more focus on many-objective optimization like the CEC2017 competition [Cheng et al., 2017b]. It could be interesting to see how the algorithm performs on problems with many objectives, and look at some improvements that will make the algorithm perform particularly well on these types of problems. The reference points used by NSGA-III are especially interesting to investigate since MOAMO heavily relies on non-dominated sorting.

8.2.3. Improve the Framework

The main focus of this thesis was always to create a multi-objective algorithm. The framework was built to help us achieve this goal. Because of this some of the implementation decisions worked very well for the scope of this thesis, but might not be the most practical for others that wish to use the framework in the future. Even though several of the multi-objective test functions have a scalable number of objectives they are hard-coded to two or three in the implemented test functions. A potential fix for this is to create a test function generator which takes the number of decision variables and the number of objectives as input and returns a test function based on this input. Another example of an improvement is to define traits for techniques such as mutations, archives, and crossovers. A trait in Rust is similar to interfaces in other programming languages. Then the algorithm

can for example take the mutation operator as input which makes it easier to swap out parts of the algorithm.

Another improvement that can be made is adding support for parallel computing. Parallel computing focuses on splitting a large problem into smaller problems that can be solved independently in parallel. For example the framework runner will only run one algorithm on one test function at the same time, but since these are independent runs they could all run in parallel to save time. Some of the algorithms in the framework could also utilize parallel computing. For example during the animal migration phase and population update phase in MOAMO. In both of these phases a new population is created. None of the calculations done when creating a new solution depends on the other solutions that are created during the same phase in the same generation. Because of this, all of the new solutions created in each phase could be created in parallel. While this does not reduce the computational complexity, it will improve run time.

8.2.4. Testing on Other Test Suites

MOAMO has only been tested on DTLZ, UF, and ZDT. While these are well-known test suites, it is still worthwhile to see MOAMO's performance on other test suites like WFG [Huband et al., 2005] and LSMOP [Cheng et al., 2017a] which are both multi-objective test suites with a scalable number of objectives. Other test suites can present different challenges for the algorithm, and making sure the algorithm performs well on more problems is in line with the vision of making a general algorithm.

8.2.5. Test MOAMO on a Real World Problem

During the development of MOAMO, it was never applied to a real-world problem. While MOAMO has proven that it can find decent solutions on test functions, real-world problems can be harder to solve as they usually involve uncertainties, local solutions, and deceptive global solutions [Mirjalili et al., 2017]. While test functions are excellent during development, the ultimate goal of any optimization algorithm should be to solve real-world problems. Applying MOAMO to a real-world problem would be a great way to further test the quality of the algorithm on a problem that can present different difficulties than the test functions it has been tested on in this thesis.

Appendices

A. Structured Literature Review Process

This appendix explains the review process of the structured literature review used to find the related work in Chapter 3.

A.1. Identification of Research

This section presents the steps that were required to find the related work.

A.1.1. Search Engine

The majority of articles were found using Google Scholar. Google Scholar aggregates most of the prominent digital research literature libraries in one place, making it ideal for this task. Additionally, some articles were identified using IEEE Xplore since the site provides better filtering options than Google Scholar and many of the related articles are hosted there.

A.1.2. Search Strategy

Before starting the search, a set of search terms had to be made. The search terms are based on the search research questions from Section 3.1.1 and are grouped into three categories: algorithm, single or multi-objective, and type of contribution.

A visualization using AND ($\wedge$) between groups and OR ($\vee$) inside groups is shown below:

$$
\begin{aligned}
&(\text{metaheuristic} \vee \text{optimization algorithm} \vee \text{evolutionary algorithm}) \wedge \\
&(\text{multi-objective} \vee \text{single-objective}) \wedge \\
&(\text{new} \vee \text{improved} \vee \text{hybrid})
\end{aligned}
$$

This search resulted in around 400 articles.

A.2. Selection of Primary Studies

The first sizable cut of articles happened here. Because of the large amount of articles, some were cut based on either reading the title, or a quick read of the abstract. The biggest reasons for cutting articles in this step were:

1. Article did not focus on optimization algorithms.
2. Article covered an algorithm that was problem specific.
3. Article focused on improvement but was very old.

After this screening, there were 104 articles left.

A.3. Study Quality Assessment

To asses the quality of the literature several criteria were used. Based on the inclusion and quality criteria it was decided whether the articles would be removed or not.

A.3.1. Inclusion Criteria

For a article to be included in the literature review it must cover one or more of the following criteria shown in Table A.1. The first three inclusion criteria (IC) are the primary inclusion criteria, while the two next are secondary. At the end of the table, the quality criteria (QC) are listed.

Table A.1.: Criteria grouped into primary inclusion criteria, secondary inclusion criteria and quality criteria.

Criteria identification	Criteria
IC 1	Presents a new algorithm
IC 2	The study's main concern is metaheuristics
IC 3	Improves an existing algorithm
IC 4	Converts an algorithm from single-objective to multi-objective
IC 5	The study focuses on multi-objective optimization
QC 1	The study is innovative
QC 2	The algorithm design is justified and reproducible

A.3.2. Abstract Inclusion Criteria Screening

In this step, the abstracts of the remaining 104 articles were read and evaluated based on the inclusion criteria seen in Table A.1. In certain cases where the abstract did not give sufficient information about the article, the introduction and conclusion

of the article were skimmed. After this step, there were 64 articles left.

A.3.3. Full Text Inclusion Criteria Screening

The remaining 64 articles were read, and the majority of them were cut as they did not meet the inclusion criteria. After this screening, there were 18 articles left.

A.3.4. Quality Criteria Questions

The quality criteria were expanded to more specific questions. The following questions are based on the questions provided by Kofod-Petersen [2014].

QC1: Is there a clear statement of the aim of the research?

QC2: Is the study put into context of other studies and research?

QC3: Is the study algorithm reproducible?

QC4: Are system or algorithmic design decisions justified?

QC5: Is the experimental procedure thoroughly explained and reproducible?

QC6: Is it clearly stated in the study which other algorithms the study's algorithm(s) have been compared with?

QC7: Is the study innovative?

QC8: Are the test results thoroughly analyzed?

QC9: Does the test evidence support the findings presented?

QC10: Is the test data set reproducible?

QC11: Are the performance metrics used in the study explained and justified?

The answers to these questions can be found in Table A.3.

A.3.5. Full Text Quality Criteria Screening

The remaining 18 articles were thoroughly read again and evaluated based on the quality criteria questions. The evaluated articles are listed in Table A.2, where each row contains the algorithm's abbreviation, full title, and author name(s). The evaluation is shown in Table A.3 using the previously mentioned abbreviation and

a score of either 0, 0.5, or 1 for each Quality Criteria question.

Since both Dandelion Algorithm and Dragonfly Algorithm use the abbreviation "DA" Dragonfly Algorithm is referred to as DFA and Dandelion Algorithm as DA to avoid confusion.

The cutoff point was set at a score of 10 points which two articles did not meet. With the exception of the articles by Yang et al. [2013] and Rao and Waghmare [2014] all the studies from Table A.2 are part of the core articles. These remaining 16 articles are presented in Section 3.2.

Table A.2.: The final selection of articles for the literature review.

Abbrevation	Title	Author(s)
AMO	Animal migration optimization: an optimization algorithm inspired by animal migration behavior	Li et al. [2014]
DA	A new dandelion algorithm and optimization for extreme learning machine	Gong et al. [2017]
EWA	Earthworm optimization algorithm: a bio-inspired metaheuristic algorithm for global optimization problems	Wang et al. [2015]
LOA	Lion optimization algorithm (LOA): a nature-inspired metaheuristic algorithm	Yazdani and Jolai [2016]
MS	Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems	Wang [2016]
SOS	Symbiotic Organisms Search: A new metaheuristic optimization algorithm	Cheng and Prayogo [2014]
AMOSA	A simulated annealing-based multiobjective optimization algorithm: AMOSA	Bandyopadhyay et al. [2008]
DPP2	A modified dual-population approach for solving multi-objective problems	Bui et al. [2017]
MOALO	Multi-objective ant lion optimizer: a multi-objective optimization algorithm for solving engineering problems	Mirjalili et al. [2017]
(MO)DFA	Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems	Mirjalili [2016]
MOEA/DD	An evolutionary many-objective optimization algorithm based on dominance and decomposition	Li et al. [2015]
MOFPA	Multi-objective Flower Algorithm for Optimization	Yang et al. [2013]
MOGWO	Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization	Mirjalili et al. [2016]
MOWOA	Multi-objective whale optimization	Kumawat et al. [2017]
NSIWO	Multiobjective invasive weed optimization: Application to analysis of Pareto improvement models in electricity markets	Nikoofard et al. [2012]
NS-MFO	Non-dominated sorting moth flame optimization (NS-MFO) for multi-objective problems	Savsani and Tawhid [2017]
SMPSO	SMPSO: A new PSO-based metaheuristic for multi-objective optimization	Nebro et al. [2009a]
TLBO	A comparative study of a teaching–learning-based optimization algorithm on multi-objective unconstrained and constrained functions	Rao and Waghmare [2014]

Table A.3.: Rating of each article using the quality criteria.

Article	QC1	QC2	QC3	QC4	QC5	QC6	QC7	QC8	QC9	QC10	QC11	Score
AMO	1	1	1	1	1	1	0.5	1	1	1	1	10.5
DA	1	1	1	1	1	1	1	1	1	1	0.5	10.5
EWA	1	1	1	1	1	1	1	1	1	1	1	11
LOA	1	1	1	1	1	1	1	1	1	1	1	11
MS	1	1	1	1	1	1	1	1	1	1	1	11
SOS	1	1	1	1	1	1	1	1	1	1	1	11
AMOSA	1	1	1	1	1	1	0.5	1	0.5	1	1	10
DPP2	1	1	1	1	1	1	0.5	1	0.5	1	1	10
MOALO	1	1	1	1	1	1	1	1	0.5	1	1	10.5
(MO)DFA	1	1	1	1	1	1	1	1	0.5	1	1	10.5
MOEA/DD	1	1	1	1	1	1	1	1	1	1	1	11
MOFPA	1	0.5	1	1	0.5	1	0.5	0.5	1	1	1	9
MOGWO	1	1	1	1	1	1	1	1	1	1	1	11
MOWOA	1	1	1	0.5	1	1	0.5	1	1	1	1	10
NSIWO	1	1	1	1	1	1	0.5	1	0.5	1	1	10
NS-MFO	1	1	1	1	1	1	0.5	1	1	1	1	10.5
SMPSO	1	1	1	1	1	1	0.5	1	1	1	1	10.5
TLBO	1	1	0.5	1	1	1	0	1	1	1	1	9.5

B. Multi-Objective Extension Results

The following tables show the results of the multi-objective extension experiment in Chapter 6. There are 6 tables in total, one for each performance metric on each test suite. Each row has three values: mean is the mean value that the algorithm got over 30 runs on the test function. SD is the standard deviation for the 30 runs. Rank is how good the mean value was compared to the other algorithms on the test function.

Table B.1.: Results on DTLZ measured using GD

Test function	Measure	Archive	Hybrid	Non-dominated
DTLZ1	Mean	5.66E+01	**4.66E-01**	5.03E-01
	SD	5.95E+00	3.54E-01	5.13E-01
	Rank	3	1	2
DTLZ2	Mean	2.90E-02	**2.55E-03**	2.60E-03
	SD	2.78E-03	3.26E-04	3.97E-04
	Rank	3	1	2
DTLZ3	Mean	1.11E+02	**2.52E-02**	3.77E-02
	SD	8.91E+00	6.70E-02	5.75E-02
	Rank	3	1	2
DTLZ4	Mean	2.71E-02	**4.92E-03**	5.13E-03
	SD	7.75E-03	5.08E-04	1.29E-03
	Rank	3	1	2
DTLZ5	Mean	4.25E-02	**2.83E-04**	3.18E-04
	SD	5.24E-03	2.22E-05	2.98E-05
	Rank	3	1	2
DTLZ6	Mean	4.70E-04	4.71E-04	**4.69E-04**
	SD	1.53E-05	1.81E-05	1.74E-05
	Rank	2	3	1
DTLZ7	Mean	3.61E-03	2.14E-03	**2.09E-03**
	SD	2.47E-04	1.88E-04	2.25E-04
	Rank	3	2	1

Table B.2.: Results on DTLZ measured using HV

Test function	Measure	Archive	Hybrid	Non-dominated
DTLZ1	Mean	0.00E+00	8.34E-02	**1.57E-01**
	SD	0.00E+00	2.27E-01	2.99E-01
	Rank	3	2	1
DTLZ2	Mean	8.65E-02	**3.76E-01**	3.76E-01
	SD	2.36E-02	3.63E-03	3.87E-03
	Rank	3	1	2
DTLZ3	Mean	0.00E+00	**3.11E-01**	2.38E-01
	SD	0.00E+00	1.42E-01	1.74E-01
	Rank	3	1	2
DTLZ4	Mean	1.77E-01	**3.72E-01**	3.70E-01
	SD	3.39E-02	2.93E-03	4.21E-03
	Rank	3	1	2
DTLZ5	Mean	1.51E-02	**9.37E-02**	9.34E-02
	SD	6.73E-03	1.60E-04	2.30E-04
	Rank	3	1	2
DTLZ6	Mean	**9.55E-02**	9.54E-02	9.53E-02
	SD	6.86E-05	8.51E-05	9.77E-05
	Rank	1	2	3
DTLZ7	Mean	2.64E-01	**2.94E-01**	2.93E-01
	SD	6.12E-03	2.96E-03	2.19E-03
	Rank	3	1	2

Table B.3.: Results on DTLZ measured using IGD

Test function	Measure	Archive	Hybrid	Non-dominated
DTLZ1	Mean	2.70E+00	4.22E-02	**4.01E-02**
	SD	3.39E-01	3.05E-02	3.27E-02
	Rank	3	2	1
DTLZ2	Mean	2.57E-03	**6.39E-04**	6.51E-04
	SD	1.96E-04	2.33E-05	2.12E-05
	Rank	3	1	2
DTLZ3	Mean	5.93E+00	**4.78E-03**	7.34E-03
	SD	6.00E-01	1.04E-02	1.06E-02
	Rank	3	1	2
DTLZ4	Mean	2.38E-03	**1.20E-03**	1.21E-03
	SD	2.79E-04	8.50E-05	9.97E-05
	Rank	3	1	2
DTLZ5	Mean	4.96E-04	**1.30E-05**	1.42E-05
	SD	6.52E-05	8.28E-07	7.49E-07
	Rank	3	1	2
DTLZ6	Mean	2.93E-05	**2.92E-05**	3.05E-05
	SD	1.88E-06	2.60E-06	2.05E-06
	Rank	2	1	3
DTLZ7	Mean	2.21E-03	**1.84E-03**	1.84E-03
	SD	1.28E-04	1.13E-04	8.12E-05
	Rank	3	1	2

Table B.4.: Results on UF measured using GD

Test function	Measure	Archive	Hybrid	Non-dominated
UF1	Mean	2.86E-03	7.28E-04	**7.25E-04**
	SD	1.27E-03	3.30E-04	2.51E-04
	Rank	3	2	1
UF2	Mean	8.35E-03	**8.76E-04**	9.52E-04
	SD	2.58E-03	2.92E-04	2.02E-04
	Rank	3	1	2
UF3	Mean	8.83E-03	**2.79E-03**	2.97E-03
	SD	1.46E-02	1.07E-03	1.15E-03
	Rank	3	1	2
UF4	Mean	7.99E-03	**4.21E-03**	4.39E-03
	SD	3.54E-04	3.42E-05	3.27E-05
	Rank	3	1	2
UF5	Mean	6.08E-02	**2.65E-02**	3.20E-02
	SD	2.64E-02	9.10E-03	2.41E-02
	Rank	3	1	2
UF6	Mean	5.49E-02	4.69E-03	**2.50E-03**
	SD	1.21E-02	1.23E-02	4.21E-03
	Rank	3	2	1
UF7	Mean	2.36E-03	5.43E-04	**5.27E-04**
	SD	1.22E-03	1.57E-04	2.24E-04
	Rank	3	2	1
UF8	Mean	**1.19E-02**	2.07E-01	2.00E-01
	SD	3.25E-03	7.95E-02	6.64E-02
	Rank	1	3	2
UF9	Mean	3.57E-02	4.68E-02	**3.38E-02**
	SD	3.42E-02	2.55E-02	2.95E-02
	Rank	2	3	1
UF10	Mean	3.69E-01	2.61E-01	**2.12E-01**
	SD	3.74E-02	4.11E-01	3.48E-01
	Rank	3	2	1

Table B.5.: Results on UF measured using HV

Test function	Measure	Archive	Hybrid	Non-dominated
UF1	Mean	5.47E-01	5.98E-01	**5.99E-01**
	SD	2.69E-02	2.20E-02	1.84E-02
	Rank	3	2	1
UF2	Mean	5.76E-01	**6.50E-01**	6.48E-01
	SD	5.00E-03	9.04E-04	1.15E-03
	Rank	3	1	2
UF3	Mean	3.53E-01	**4.89E-01**	4.79E-01
	SD	1.02E-02	3.67E-02	4.92E-02
	Rank	3	1	2
UF4	Mean	2.19E-01	**2.73E-01**	2.71E-01
	SD	1.92E-03	6.42E-04	4.87E-04
	Rank	3	1	2
UF5	Mean	6.08E-02	**2.06E-01**	2.03E-01
	SD	4.57E-02	3.80E-02	3.51E-02
	Rank	3	1	2
UF6	Mean	6.48E-02	**2.54E-01**	2.34E-01
	SD	2.75E-02	4.50E-02	3.37E-02
	Rank	3	1	2
UF7	Mean	4.13E-01	**4.67E-01**	4.66E-01
	SD	2.57E-02	3.39E-03	3.36E-03
	Rank	3	1	2
UF8	Mean	**1.58E-01**	1.46E-01	1.48E-01
	SD	4.13E-03	5.16E-02	3.65E-02
	Rank	1	3	2
UF9	Mean	1.84E-01	5.01E-01	**5.30E-01**
	SD	2.03E-02	8.89E-02	8.83E-02
	Rank	3	2	1
UF10	Mean	0.00E+00	**3.33E-02**	2.52E-02
	SD	0.00E+00	3.86E-02	3.69E-02
	Rank	3	1	2

Table B.6.: Results on UF measured using IGD

Test function	Measure	Archive	Hybrid	Non-dominated
UF1	Mean	2.89E-03	2.00E-03	**1.98E-03**
	SD	6.61E-04	6.17E-04	5.38E-04
	Rank	3	2	1
UF2	Mean	2.89E-03	**4.24E-04**	4.95E-04
	SD	2.27E-04	5.88E-05	7.12E-05
	Rank	3	1	2
UF3	Mean	1.09E-02	**5.05E-03**	5.38E-03
	SD	9.29E-04	1.75E-03	2.10E-03
	Rank	3	1	2
UF4	Mean	2.59E-03	**1.28E-03**	1.34E-03
	SD	7.94E-05	1.67E-05	1.48E-05
	Rank	3	1	2
UF5	Mean	6.62E-02	**4.14E-02**	4.17E-02
	SD	7.14E-03	3.40E-03	2.98E-03
	Rank	3	1	2
UF6	Mean	7.71E-03	7.76E-03	**6.43E-03**
	SD	9.96E-04	3.90E-03	3.27E-03
	Rank	2	3	1
UF7	Mean	2.40E-03	**1.15E-03**	1.19E-03
	SD	7.21E-04	1.50E-04	1.39E-04
	Rank	3	1	2
UF8	Mean	3.09E-03	**2.78E-03**	2.85E-03
	SD	1.48E-04	5.30E-04	4.36E-04
	Rank	3	1	2
UF9	Mean	4.56E-03	2.30E-03	**1.99E-03**
	SD	2.15E-04	9.13E-04	9.99E-04
	Rank	3	2	1
UF10	Mean	2.12E-02	4.96E-03	**4.36E-03**
	SD	1.04E-03	1.17E-03	1.28E-03
	Rank	3	2	1

C. Multi-Objective Comparison Results

The following tables show the results of the multi-objective comparison experiment in Chapter 7. There are 9 tables in total, one for each performance metric on each test suite. Each row has three values: mean is the mean value that the algorithm got over 30 runs on the test function. SD is the standard deviation for the 30 runs. Rank is how good the mean value was compared to the other algorithms on the test function.

Table C.1.: Results on ZDT measured using GD

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
ZDT1	Mean	1.44E-04	1.47E-04	4.51E-05	1.82E-04	**4.24E-05**	1.15E-04	1.27E-04	2.00E-04	1.52E-04
	SD	9.42E-06	6.83E-06	1.55E-06	2.90E-05	1.08E-06	6.22E-05	1.99E-05	2.72E-05	2.26E-05
	Rank	5	6	2	8	1	3	4	9	7
ZDT2	Mean	4.67E-05	4.55E-05	4.45E-05	1.30E-04	**4.45E-05**	1.41E-04	4.71E-05	1.37E-04	8.44E-05
	SD	2.28E-06	2.10E-06	2.97E-08	3.08E-05	1.01E-07	9.74E-05	2.55E-06	6.17E-05	1.16E-05
	Rank	4	3	2	7	1	9	5	8	6
ZDT3	Mean	9.80E-05	9.58E-05	3.71E-03	1.18E-04	9.72E-05	1.45E-03	**9.48E-05**	1.29E-04	1.05E-04
	SD	6.26E-06	4.90E-06	9.38E-04	1.41E-05	8.14E-06	2.40E-03	5.38E-06	2.77E-05	8.50E-06
	Rank	4	2	9	6	3	8	1	7	5
ZDT4	Mean	1.89E-04	1.19E-04	1.36E-04	1.31E-04	1.04E-04	1.03E-01	**6.92E-05**	2.64E-02	1.06E-03
	SD	5.48E-05	3.80E-05	4.23E-05	3.02E-05	2.46E-05	1.29E-01	8.06E-06	7.30E-02	3.16E-03
	Rank	6	3	5	4	2	9	1	8	7
ZDT6	Mean	7.30E-05	7.34E-05	7.39E-05	7.53E-05	**7.17E-05**	1.41E-02	6.78E-03	6.53E-03	1.23E-03
	SD	2.06E-06	3.26E-06	2.33E-06	2.76E-06	2.59E-06	1.54E-02	1.90E-02	9.07E-03	5.03E-03
	Rank	2	3	4	5	1	9	8	7	6

Table C.2.: Results on ZDT measured using HV

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
ZDT1	Mean	**6.62E-01**	6.62E-01	6.62E-01	6.60E-01	6.62E-01	5.45E-01	6.62E-01	6.60E-01	6.60E-01
	SD	1.56E-05	9.92E-06	9.80E-06	2.67E-04	7.15E-06	9.11E-02	1.39E-05	2.21E-04	2.51E-04
	Rank	1	3	4	8	5	9	2	6	7
ZDT2	Mean	3.29E-01	**3.29E-01**	3.28E-01	3.27E-01	3.28E-01	2.25E-01	3.29E-01	3.28E-01	3.27E-01
	SD	1.76E-05	1.33E-05	1.11E-08	1.95E-04	1.99E-08	9.66E-02	1.89E-05	2.31E-04	3.01E-04
	Rank	2	1	4	7	5	9	3	6	8
ZDT3	Mean	5.16E-01	**5.16E-01**	5.14E-01	5.15E-01	5.15E-01	4.35E-01	5.16E-01	5.14E-01	5.16E-01
	SD	9.35E-06	5.14E-06	5.56E-05	8.82E-05	7.10E-05	6.02E-02	7.63E-06	5.86E-04	6.48E-05
	Rank	3	1	7	5	6	9	2	8	4
ZDT4	Mean	6.60E-01	6.61E-01	6.60E-01	6.60E-01	6.60E-01	5.69E-01	**6.62E-01**	6.61E-01	6.48E-01
	SD	8.33E-04	6.34E-04	6.07E-04	4.97E-04	3.49E-04	5.73E-02	1.81E-05	4.96E-04	4.24E-02
	Rank	6	2	5	7	4	9	1	3	8
ZDT6	Mean	4.01E-01	4.01E-01	4.01E-01	3.98E-01	4.00E-01	2.88E-01	**4.01E-01**	4.00E-01	4.00E-01
	SD	3.91E-05	5.22E-05	4.42E-05	3.92E-04	4.72E-05	1.21E-01	3.77E-05	1.56E-04	7.01E-04
	Rank	2	3	4	8	5	9	1	7	6

Table C.3.: Results on ZDT measured using IGD

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
ZDT1	Mean	1.35E-04	1.35E-04	1.65E-04	1.84E-04	1.64E-04	5.91E-03	**1.34E-04**	1.72E-04	1.78E-04
	SD	7.10E-07	7.25E-07	1.43E-06	8.39E-06	1.14E-07	2.83E-03	6.99E-07	6.50E-06	1.41E-05
	Rank	2	3	5	8	4	9	1	6	7
ZDT2	Mean	1.39E-04	1.39E-04	1.41E-04	1.87E-04	1.41E-04	8.01E-03	**1.39E-04**	1.71E-04	1.77E-04
	SD	1.63E-06	1.09E-06	4.78E-10	1.02E-05	1.72E-09	5.40E-03	1.46E-06	7.24E-06	1.04E-05
	Rank	3	2	4	8	5	9	1	6	7
ZDT3	Mean	1.00E-04	9.89E-05	2.60E-04	1.31E-04	1.35E-04	6.59E-03	**9.88E-05**	1.34E-04	1.28E-04
	SD	1.05E-06	9.18E-07	3.58E-06	5.43E-06	3.18E-06	3.23E-03	9.41E-07	8.39E-06	7.96E-06
	Rank	3	2	8	5	7	9	1	6	4
ZDT4	Mean	1.49E-04	1.40E-04	1.80E-04	1.80E-04	1.75E-04	6.56E-03	**1.35E-04**	1.70E-04	4.45E-04
	SD	7.33E-06	3.90E-06	2.37E-06	5.22E-06	1.31E-06	3.37E-03	7.65E-07	6.05E-06	9.72E-04
	Rank	3	2	7	6	5	9	1	4	8
ZDT6	Mean	1.34E-04	1.33E-04	1.41E-04	2.28E-04	1.38E-04	8.93E-03	**1.33E-04**	1.66E-04	1.77E-04
	SD	5.44E-07	3.50E-07	1.02E-07	1.32E-05	1.24E-07	9.11E-03	9.05E-07	6.51E-06	1.52E-05
	Rank	3	2	5	8	4	9	1	6	7

Table C.4.: Results on DTLZ measured using GD

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
DTLZ1	Mean	9.98E-04	3.07E-01	**7.21E-04**	3.24E-02	2.96E-03	4.77E-02	6.31E-01	1.36E-01	5.78E-01
	SD	5.46E-04	7.63E-01	1.57E-04	9.08E-02	1.03E-02	2.06E-01	1.13E+00	3.12E-01	4.20E-01
	Rank	2	7	1	4	3	5	9	6	8
DTLZ2	Mean	5.26E-04	1.07E-03	**5.04E-04**	1.04E-03	5.96E-04	5.30E-04	2.50E-03	2.00E-03	2.43E-03
	SD	1.99E-05	1.17E-04	1.26E-06	7.92E-05	2.82E-05	6.80E-05	2.55E-04	1.71E-04	2.04E-04
	Rank	2	6	1	5	4	3	9	7	8
DTLZ3	Mean	4.13E-03	8.96E-03	**1.10E-03**	1.57E-02	1.68E-02	4.68E-02	3.20E+00	4.25E-02	2.36E-02
	SD	1.48E-02	1.90E-02	1.75E-04	3.08E-02	8.43E-02	1.65E-01	5.81E+00	7.76E-02	4.30E-02
	Rank	2	3	1	4	5	8	9	7	6
DTLZ4	Mean	3.87E-03	3.87E-03	4.21E-03	3.98E-03	4.09E-03	**1.47E-04**	4.49E-03	4.07E-03	5.10E-03
	SD	1.29E-04	1.06E-03	1.97E-05	1.71E-04	3.64E-04	5.90E-04	2.20E-04	3.82E-04	8.42E-04
	Rank	3	2	7	4	6	1	8	5	9
DTLZ5	Mean	2.06E-04	2.31E-04	1.70E-01	2.64E-04	3.10E-04	2.87E-04	**2.03E-04**	3.23E-04	2.96E-04
	SD	8.97E-06	3.50E-05	4.29E-03	2.58E-05	6.11E-05	1.88E-04	1.08E-05	2.59E-05	2.62E-05
	Rank	2	3	9	4	7	5	1	8	6
DTLZ6	Mean	2.89E-02	5.34E-02	6.56E-01	2.74E-02	3.76E-02	2.99E-03	**4.66E-04**	3.42E-02	4.71E-04
	SD	4.05E-03	1.06E-02	1.92E-02	3.52E-03	7.96E-03	3.25E-03	8.99E-06	8.24E-03	1.60E-05
	Rank	5	8	9	4	7	3	1	6	2
DTLZ7	Mean	1.32E-03	2.33E-03	2.17E-01	1.95E-03	**1.31E-03**	2.36E-03	3.05E-03	2.33E-03	2.14E-03
	SD	4.03E-04	2.06E-04	9.58E-03	2.33E-04	1.69E-04	1.48E-03	5.76E-04	2.28E-04	1.85E-04
	Rank	2	6	9	3	1	7	8	5	4

Table C.5.: Results on DTLZ measured using HV

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
DTLZ1	Mean	7.59E-01	4.98E-01	**7.92E-01**	7.29E-01	7.86E-01	4.60E-01	6.93E-01	6.16E-01	2.79E-02
	SD	1.38E-02	3.33E-01	2.54E-03	1.39E-01	6.63E-03	7.86E-02	1.05E-01	3.00E-01	1.18E-01
	Rank	3	7	1	4	2	8	5	6	9
DTLZ2	Mean	4.00E-01	3.95E-01	4.23E-01	3.91E-01	**4.25E-01**	1.81E-01	3.75E-01	3.89E-01	3.76E-01
	SD	4.07E-03	3.89E-03	9.72E-05	4.84E-03	1.20E-03	4.90E-02	5.55E-03	3.00E-03	4.11E-03
	Rank	3	4	2	5	1	9	8	6	7
DTLZ3	Mean	3.65E-01	3.48E-01	**4.10E-01**	3.76E-01	4.05E-01	1.57E-01	2.70E-01	3.78E-01	2.84E-01
	SD	6.98E-02	3.15E-02	5.79E-03	9.24E-03	9.06E-03	6.26E-02	1.28E-01	7.27E-02	1.53E-01
	Rank	5	6	1	4	2	9	8	3	7
DTLZ4	Mean	4.02E-01	3.66E-01	**4.16E-01**	3.88E-01	3.54E-01	1.37E-02	3.79E-01	3.75E-01	3.73E-01
	SD	3.92E-03	9.95E-02	8.16E-05	3.69E-03	1.02E-01	5.22E-02	2.80E-03	4.56E-02	3.47E-03
	Rank	2	7	1	3	8	9	4	5	6
DTLZ5	Mean	**9.49E-02**	9.48E-02	7.65E-02	9.42E-02	9.02E-02	6.96E-02	9.48E-02	9.36E-02	9.36E-02
	SD	1.13E-05	1.06E-05	3.85E-04	1.03E-04	6.89E-04	2.56E-02	1.50E-05	2.03E-04	1.62E-04
	Rank	1	3	8	4	7	9	2	6	5
DTLZ6	Mean	4.51E-03	8.24E-05	2.13E-03	7.43E-03	2.39E-03	5.63E-02	**9.57E-02**	6.96E-03	9.54E-02
	SD	4.55E-03	2.79E-04	3.38E-03	6.94E-03	2.15E-03	3.10E-02	1.40E-05	7.18E-03	9.17E-05
	Rank	6	9	8	4	7	3	1	5	2
DTLZ7	Mean	2.62E-01	2.96E-01	2.73E-01	2.96E-01	**3.05E-01**	1.55E-01	2.90E-01	2.87E-01	2.94E-01
	SD	3.06E-02	1.97E-03	1.88E-03	2.22E-03	7.56E-03	5.39E-02	3.42E-03	2.93E-03	3.04E-03
	Rank	8	3	7	2	1	9	5	6	4

Table C.6.: Results on DTLZ measured using IGD

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
DTLZ1	Mean	6.18E-04	4.11E-03	**3.38E-04**	9.34E-04	3.85E-04	3.46E-03	1.09E-03	2.78E-03	5.06E-02
	SD	1.18E-04	7.41E-03	5.97E-06	2.01E-03	5.69E-05	7.86E-04	9.62E-04	5.61E-03	3.23E-02
	Rank	3	8	1	4	2	7	5	6	9
DTLZ2	Mean	6.66E-04	6.26E-04	**4.50E-04**	6.31E-04	4.66E-04	4.18E-03	6.44E-04	5.44E-04	6.37E-04
	SD	4.54E-05	2.42E-05	3.14E-07	2.96E-05	5.24E-06	9.83E-04	3.06E-05	1.55E-05	2.10E-05
	Rank	8	4	1	5	2	9	7	3	6
DTLZ3	Mean	1.63E-03	1.25E-03	**7.50E-04**	1.04E-03	8.03E-04	8.16E-03	3.49E-03	1.37E-03	4.59E-03
	SD	2.71E-03	4.16E-04	1.16E-05	5.23E-05	7.75E-05	2.44E-03	2.94E-03	2.88E-03	7.10E-03
	Rank	6	4	1	3	2	9	7	5	8
DTLZ4	Mean	9.70E-04	1.59E-03	**5.67E-04**	1.07E-03	2.27E-03	9.12E-03	1.12E-03	1.24E-03	1.20E-03
	SD	7.72E-05	2.12E-03	4.28E-07	9.14E-05	2.79E-03	1.02E-03	8.39E-05	1.72E-03	9.00E-05
	Rank	2	7	1	3	8	9	4	6	5
DTLZ5	Mean	9.40E-06	9.88E-06	1.11E-04	1.29E-05	4.63E-05	7.05E-04	**9.29E-06**	1.27E-05	1.32E-05
	SD	1.86E-07	2.58E-07	8.43E-07	6.34E-07	9.94E-06	4.24E-04	2.11E-07	5.54E-07	7.20E-07
	Rank	2	3	8	5	7	9	1	4	6
DTLZ6	Mean	1.84E-03	2.96E-03	2.01E-03	1.67E-03	1.88E-03	2.49E-03	**2.28E-05**	1.75E-03	2.96E-05
	SD	2.63E-04	6.02E-04	3.91E-04	2.87E-04	3.01E-04	1.11E-03	5.02E-07	3.59E-04	2.50E-06
	Rank	5	9	7	3	6	8	1	4	2
DTLZ7	Mean	1.44E-02	1.86E-03	2.91E-03	1.78E-03	2.03E-03	2.13E-02	2.15E-03	**1.57E-03**	1.84E-03
	SD	7.73E-03	7.90E-05	4.89E-05	1.07E-04	2.60E-03	2.99E-03	2.11E-04	4.60E-05	1.01E-04
	Rank	8	4	7	2	5	9	6	1	3

Table C.7.: Results on UF measured using GD

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
UF1	Mean	8.90E-04	1.37E-03	7.76E-03	5.70E-04	6.71E-04	1.20E-03	1.00E-02	**3.65E-04**	7.13E-04
	SD	1.25E-03	2.54E-03	1.86E-03	3.83E-04	9.10E-04	1.27E-03	1.06E-02	2.50E-04	3.31E-04
	Rank	5	7	8	2	3	6	9	1	4
UF2	Mean	1.04E-03	1.75E-03	1.89E-03	1.42E-03	1.31E-03	2.98E-03	3.63E-03	1.70E-03	**8.27E-04**
	SD	3.73E-04	2.99E-04	1.08E-03	3.08E-04	5.82E-04	1.83E-03	1.19E-03	5.35E-04	7.32E-05
	Rank	2	6	7	4	3	8	9	5	1
UF3	Mean	3.29E-03	2.51E-03	1.91E-03	2.09E-03	2.26E-03	1.00E-02	1.27E-02	**1.73E-03**	2.82E-03
	SD	2.41E-03	1.30E-03	1.90E-03	1.27E-03	1.70E-03	6.62E-03	3.90E-03	1.34E-03	1.14E-03
	Rank	7	5	2	3	4	8	9	1	6
UF4	Mean	5.11E-03	5.65E-03	**4.19E-03**	4.80E-03	4.40E-03	1.25E-02	5.65E-03	5.11E-03	4.21E-03
	SD	2.36E-04	3.67E-04	1.03E-04	5.22E-05	1.43E-04	5.38E-03	3.83E-04	6.33E-05	4.45E-05
	Rank	5	8	1	4	3	9	7	6	2
UF5	Mean	7.79E-02	2.94E-02	2.01E-02	2.61E-02	2.87E-02	1.70E-02	2.52E-01	**1.44E-02**	2.67E-02
	SD	3.97E-02	1.56E-02	1.36E-02	1.30E-02	1.44E-02	1.86E-02	1.53E-01	4.38E-03	1.00E-02
	Rank	8	7	3	4	6	2	9	1	5
UF6	Mean	2.96E-02	7.66E-03	1.03E-02	1.25E-02	6.97E-03	2.97E-02	7.03E-02	**6.45E-03**	7.80E-03
	SD	4.55E-02	7.77E-03	1.18E-02	2.21E-02	1.24E-02	4.06E-02	2.68E-02	7.35E-03	2.93E-02
	Rank	7	3	5	6	2	8	9	1	4
UF7	Mean	**5.29E-04**	8.44E-04	5.17E-03	9.59E-04	6.48E-04	9.00E-04	2.69E-03	1.23E-03	5.48E-04
	SD	8.02E-04	6.44E-04	1.58E-03	1.74E-03	8.23E-04	1.29E-03	9.64E-04	2.77E-03	2.15E-04
	Rank	1	4	9	6	3	5	8	7	2
UF8	Mean	5.96E-02	2.23E-01	7.87E-02	2.24E-01	**7.40E-04**	2.97E-03	1.73E-01	2.31E-01	1.66E-01
	SD	4.93E-02	6.22E-02	4.64E-02	5.99E-02	1.98E-04	1.41E-03	6.30E-02	2.14E-02	1.07E-01
	Rank	3	7	4	8	1	2	6	9	5
UF9	Mean	1.64E-01	3.33E-01	7.32E-02	1.63E-01	1.27E-02	**9.60E-03**	3.70E-01	5.22E-02	5.85E-02
	SD	6.46E-02	7.26E-02	2.20E-02	8.85E-02	4.35E-03	7.04E-03	1.40E-01	3.52E-02	4.24E-02
	Rank	7	8	5	6	2	1	9	3	4
UF10	Mean	9.77E-02	2.88E-01	2.94E-01	2.58E-01	1.67E-02	**8.91E-03**	6.43E-01	3.99E-01	2.21E-01
	SD	1.09E-01	8.63E-02	1.30E-01	2.26E-01	1.85E-02	5.24E-03	2.17E-01	5.08E-01	3.53E-01
	Rank	3	6	7	5	2	1	9	8	4

Table C.8.: Results on UF measured using HV

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
UF1	Mean	5.72E-01	5.53E-01	5.59E-01	5.52E-01	5.41E-01	3.95E-01	5.55E-01	5.42E-01	**6.05E-01**
	SD	3.37E-02	3.07E-02	2.65E-02	2.26E-02	2.68E-02	8.71E-02	1.34E-02	2.17E-02	1.87E-02
	Rank	2	5	3	6	8	9	4	7	1
UF2	Mean	6.37E-01	6.24E-01	6.27E-01	6.29E-01	6.26E-01	4.58E-01	6.35E-01	6.24E-01	**6.49E-01**
	SD	7.11E-03	6.77E-03	1.50E-02	7.38E-03	8.64E-03	7.83E-02	2.39E-03	5.20E-03	8.07E-04
	Rank	2	8	5	4	6	9	3	7	1
UF3	Mean	4.27E-01	4.09E-01	3.55E-01	4.18E-01	4.00E-01	2.67E-01	**4.94E-01**	4.25E-01	4.91E-01
	SD	6.86E-02	5.63E-02	3.53E-02	4.86E-02	5.66E-02	5.79E-02	4.73E-02	4.11E-02	4.09E-02
	Rank	3	6	8	5	7	9	1	4	2
UF4	Mean	2.63E-01	2.56E-01	2.72E-01	2.65E-01	2.70E-01	1.24E-01	2.56E-01	2.62E-01	**2.73E-01**
	SD	3.07E-03	4.67E-03	1.24E-03	6.35E-04	1.79E-03	5.26E-02	4.60E-03	6.69E-04	7.33E-04
	Rank	5	8	2	4	3	9	7	6	1
UF5	Mean	1.66E-01	1.56E-01	1.44E-01	1.89E-01	1.70E-01	1.09E-01	2.94E-03	1.94E-01	**2.08E-01**
	SD	8.00E-02	7.88E-02	9.39E-02	6.71E-02	6.08E-02	6.68E-02	1.54E-02	6.47E-02	4.31E-02
	Rank	5	6	7	3	4	8	9	2	1
UF6	Mean	2.03E-01	1.95E-01	1.94E-01	2.19E-01	2.18E-01	1.13E-01	4.07E-02	**2.60E-01**	2.46E-01
	SD	7.71E-02	7.76E-02	6.72E-02	6.46E-02	7.56E-02	1.00E-01	4.60E-02	6.11E-02	4.28E-02
	Rank	5	6	7	3	4	8	9	1	2
UF7	Mean	3.04E-01	3.31E-01	3.50E-01	3.70E-01	3.30E-01	1.59E-01	4.58E-01	3.42E-01	**4.61E-01**
	SD	8.77E-02	1.13E-01	9.09E-02	9.62E-02	9.78E-02	9.27E-02	5.48E-03	1.00E-01	3.30E-02
	Rank	8	6	4	3	7	9	2	5	1
UF8	Mean	2.31E-01	5.87E-02	**2.56E-01**	1.16E-01	2.01E-01	7.89E-02	1.44E-01	1.37E-01	1.74E-01
	SD	6.23E-02	5.21E-02	4.09E-02	3.90E-02	1.92E-03	4.76E-02	3.33E-02	1.66E-02	6.61E-02
	Rank	2	9	1	7	3	8	5	6	4
UF9	Mean	3.81E-01	2.34E-01	**5.39E-01**	3.74E-01	4.98E-01	2.09E-01	2.08E-01	5.20E-01	4.85E-01
	SD	9.78E-02	7.69E-02	5.81E-02	9.94E-02	5.06E-02	6.54E-02	5.91E-02	5.59E-02	1.03E-01
	Rank	5	7	1	6	3	8	9	2	4
UF10	Mean	5.87E-02	1.01E-02	9.50E-02	2.60E-02	**9.63E-02**	5.94E-02	5.86E-02	1.96E-02	2.81E-02
	SD	3.57E-02	1.51E-02	8.72E-02	2.62E-02	7.84E-02	2.19E-02	3.79E-02	2.93E-02	3.09E-02
	Rank	4	9	2	7	1	3	5	8	6

Table C.9.: Results on UF measured using IGD

Test function	Measure	AbYSS	MOCell	MOEA/DD	NSGA-II	NSGA-III	PAES	SMPSO	SPEA2	MOAMO
UF1	Mean	3.35E-03	3.55E-03	3.25E-03	3.66E-03	4.03E-03	9.11E-03	2.15E-03	3.81E-03	**1.84E-03**
	SD	1.50E-03	9.37E-04	9.32E-04	8.62E-04	1.19E-03	2.94E-03	2.59E-04	9.90E-04	5.78E-04
	Rank	4	5	3	6	8	9	2	7	1
UF2	Mean	1.91E-03	1.83E-03	2.92E-03	1.83E-03	2.19E-03	7.11E-03	9.76E-04	1.57E-03	**4.60E-04**
	SD	9.50E-04	9.08E-04	1.87E-03	1.04E-03	1.12E-03	2.40E-03	9.09E-05	4.69E-04	4.77E-05
	Rank	6	4	8	5	7	9	2	3	1
UF3	Mean	8.96E-03	8.50E-03	1.15E-02	8.75E-03	9.36E-03	1.23E-02	**3.92E-03**	8.56E-03	5.10E-03
	SD	3.07E-03	1.65E-03	1.05E-03	1.67E-03	1.97E-03	1.68E-03	1.19E-03	1.70E-03	1.96E-03
	Rank	6	3	8	5	7	9	1	4	2
UF4	Mean	1.49E-03	1.76E-03	1.30E-03	1.45E-03	1.38E-03	9.35E-03	1.65E-03	1.54E-03	**1.27E-03**
	SD	7.49E-05	1.42E-04	1.85E-05	1.80E-05	5.04E-05	3.48E-03	9.30E-05	1.79E-05	1.66E-05
	Rank	5	8	2	4	3	9	7	6	1
UF5	Mean	6.78E-02	7.44E-02	8.90E-02	7.12E-02	6.98E-02	1.20E-01	2.66E-01	5.70E-02	**4.18E-02**
	SD	1.92E-02	2.87E-02	3.96E-02	2.66E-02	2.27E-02	2.45E-02	1.18E-01	1.24E-02	4.01E-03
	Rank	3	6	7	5	4	8	9	2	1
UF6	Mean	1.50E-02	1.60E-02	1.60E-02	1.16E-02	1.46E-02	2.07E-02	1.40E-02	1.06E-02	**6.38E-03**
	SD	4.23E-03	7.02E-03	6.61E-03	4.97E-03	7.07E-03	5.74E-03	3.30E-03	3.71E-03	3.36E-03
	Rank	6	8	7	3	5	9	4	2	1
UF7	Mean	1.07E-02	8.82E-03	8.18E-03	6.61E-03	8.90E-03	1.75E-02	**1.02E-03**	8.05E-03	1.54E-03
	SD	5.43E-03	6.77E-03	5.82E-03	6.10E-03	6.23E-03	5.37E-03	1.68E-04	6.50E-03	2.14E-03
	Rank	8	6	5	3	7	9	1	4	2
UF8	Mean	2.12E-03	3.41E-03	**1.60E-03**	2.98E-03	5.38E-03	5.75E-03	2.89E-03	2.89E-03	2.47E-03
	SD	5.48E-04	6.96E-04	7.43E-04	4.35E-04	2.27E-04	1.07E-03	2.95E-04	2.24E-04	7.28E-04
	Rank	2	7	1	6	8	9	4	5	3
UF9	Mean	3.00E-03	3.88E-03	**1.53E-03**	2.65E-03	2.49E-03	4.66E-03	4.15E-03	1.68E-03	2.42E-03
	SD	6.94E-04	7.44E-04	5.29E-04	8.71E-04	8.33E-04	6.76E-04	5.28E-04	6.30E-04	9.75E-04
	Rank	6	7	1	5	4	9	8	2	3
UF10	Mean	4.80E-03	6.28E-03	**3.40E-03**	4.47E-03	4.14E-03	5.50E-03	3.65E-03	4.45E-03	4.64E-03
	SD	8.78E-04	1.18E-03	1.60E-03	9.87E-04	8.42E-04	5.92E-04	4.13E-04	1.51E-03	9.06E-04
	Rank	7	9	1	5	3	8	2	4	6

D. Single-Objective Test Functions

The single-objective test functions used in this thesis are listed here together with their optimal input.

High Conditioned Elliptic

$$\sum_{i=1}^{n}(10^6)^{\frac{i-1}{n-1}}x_i^2$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Bent Cigar

$$x_1^2 + 10^6 \sum_{i=2}^{n} x_i^2$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Discus

$$10^6 x_1^2 + \sum_{i=2}^{n} x_i^2$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Rosenbrock

$$\sum_{i=1}^{n-1}[100(x_i^2 - x_{i+1})^2 + (x_i - 1)^2]$$

Optimal: $x = [1, ..., 1]$, $f(x) = 0$

Ackley

$$20e^{-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}} - e^{\frac{1}{n}\sum_{i=1}^{n}\cos(2\pi x_i)} + 20 + e$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Weierstrass

$$a = 0.5, b = 3, kmax = 20$$

$$\sum_{i=1}^{n}\left[\sum_{k=0}^{kmax} a^k \cos\left(2\pi b^k (x_i + 0.5)\right)\right] - n\sum_{k=0}^{kmax} a^k \cos(\pi b^k)$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Griewank

$$\frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n}\cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Rastrigin

$$10n + \sum_{i=1}^{n}\left[x_i^2 - 10\cos(2\pi x_i)\right]$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Modified Schwefel

$$418.9828872724338n - \sum_{i=1}^{n} g(z_i)$$

$$z_i = x_i + 4.209687462275036 \cdot 10^2$$

$$g(z_i) = \begin{cases} z_i \sin(|z_i|^{1/2}) & \text{if } |z_i| \leq 500 \\ (500 - z_i \bmod 500)\sin(\sqrt{|500 - z_i \bmod 500|}) - \frac{(z_i - 500)^2}{10000n} & \text{if } z_i > 500 \\ (|z_i| \bmod 500 - 500)\sin(\sqrt{||z_i| \bmod 500 - 500|}) - \frac{(z_i - 500)^2}{10000n} & \text{if } z_i < -500 \end{cases}$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Katsuura

$$\frac{10}{n^2}\prod_{i=1}^{n}\left[1+i\sum_{k=1}^{32}\frac{|2^k x_i-\lfloor 2^k x_i\rceil|}{2^k}\right]^{\frac{10}{n^{1.2}}}-\frac{10}{n^2}$$

Optimal: $x=[0,...,0],\ f(x)=0$

HappyCat

$$\left|\sum_{i=1}^{n}{x_i}^2-n\right|^{1/4}+(0.5\sum_{i=1}^{n}{x_i}^2+\sum_{i=1}^{n}x_i)/n+0.5$$

Optimal: $x=[0,...,0],\ f(x)=0$

HGBat

$$\left|(\sum_{i=1}^{n}{x_i}^2)^2-(\sum_{i=1}^{n}x_i)^2\right|^{1/2}+(0.5\sum_{i=1}^{n}{x_i}^2+\sum_{i=1}^{n}x_i)/n+0.5$$

Optimal: $x=[0,...,0],\ f(x)=0$

Expanded Griewank's plus Rosenbrock

$r(x)=rosenbrock(x), g(x)=griewank(x)$
$g(r(x_1,x_2))+g(r(x_2,x_3))+...+g(r(x_{n-1},x_n))+g(r(x_n,x_1))$

Optimal: $x=[1,...,1],\ f(x)=0$

Expanded Scaffer's F6

Schaffer F6: $s(x)=0.5+\dfrac{\sin^2(\sqrt{x^2+y^2})-0.5}{(1+0.001(x^2+y^2))^2}$
$s(x_1,x_2)+s(x_2,x_3)+...+s(x_{n-1},x_n)+s(x_n,x_1)$

Optimal: $x=[0,...,0],\ f(x)=0$

Axis Parallel Hyper-Ellipsoid

$$\sum_{i=1}^{n} i x_i^2$$

Optimal: $x = [0, ..., 0]$, $f(x) = 0$

Moved-Axis Parallel Hyper-Ellipsoid

$$\sum_{i=1}^{n} \left[i \cdot (x_i - 5 \cdot i) \right]^2$$

Optimal: $x = [5, 10, ..., 5 * i]$, $f(x) = 0$

Modified Easom

Modified to handle n-decision variables.

$$-\prod_{i=1}^{n} \cos(x_i) \exp\left(\sum_{i=1}^{n} (-(x_i - \pi)^2) \right)$$

Optimal: $x = [\pi, ..., \pi]$, $f(x) = -1$

E. Multi-Objective Test Functions

This appendix covers the multi-objective test functions that has been used in this thesis. n is the number of decision variables and x is the set of decision variables. m is the number of objectives.

E.1. DTLZ

For all the DTLZ test functions $\mathbf{x_m}$ is defined as the following:
$k = n - M + 1$
$\mathbf{x_m} = x[n-k], ..., x[n]$

DTLZ1

$g(\mathbf{x_m}) = 100\left(|\mathbf{x_m}| + \sum_{x_i \in \mathbf{x_m}} ((x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)))\right)$
$F_1(x) = \frac{1}{2}(1 + g(\mathbf{x_m})) \prod_{i=1}^{m-1} x_i$
$F_2(x) = \frac{1}{2}(1 + g(\mathbf{x_m}))(1 - x_{m-1}) \prod_{i=1}^{m-2} x_i$
$\vdots$
$F_{m-1}(x) = \frac{1}{2}(1 + g(\mathbf{x_m}))(1 - x_2)x_1$
$F_m(x) = \frac{1}{2}(1 - x_1)(1 + g(\mathbf{x_m}))$
$x \in [0, 1]$

DTLZ2

$g(\mathbf{x_m}) = \sum_{x_i \in \mathbf{x_m}} (x_i - 0.5)^2$
$F_1(x) = (1 + g(\mathbf{x_m})) \prod_{i=1}^{m-1} \cos(0.5 x_i \pi)$
$F_2(x) = (1 + g(\mathbf{x_m})) \sin(0.5 x_{m-1} \pi) \prod_{i=1}^{m-2} \cos(0.5 x_i \pi)$
$\vdots$
$F_m(x) = (1 + g(\mathbf{x_m})) \sin(0.5 x_1 \pi)$
$x \in [0, 1]$

DTLZ3

$g(\mathbf{x_m}) = 100\left(|\mathbf{x_m}| + \sum_{x_i \in \mathbf{x_m}} ((x_i - 0.5)^2 - \cos(20\pi(x_i - 0.5)))\right)$
$F_1(x) = (1 + g(\mathbf{x_m})) \prod_{i=1}^{m-1} \cos(0.5 x_i \pi)$
$F_2(x) = (1 + g(\mathbf{x_m})) \sin(0.5 x_{m-1} \pi) \prod_{i=1}^{m-2} \cos(0.5 x_i \pi)$
$\vdots$
$F_m(x) = (1 + g(\mathbf{x_m})) \sin(0.5 x_1 \pi)$
$x \in [0, 1]$

DTLZ4

$\alpha = 100$
$g(\mathbf{x_m}) = \sum_{x_i \in \mathbf{x_m}} (x_i - 0.5)^2$
$F_1(x) = (1 + g(\mathbf{x_m})) \prod_{i=1}^{m-1} \cos(0.5 x_i^\alpha \pi)$
$F_2(x) = (1 + g(\mathbf{x_m})) \sin(0.5 x_{m-1}^\alpha \pi) \prod_{i=1}^{m-2} \cos(0.5 x_i^\alpha \pi)$
$\vdots$
$F_m(x) = (1 + g(\mathbf{x_m})) \sin(0.5 x_1^\alpha \pi)$
$x \in [0, 1]$

DTLZ5

$g(\mathbf{x_m}) = \sum_{x_i \in \mathbf{x_m}} (x_i - 0.5)^2$
$\theta_i = \frac{\pi}{4(1+g(\mathbf{x_m}))}(1 + 2g(\mathbf{x_m}) x_i)$ for i = 2, 3, ..., (m - 1)
$F_1(x) = (1 + g(\mathbf{x_m})) \prod_{i=1}^{m-1} \cos(0.5 \theta_i \pi)$
$F_2(x) = (1 + g(\mathbf{x_m})) \sin(0.5 \theta_{m-1} \pi) \prod_{i=1}^{m-2} \cos(0.5 \theta_i \pi)$
$\vdots$
$F_m(x) = (1 + g(\mathbf{x_m})) \sin(0.5 \theta_1 \pi)$
$x \in [0, 1]$

DTLZ6

$g(\mathbf{x_m}) = \sum_{x_i \in \mathbf{x_m}} x_i^{0.1}$
$\theta_i = \frac{\pi}{4(1+g(\mathbf{x_m}))}(1 + 2g(\mathbf{x_m})x_i)$ for i = 2, 3, ..., (m - 1)
$F_1(x) = (1 + g(\mathbf{x_m})) \prod_{i=1}^{m-1} \cos(0.5\theta_i\pi)$
$F_2(x) = (1 + g(\mathbf{x_m})) \sin(0.5\theta_{m-1}\pi) \prod_{i=1}^{m-2} \cos(0.5\theta_i\pi)$
$\vdots$
$F_m(x) = (1 + g(\mathbf{x_m})) \sin(0.5\theta_1\pi)$
$x \in [0, 1]$

DTLZ7

$g(\mathbf{x_m}) = 1 + \frac{9}{|\mathbf{x_m}|} \sum_{x_i \in \mathbf{x_m}} x_i$
$h(F_1, F_2, \ldots, F_{m-1}, g) = m - \sum_{i=1}^{m-1} \left(\frac{F_i}{1+g}(1 + \sin 3\pi F_i) \right)$
$F_1(x) = x_1$
$\vdots$
$F_{m-1}(x) = x_{m-1}$
$F_m(x) = (1 + g(\mathbf{x_m}))h(F_1, F_2, \ldots, F_{m-1}, g)$
$x \in [0, 1]$

E.2. UF

Many of the UF test functions are structured similarly. The decision variables x are split into either two or three groups J based on the number of objectives.

UF1

$F_1 = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left[x_j - \sin(6\pi x_1 + \frac{j\pi}{n}) \right]^2$
$F_2 = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left[x_j - \sin(6\pi x_1 + \frac{j\pi}{n}) \right]^2$

$J_1 = \{j | j \text{ is odd and } 2 \leq j \leq n\}$
$J_2 = \{j | j \text{ is even and } 2 \leq j \leq n\}$
$x_1 \in [0, 1], x_i \in [-1, 1], i = 2, \cdots, n$

UF2

$F_1 = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} y_j^2$
$F_2 = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} y_j^2$

$J_1 = \{j | j \text{ is odd and } 2 \leq j \leq n\}$
$J_2 = \{j | j \text{ is even and } 2 \leq j \leq n\}$
$y_j = \begin{cases} x_j - \left(0.3x_1^2 \cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1\right) \cos(6\pi x_1 + \frac{j\pi}{n}) \text{ for } j \in J_1 \\ x_j - \left(0.3x_1^2 \cos(24\pi x_1 + \frac{4j\pi}{n}) + 0.6x_1\right) \sin(6\pi x_1 + \frac{j\pi}{n}) \text{ for } j \in J_2 \end{cases}$
$x_1 \in [0, 1], x_i \in [-1, 1], i = 2, \ldots, n$

UF3

$F_1 = x_1 + \frac{2}{|J_1|}(4 \sum_{j \in J_1} y_j^2 - 2 \prod_{j \in J_1} \cos(\frac{20 y_j \pi}{\sqrt{j}}) + 2)$
$F_2 = 1 - \sqrt{x_1} + \frac{2}{|J_2|}(4 \sum_{j \in J_2} y_j^2 - 2 \prod_{j \in J_2} \cos(\frac{20 y_j \pi}{\sqrt{j}}) + 2)$

$J_1 = \{j | j \text{ is odd and } 2 \leq j \leq n\}$
$J_2 = \{j | j \text{ is even and } 2 \leq j \leq n\}$
$y_j = x_j - x_1^{0.5(1.0 + \frac{3(j-2)}{n-2})}, j = 2, \ldots, n$
$x \in [0, 1]$

UF4

$F_1 = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} h(y_j)$
$F_2 = 1 - x_1^2 + \frac{2}{|J_2|} \sum_{j \in J_2} h(y_j)$

$J_1 = \{j | j \text{ is odd and } 2 \leq j \leq n\}$
$J_2 = \{j | j \text{ is even and } 2 \leq j \leq n\}$
$y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2, \ldots, n$
$h(t) = \frac{|t|}{1 + e^{2|t|}}$
$x_1 \in [0, 1], x_i \in [-2, 2], i = 2, \ldots, n$

UF5

$F_1 = x_1 + (\frac{1}{2N} + \varepsilon)|\sin(2N\pi x_1)| + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j)$
$F_2 = 1 - x_1 + (\frac{1}{2N} + \varepsilon)|\sin(2N\pi x_1)| + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j)$

$N = 10, \varepsilon = 0.1$
$J_1 = \{j|j \text{ is odd and } 2 \leq j \leq n\}$
$J_2 = \{j|j \text{ is even and } 2 \leq j \leq n\}$
$y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2, \ldots, n$
$h(t) = 2t^2 - \cos(4\pi t) + 1$
$x_1 \in [0, 1], x_i \in [-1, 1], i = 2, \ldots, n$

UF6

$F_1 = x_1 + \max\{0, 2(\frac{1}{2N} + \varepsilon)\sin(2N\pi x_1)\} + \frac{2}{|J_1|}(4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos(\frac{20y_j\pi}{\sqrt{j}}) + 2)$
$F_2 = 1 - x_1 + \max\{0, 2(\frac{1}{2N} + \varepsilon)\sin(2N\pi x_1)\} + \frac{2}{|J_2|}(4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos(\frac{20y_j\pi}{\sqrt{j}}) + 2)$

$N = 2, \varepsilon = 0.1$
$J_1 = \{j|j \text{ is odd and } 2 \leq j \leq n\}$
$J_2 = \{j|j \text{ is even and } 2 \leq j \leq n\}$
$y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2, \ldots, n$
$x_1 \in [0, 1], x_i \in [-1, 1], i = 2, \ldots, n$

UF7

$F_1 = \sqrt[5]{x_1} + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2$
$F_2 = 1 - \sqrt[5]{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2$

$J_1 = \{j|j \text{ is odd and } 2 \leq j \leq n\}$
$J_2 = \{j|j \text{ is even and } 2 \leq j \leq n\}$
$y_j = x_j - \sin(6\pi x_1 + \frac{j\pi}{n}), j = 2, \ldots, n$
$x_1 \in [0, 1], x_i \in [-1, 1], i = 2, \ldots, n$

UF8

$$F_1 = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2$$
$$F_2 = \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2$$
$$F_3 = \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}))^2$$

$$J_1 = \{j|3 \le j \le n, \text{ and } j-1 \text{ is a multiplication of 3 }\}$$
$$J_2 = \{j|3 \le j \le n, \text{ and } j-2 \text{ is a multiplication of 3 }\}$$
$$J_3 = \{j|3 \le j \le n, \text{ and } j \text{ is a multiplication of 3 }\}$$
$$x_1 \in [0,1], x_2 \in [0,1], x_i \in [-2,2]\,, i = 3,\ldots,n$$

UF9

$$F_1 = 0.5[max\{0,(1+\varepsilon)(1-4(2x_1-1)^2)\}+2x_1]x_2 + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n})\right)^2$$
$$F_2 = 0.5[max\{0,(1+\varepsilon)(1-4(2x_1-1)^2)\}-2x_1+2]x_2 + \frac{2}{|J_2|}\sum_{j\in J_2}\left(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n})\right)^2$$
$$F_3 = 1 - x_2 + \frac{2}{|J_3|}\sum_{j\in J_3}\left(x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n})\right)^2$$

$$\varepsilon = 0.1$$
$$J_1 = \{j|3 \le j \le n, \text{ and } j-1 \text{ is a multiplication of 3 }\}$$
$$J_2 = \{j|3 \le j \le n, \text{ and } j-2 \text{ is a multiplication of 3 }\}$$
$$J_3 = \{j|3 \le j \le n, \text{ and } j \text{ is a multiplication of 3 }\}$$
$$x_1 \in [0,1], x_2 \in [0,1], x_i \in [-2,2]\,, i = 3,\ldots,n$$

UF10

$$F_1 = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j\in J_1}\left(4y_j^2 - \cos(8\pi y_j) + 1\right)$$
$$F_2 = \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left(4y_j^2 - \cos(8\pi y_j) + 1\right)$$
$$F_3 = \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left(4y_j^2 - \cos(8\pi y_j) + 1\right)$$

$$J_1 = \{j|3 \le j \le n, \text{ and } j-1 \text{ is a multiplication of 3 }\}$$
$$J_2 = \{j|3 \le j \le n, \text{ and } j-2 \text{ is a multiplication of 3 }\}$$
$$J_3 = \{j|3 \le j \le n, \text{ and } j \text{ is a multiplication of 3 }\}$$
$$y_j = x_j - 2x_2\sin(2\pi x_1 + \frac{j\pi}{n}), j = 3,\ldots,n$$
$$x_1 \in [0,1], x_2 \in [0,1], x_i \in [-2,2]\, i = 3,\ldots,n$$

E.3. ZDT

ZDT1-4 follow a very similar structure where F_1 is only dependent on the first decision variable x_1, while ZDT6 uses a more complex calculation of F_1.

ZDT1

$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right) / (n-1)$
$F_1(x) = x_1$
$F_2(x) = g(x)\left[1 - \sqrt{x_1/g(x)}\right]$
$x \in [0,1]$

ZDT2

$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right) / (n-1)$
$F_1(x) = x_1$
$F_2(x) = g(x)\left[1 - (x_1/g(x))^2\right]$
$x \in [0,1]$

ZDT3

$g(x) = 1 + 9\left(\sum_{i=2}^{n} x_i\right) / (n-1)$
$F_1(x) = x_1$
$F_2(x) = g(x)\left[1 - \sqrt{x_1/g(x)} - x_1/g(x)\sin(10\pi x_1)\right]$
$x \in [0,1]$

ZDT4

$g(x) = 1 + 10(n-1) + \sum_{i=2}^{n}\left[x_i^2 - 10\cos(4\pi x_i)\right]$
$F_1(x) = x_1$
$F_2(x) = g(x)\left[1 - \sqrt{x_1/g(x)}\right]$
$x_1 \in [0,1], x_i \in [-5,5]\, i = 2,\ldots,n$

ZDT6

$g(x) = 1 + 9\left[\left(\sum_{i=2}^{n} x_i\right) / (n-1)\right]^{0.25}$
$F_1(x) = 1 - \exp(-4x_1)\sin^6(6\pi\ x_1)$
$F_2(x) = g(x)\left[1 - (f_1(x)/g(x))^2\right]$
$x \in [0,1]$

Bibliography

Abbass, H. A. (2001). Mbo: Marriage in honey bees optimization-a haplometrosis polygynous swarming approach. In *Evolutionary Computation, 2001. Proceedings of the 2001 Congress on*, volume 1, pages 207–214. IEEE.

Bandyopadhyay, S., Saha, S., Maulik, U., and Deb, K. (2008). A simulated annealing-based multiobjective optimization algorithm: Amosa. *IEEE transactions on evolutionary computation*, 12(3):269–283.

Birattari, M., Paquete, L., Stützle, T., and Varrentrapp, K. (2001). Classification of metaheuristics and design of experiments for the analysis of components. *Teknik Rapor, AIDA-01-05*.

Blum, C. and Roli, A. (2003). Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM computing surveys (CSUR)*, 35(3):268–308.

Branke, J., Kaußler, T., and Schmeck, H. (2001). Guidance in evolutionary multi-objective optimization. *Advances in Engineering Software*, 32(6):499–507.

Bui, L. T., Nguyen, T. T., et al. (2017). A modified dual-population approach for solving multi-objective problems. In *Intelligent and Evolutionary Systems (IES), 2017 21st Asia Pacific Symposium on*, pages 89–94. IEEE.

Cheng, M.-Y. and Prayogo, D. (2014). Symbiotic organisms search: a new metaheuristic optimization algorithm. *Computers & Structures*, 139:98–112.

Cheng, R., Jin, Y., Olhofer, M., et al. (2017a). Test problems for large-scale multi-objective and many-objective optimization. *IEEE transactions on cybernetics*, 47(12):4108–4121.

Cheng, R., Li, M., Tian, Y., Zhang, X., Yang, S., Jin, Y., and Yao, X. (2017b). Benchmark functions for the cec'2017 competition on many-objective optimization. Technical report, University of Birmingham, UK.

Coello, C. A. C. (2015). Multi-objective evolutionary algorithms in real-world applications: Some recent results and current challenges. In *Advances in evolutionary and deterministic methods for design, optimization and control in engineering and sciences*, pages 3–18. Springer.

Coello, C. A. C. and Lamont, G. B. (2004). *Applications of multi-objective evolutionary algorithms*, volume 1. World Scientific.

Coello, C. A. C., Lamont, G. B., Van Veldhuizen, D. A., et al. (2007). *Evolutionary algorithms for solving multi-objective problems*, volume 5. Springer.

Coello, C. A. C., Pulido, G. T., and Lechuga, M. S. (2004). Handling multiple objectives with particle swarm optimization. *IEEE Transactions on evolutionary computation*, 8(3):256–279.

Coello, C. A. C. and Sierra, M. R. (2004). A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm. In *Mexican International Conference on Artificial Intelligence*, pages 688–697. Springer.

Deb, K., Agrawal, S., Pratap, A., and Meyarivan, T. (2000). A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: Nsga-ii. In *International Conference on Parallel Problem Solving From Nature*, pages 849–858. Springer.

Deb, K. and Algorithms, M.-O. E. (1999). Introducing bias among pareto-optimal solutions. *Kanpur Genetic Algorithms Laboratory Report*, (99002).

Deb, K. and Jain, H. (2014). An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part i: Solving problems with box constraints. *IEEE Trans. Evolutionary Computation*, 18(4):577–601.

Deb, K., Thiele, L., Laumanns, M., and Zitzler, E. (2005). Scalable test problems for evolutionary multiobjective optimization. In *Evolutionary multiobjective optimization*, pages 105–145. Springer.

Derrac, J., García, S., Molina, D., and Herrera, F. (2011). A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms. *Swarm and Evolutionary Computation*, 1(1):3–18.

Dorigo, M., Maniezzo, V., and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41.

Durillo, J. J. and Nebro, A. J. (2011). jmetal: A java framework for multi-objective optimization. *Advances in Engineering Software*, 42(10):760–771.

Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pages 39–43. IEEE.

Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association*, 32(200):675–701.

Goldberg, D. E. and Holland, J. H. (1988). Genetic algorithms and machine learning. *Machine learning*, 3(2):95–99.

Gong, C., Han, S., Li, X., Zhao, L., and Liu, X. (2017). A new dandelion algorithm and optimization for extreme learning machine. *Journal of Experimental & Theoretical Artificial Intelligence*, pages 1–14.

Hasenjäger, M., Sendhoff, B., Sonoda, T., and Arima, T. (2005). Three dimensional evolutionary aerodynamic design optimization with cma-es. In *Proceedings of the 7th annual conference on Genetic and evolutionary computation*, pages 2173–2180. ACM.

Hodges, J., Lehmann, E. L., et al. (1962). Rank methods for combination of independent experiments in analysis of variance. *The Annals of Mathematical Statistics*, 33(2):482–497.

Holm, S. (1979). A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pages 65–70.

Hornby, G., Globus, A., Linden, D., and Lohn, J. (2006). Automated antenna design with evolutionary algorithms. In *Space 2006*, page 7242.

Huband, S., Barone, L., While, L., and Hingston, P. (2005). A scalable multi-objective test problem toolkit. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 280–295. Springer.

Hutson, M. (2018). Artificial intelligence faces reproducibility crisis.

Jain, A. K. and Dubes, R. C. (1988). Algorithms for clustering data.

Karaboga, D. and Basturk, B. (2007). A powerful and efficient algorithm for numerical function optimization: artificial bee colony (abc) algorithm. *Journal of global optimization*, 39(3):459–471.

Kirkpatrick, S., Gelatt, C. D., and Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, 220(4598):671–680.

Knowles, J. and Corne, D. (1999). The pareto archived evolution strategy: A new baseline algorithm for pareto multiobjective optimisation. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 1, pages 98–105. IEEE.

Kofod-Petersen, A. (2014). How to do a structured literature review in computer science. *Ver. 0.2. October*, 1.

Konak, A., Coit, D. W., and Smith, A. E. (2006). Multi-objective optimization using genetic algorithms: A tutorial. *Reliability Engineering & System Safety*, 91(9):992–1007.

Kumawat, I. R., Nanda, S. J., and Maddila, R. K. (2017). Multi-objective whale optimization. In *TENCON 2017 - 2017 IEEE Region 10 Conference*, pages 2747–2752.

Laumanns, M., Thiele, L., Deb, K., and Zitzler, E. (2002). Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282.

Li, K., Deb, K., Zhang, Q., and Kwong, S. (2015). An evolutionary many-objective optimization algorithm based on dominance and decomposition. *IEEE Transactions on Evolutionary Computation*, 19(5):694–716.

Li, X., Zhang, J., and Yin, M. (2014). Animal migration optimization: an optimization algorithm inspired by animal migration behavior. *Neural Computing and Applications*, 24(7-8):1867–1877.

Liang, J., Qu, B., and Suganthan, P. (2013). Problem definitions and evaluation criteria for the cec 2014 special session and competition on single objective real-parameter numerical optimization. *Computational Intelligence Laboratory, Zhengzhou University, Zhengzhou China and Technical Report, Nanyang Technological University, Singapore*.

Luke, S. (2009). *Essentials of metaheuristics*, volume 113. Lulu Raleigh.

Mehrabian, A. R. and Lucas, C. (2006). A novel numerical optimization algorithm inspired from weed colonization. *Ecological informatics*, 1(4):355–366.

Meunier, H., Talbi, E.-G., and Reininger, P. (2000). A multiobjective genetic algorithm for radio network optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 317–324. IEEE.

Michael, R. G. and David, S. J. (1979). Computers and intractability: a guide to the theory of np-completeness. *WH Free. Co., San Fr*, pages 90–91.

Mirjalili, S. (2015a). The ant lion optimizer. *Advances in Engineering Software*, 83:80–98.

Mirjalili, S. (2015b). Moth-flame optimization algorithm: A novel nature-inspired heuristic paradigm. *Knowledge-Based Systems*, 89:228–249.

Mirjalili, S. (2016). Dragonfly algorithm: a new meta-heuristic optimization technique for solving single-objective, discrete, and multi-objective problems. *Neural Computing and Applications*, 27(4):1053–1073.

Mirjalili, S., Jangir, P., and Saremi, S. (2017). Multi-objective ant lion optimizer: a multi-objective optimization algorithm for solving engineering problems. *Applied Intelligence*, 46(1):79–95.

Mirjalili, S. and Lewis, A. (2016). The whale optimization algorithm. *Advances in Engineering Software*, 95:51–67.

Mirjalili, S., Saremi, S., Mirjalili, S. M., and Coelho, L. d. S. (2016). Multi-objective grey wolf optimizer: a novel algorithm for multi-criterion optimization. *Expert Systems with Applications*, 47:106–119.

Nebro, A. J., Durillo, J. J., Garcia-Nieto, J., Coello, C. C., Luna, F., and Alba, E. (2009a). Smpso: A new pso-based metaheuristic for multi-objective optimization. In *Computational intelligence in miulti-criteria decision-making, 2009. mcdm'09. ieee symposium on*, pages 66–73. IEEE.

Nebro, A. J., Durillo, J. J., Luna, F., Dorronsoro, B., and Alba, E. (2009b). Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7):726–746.

Nebro, A. J., Luna, F., Alba, E., Dorronsoro, B., Durillo, J. J., and Beham, A. (2008). Abyss: Adapting scatter search to multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 12(4):439–457.

Nikoofard, A. H., Hajimirsadeghi, H., Rahimi-Kian, A., and Lucas, C. (2012). Multiobjective invasive weed optimization: Application to analysis of pareto improvement models in electricity markets. *Applied Soft Computing*, 12(1):100–112.

Parpinelli, R. S. and Lopes, H. S. (2011). New inspirations in swarm intelligence: a survey. *International Journal of Bio-Inspired Computation*, 3(1):1–16.

Pham, D. T., Ghanbarzadeh, A., Koç, E., Otri, S., Rahim, S., and Zaidi, M. (2006). -the bees algorithm—a novel tool for complex optimisation problems. In *Intelligent Production Machines and Systems*, pages 454–459. Elsevier.

Rao, R. V. and Waghmare, G. (2014). A comparative study of a teaching–learning-based optimization algorithm on multi-objective unconstrained and constrained functions. *Journal of King Saud University-Computer and Information Sciences*, 26(3):332–346.

Reyes-Sierra, M., Coello, C. C., et al. (2006). Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3):287–308.

Santiago, A., Huacuja, H. J. F., Dorronsoro, B., Pecero, J. E., Santillan, C. G., Barbosa, J. J. G., and Monterrubio, J. C. S. (2014). A survey of decomposition methods for multi-objective optimization. In *Recent Advances on Hybrid Approaches for Designing Intelligent Systems*, pages 453–465. Springer.

Savsani, V. and Tawhid, M. A. (2017). Non-dominated sorting moth flame optimization (ns-mfo) for multi-objective problems. *Engineering Applications of Artificial Intelligence*, 63:20–32.

Schrijver, A. (2005). On the history of combinatorial optimization (till 1960). *Handbooks in operations research and management science*, 12:1–68.

Sierra, M. R. and Coello, C. A. C. (2005). Improving pso-based multi-objective optimization using crowding, mutation and ϵ-dominance. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 505–519. Springer.

Srinivas, N. and Deb, K. (1994). Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary computation*, 2(3):221–248.

Stützle, T. (1999). Local search algorithms for combinatorial problems-analysis, algorithms and new applications. *DISKI-Dissertationen zur Künstlichen Intelligenz, Infix, Sankt Augustin, Germany*.

Tran, D.-H., Cheng, M.-Y., and Prayogo, D. (2016). A novel multiple objective symbiotic organisms search (mosos) for time–cost–labor utilization tradeoff problem. *Knowledge-Based Systems*, 94:132–145.

Veldhuizen, D. A. V. (1999). Multiobjective evolutionary algorithms: Classifications, analyses, and new innovations. Technical report, Evolutionary Computation.

Veldhuizen, D. A. V. and Lamont, G. B. (2000). Multiobjective evolutionary algorithms: Analyzing the state-of-the-art. *Evolutionary computation*, 8(2):125–147.

Wang, G.-G. (2016). Moth search algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. *Memetic Computing*, pages 1–14.

Wang, G.-G., Deb, S., and Coelho, L. (2015). Earthworm optimization algorithm: a bio-inspired metaheuristic algorithm for global optimization problems. *International Journal of Bio-Inspired Computation*.

Wolpert, D. H. and Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE transactions on evolutionary computation*, 1(1):67–82.

Yang, X.-S. (2010). Firefly algorithm, stochastic test functions and design optimisation. *International Journal of Bio-Inspired Computation*, 2(2):78–84.

Yang, X.-S. (2012). Flower pollination algorithm for global optimization. In *International conference on unconventional computing and natural computation*, pages 240–249. Springer.

Yang, X.-S. and Deb, S. (2009). Cuckoo search via lévy flights. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 210–214. IEEE.

Yang, X.-S., Karamanoglu, M., and He, X. (2013). Multi-objective flower algorithm for optimization. *Procedia Computer Science*, 18:861–868.

Yazdani, M. and Jolai, F. (2016). Lion optimization algorithm (loa): a nature-inspired metaheuristic algorithm. *Journal of computational design and engineering*, 3(1):24–36.

Yu, Y. (1997). Multiobjective decision theory for computational optimization in radiation therapy. *Medical Physics*, 24(9):1445–1454.

Zhang, Q. and Li, H. (2007). Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731.

Zhang, Q., Zhou, A., Zhao, S., Suganthan, P. N., Liu, W., and Tiwari, S. (2008). Multiobjective optimization test instances for the cec 2009 special session and competition. *University of Essex, Colchester, UK and Nanyang technological University, Singapore, special session on performance assessment of multi-objective optimization algorithms, technical report*, 264.

Zhang, X., Tian, Y., Cheng, R., and Jin, Y. (2015). An efficient approach to nondominated sorting for evolutionary multiobjective optimization. *IEEE Transactions on Evolutionary Computation*, 19(2):201–213.

Zitzler, E., Deb, K., and Thiele, L. (2000). Comparison of multiobjective evolutionary algorithms: Empirical results. *Evolutionary computation*, 8(2):173–195.

Zitzler, E., Laumanns, M., and Thiele, L. (2001). Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103.

Zitzler, E. and Thiele, L. (1999). Multiobjective evolutionary algorithms: a comparative case study and the strength pareto approach. *IEEE transactions on Evolutionary Computation*, 3(4):257–271.