VILNIUS UNIVERSITY

Vytautas Jančauskas

EVALUATING THE PERFORMANCE OF MULTI-OBJECTIVE PARTICLE SWARM OPTIMIZATION ALGORITHMS

Doctoral dissertation
Physical sciences, informatics (09P)

Vilnius, 2016

The dissertation work was carried out at Vilnius University from 2011 to 2015.

Scientific supervisor:

prof. habil. dr. Antanas Žilinskas (Vilnius University, physical sciences, informatics - 09P)

VILNIAUS UNIVERSITETAS

Vytautas Jančauskas

DALELIŲ SPIEČIŲ OPTIMIZAVIMO ALGORITMŲ TAIKYMO DAUGIAKRITERIAMS UŽDAVINIAMS EFEKTYVUMO TYRIMAS

Daktaro disertacija
Fiziniai mokslai, informatika (09P)

Vilnius, 2016

Disertacija rengta 2011 - 2015 metais Vilniaus universitete.

Mokslinis vadovas:

prof. habil. dr. Antanas Žilinskas (Vilniaus universitetas, fiziniai mokslai, informatika - 09P)

Abstract

In this thesis the problem of evaluating the performance of multi-objective optimization methods for non-convex problems is examined. Namely, the performance of multi-objective particle swarm optimization methods are investigated. An overview of these methods is provided in this thesis covering most methods described in literature. A novel classification system of these methods is developed. This system uses method design choices to classify them. A thorough experimental analysis of existing methods is given. Each method is tested using a wide variety of test problems. The results are further analyzed with regards to what types of problems each method solves best. An important aspect of solution quality when it comes to multi-objective problems is the uniformity of solution spread along the real Pareto frontier. Due to the inadequacies of existing performance indicators when it comes to measuring Pareto frontier approximation solution spread, two new indicators are proposed. These two indicators are designed to capture the intuitive notion of solution spread uniformity. They are discussed in comparison with existing indicators. Two new multi-objective particle swarm optimization methods are proposed in the thesis as well. These methods are based on the idea of heterogeneous swarms — swarms where several different types of particles are used at the same type. The particles share information via the same non-dominated point archive.

Table of Contents

Notation

PSO – Particle Swarm Optimization
MOPSO – Multi-Objective Particle Swarm Optimization
PF – Pareto frontier
PS – Pareto set
n – Number of particles in the swarm
d – Problem dimensionality
k – Number of objectives
$\boldsymbol{x}_i$ – Position vector for particle i in solution space
$\boldsymbol{v}_i$ – Velocity vector for particle i
$\boldsymbol{p}_i$ – Best solution found by particle i so far
$\boldsymbol{g}_i$ – Best solution found by neighbours of particle i
f_i – The i-th objective of the form $f : \mathbb{R}^d \to \mathbb{R}$
f – A single objective of the form $f : \mathbb{R}^d \to \mathbb{R}$
w – Inertia coefficient
t – Current PSO algorithm iteration
t_{max} – Maximum number of PSO algorithm iterations if the number of iterations is fixed
$\boldsymbol{U}_{(a,b)}$ – A vector of uniformly distributed random numbers from the interval (a, b), it's dimensionality is to be determined from context
ρ_i – Influence of some attractor in PSO solution space, in case of canonical PSO algorithm ρ_1 is the influence of personal experience and ρ_2 is social influence
$U_(a, b)$ – Uniformly distributed random numbers from the interval (a, b)

Chapter 1

Introduction

Multi-objective optimization problems are the problems, where several (possibly contradictory) objectives have to be optimized at the same time. This means a single solution will be evaluated with regards to several different criteria. Most problems in engineering, economics and other areas of human activity are of this nature. For example, one might want to increase the durability of a manufactured part, but also to decrease it's price. It is easy to see how those two things may be in contradiction to one another.Most of the time, no single solution will be sufficient for the problems like these. Instead, methods that attempt to solve these problems will look for a Pareto frontier. A Pareto frontier is a set of points in the objective space, where no point dominates any other point. The concept of dominance will be discussed later. For the time being, it is sufficient to say that solutions corresponding to points in such a set cannot be improved in terms of one objective without sacrificing some other objective.

There are a lot of methods for solving or attempting to solve the multi-objective problems. It is possible to solve such problems analytically only for a very small subset of them. If nothing or little is known about a problem or if it is non-convex, exact methods for finding solutions generally do not exist. In such cases, various heuristic methods are used. Very often these are population based meta-heuristic methods. It is often the case, that methods created for single objective optimization are adapted to work with multi-objective problems. A simple way of doing so is to aggregate objectives to a single objective using weights. That way, a problem with several objectives is converted into several

single objective problems by varying those weights. This will then require running the method in question on each one of these single objective problems and storing the results. It would be ideal that one could obtain an approximation of the complete Pareto frontier in a single run of the algorithm. The nature of the multi-objective problems compared to single objective ones raises several complications when using this approach. First of all, a set of solutions, but not a single solution will have to be found. This is often solved by using a non-dominated point archive that maintains such sets using dominance criteria for accepting or rejecting a solution. Solutions are accepted into the archive if they are non-dominated by any solution already in the archive. Second of all, diversity of solutions has to be maintained throughout. We want the solutions found by the method to cover the whole of the actual Pareto frontier. This is contrary to single objective optimisation where the population will converge to a point. Heuristic algorithms will have to be changed so, that population converges to a set instead. Popular meta-heuristic methods used with multi-objective problems are genetic algorithms that work by selecting best individuals from a population and creating new solutions based on them. To work with multi-objective problems they are modified to take the above mentioned into account.

Particle Swarm Optimisation (PSO) is a population based method inspired by social behaviour of swarming animals like birds or fish. Namely, swarming (in birds and others) or schooling (in fish) behaviour is taken as an inspiration. Particles are points in the solution space that move according to the movement and velocity update rules. These rules are designed so, that particles move towards the best solution they found so far and also towards the best solution their neighbours have found. This can be viewed as particles using personal and social experience and combining them (with random weighting) to come up with new solutions. Particle Swarm Optimisation was adapted to solve multi-objective optimisation problems in ways similar to the ones used in Genetic Algorithms. There is active ongoing research into Multi-Objective Particle Swarm Optimisation (MOPSO) algorithms. Many algorithms and their variations have been proposed in literature.

There are two important issues to consider when it comes to our understanding of using PSO for multi-objective optimisation. First of all, it is unclear which of

the existing approaches is the most promising or deserves further investigation. The existing methods are often compared only to one or two other approaches. Furthermore, they are often compared only using a couple of test problems. As such, there is no way of telling which method (or group of methods) is the most promising. In particular, it would be interesting to see which choices in optimisation method design lead to successful methods. The second issue has to do with comparing two multi-objective optimisation algorithms. There are no agreed procedures to compare two solutions of a multi-objective problem. In the case of the single objective optimisation it is simple: we can compare the fitness function values obtained by the two algorithms, we can also compare the time in function evaluations (since they are usually the time limiting factor) or algorithm iterations it takes for the algorithm to find the answer to a given precision. We can also compare the percentage of times the method finds the global minimum versus the percentage of times it does not. Each of these aspects is straightforward to measure and interpret. It is not yet the case for the multi-objective optimisation. Here solutions are sets and comparing of two sets is difficult. It usually depends on what the user of the method perceives to be a good solution.

1.1 Aim and Object of the Study

The aim of this research is to improve existing methods for evaluating the performance of non-convex multi-objective optimization methods. Performance, in this case, consists of finding a good approximation to the Pareto frontier of some multi-objective problem. These methods are then used to evaluate multi-objective particle swarm optimization algorithms. Particle swarm optimization algorithms are popular for non-convex multi-objective optimization. This is evident from the extensive research done in the area of applying particle swarm optimization to multi-objective problems. However, no consistent and thorough research exists that examines these methods as a whole. As such, a study like this one seems beneficial to the field. The folowing subtasks were identified:

- Overview performance indicators meant to evaluate performance of non-convex multi-objective optimization methods. These indicators are de-

signed to measure aspects of the optimizer performance. The fit between the Pareto frontier of a problem and an approximation of that Pareto frontier produced by the optimizer is usually measured. The fit can be measured in several aspects, for example, the average distance of the points in the approximation to the Pareto frontier can be calculated. Or how uniformly the points cover the Pareto frontier. If the existing performance indicators seem inadequate to the task, new ones are to be proposed.

- Review MOPSO methods described in literature. Examine what design choices and underlying ideas are commonly used. Based on this information propose a new classification scheme if existing ones are not adequate to account for the findings of the review.

- There do not exist readily available implementations of many of the MOPSO methods that are described in the literature. To aid in this, develop a software framework that allows to implement and evaluate these methods. This will consist of implementations of the methods themselves, test problems and performance indicators. These can then be used together to evaluate the methods experimentally. This has to be done in a convenient and modular way, so as new methods can be implemented easily when needed.

- Using the aforementioned software framework, experimentally evaluate existing MOPSO methods. The purpose of this is to do an experimental comparative analysis. This will allow us to judge which methods are most suited for which kinds of problems. That's because the properties of the problems and the design choices behind the algorithms are known. Analyze the results to determine the types of MOPSO methods that are the most promising for certain types of multi-objective problems.

- Propose new MOPSO methods using the information obtained by studying and evaluating existing methods and what works and does not work in them. The data for this subtask will be obtained through the work done during the previous subtasks.

1.2 Methodology

In order to evaluate MOPSO performance, computational experiments were performed over many different MOPSO variants and the results were evaluated using several different performance indicators. Experiments were performed using both existing and proposed MOPSO variants. Convenient to use and install Python libraries were written when preparing the thesis. Libraries support single and multi-objective optimization using PSO methods. Various swarm topologies and particle update rules are supported. The software is modular and allows an easy implementation and testing of new MOPSO algorithms and convenient analysis of the results. All experiments were performed on supercomputing clusters. The need to run the experiments in a distributed computing environment arises because of the requirements for computing time inherent in these experiments. If there are n MOPSO methods, m multi-objective test problems and we want to perform k experiments (to get statistically meaningful results) we have to perform $n \times m \times k$ experiments. To get a Pareto frontier approximation for an expensive problem it often takes several minutes of time. Therefore ,the number of required time can easily approach weeks or even months if done on a single computer core. Existing library infrastructure uses distributed computing to run multiple experiments in parallel and collect the results. MPI was used for communication between tasks.

1.3 Scientific Novelty and Results

While a lot of MOPSO methods were proposed in the literature, there is no systematic overview or evaluation of them. New methods are usually compared to several popular methods and using a couple of test problems. Therefore, a study is needed that provides a more systematic treatment of the properties of existing MOPSO methods. This will allow to single out approaches that show the most promise. The following results were achieved during the study:

- Two new performance indicators were introduced. These are designed to measure how uniformly does a Pareto set approximation cover the real Pareto frontier.

- Two new MOPSO algorithms were proposed. Both use heterogeneous swarm approach in order to improve MOPSO performance. In heterogeneous swarms particles with different velocity and position update rules and other characteristics share the same nondominated point archive to exchange the information. If the particles are designed to accent different aspects of MOPSO performance ,they tend to compensate each others shortcomming and improve performance in terms of all aspects concerned.
- An overview of existing MOPSO algorithms was done. The methods were systematized and classified using several different schemes. Citations trees were constructed.
- Methodologies for comparing different multi-objective optimisation algorithms were proposed. Methodologies consist of recommendations for the experimental procedure, performance indicators and test problems. Methods for analysing the collected performance data were proposed.
- Existing MOPSO algorithms are extensively tested and evaluated relatively to each other. Most of the existing MOPSO algorithms are covered in the study as far as their descriptions are published in scientific literature.
- A large framework was developed for the Python programming language. It allows the user to use tens of different particle types, several different swarm topologies and has tens of test problems to benchmark new algorithms.

1.4 Practical Value

Multi-objective optimisation is important in many different practical applications, since a lot of real world problems are multi-objective optimisation problems. The results of this thesis were applied to the visualisation of business process modelling diagrams, for example. The problem of visualizing business process diagrams was solved by implementing a web service that provides solutions to diagram visualization problems. The service uses multi-objective

population-based methods for optimization and provides the user with a diagram given a set of inputs. The code required by the web service can be found at:

`https://bitbucket.org/bucket_brigade/aco`

Results of this thesis were used to achieve the goals of the following projects:

- "Developing of software for business process modelling and visualization" funded by a grant No. 31V-145 (2011–2012), No. 31V-31 (2013).
- "Nonconvex multiobjective optimization: methods and algorithms" funded by a grant No. MIP-063/2012 from the Research Council of Lithuania (2012–2014).

1.5 Statements to be Defended

- Proposed multi-objective optimisation performance indicators can be used to measure the uniformity of the solution spread in Pareto front approximations. These (or similar) indicators are necessary, because existing performance indicators suffer from the problems that are explained in chapter 4. Because of those problems, existing indicators that are described in chapter 2, do not accurately capture the intuitive notion of the uniform coverage. The problems are solved by the proposed indicators by taking into careful consideration what it means to cover Pareto frontier uniformaly with a discrete approximation of that frontier.
- It can be seen from the experimental evaluations that proposed heterogeneous multi-objective optimization algorithms perform well over a wide selection of performance indicators and test problems compared to the existing methods.
- It has been known for a long time that the use of mutation operators can improve the performance of single objective PSOs (mutation is not used in original PSO). However, comparative studies that contrast the methods that use mutation operators and those that do not,have not been

performed in the field of multi-objective PSO. Because of the large number of MOPSO methods that were surveyed in this research both with and without mutation, conclusions about it's use can be made. It can be seen from the data that mutation can significantly improve the performance of MOPSO methods. With all test problem, the methods that don't use mutation end up with the highest values of I_{IGD} and I_{GD} indicators by orders of magnitude. The only exception are the decomposition based methods, that do comparatively well.

- Decomposition based approaches work well on problems with discontinuous Pareto frontiers. This may be due to the fact that they do not make assumptions about the Pareto frontier being continuous the way most MOPSO approaches do.
- From the experimental results that are gathered in chapter 3 it can be seen that vector evaluated MOPSO methods do not work well compared to other MOPSO approaches. Several vector evaluated approaches have been tested and none of them come close in terms of results to other approaches.

1.6 Publications of the Author

What follows is the list of articles written by the author and published in peer-reviewed literature on topics related to this thesis during the course of writing it.

1.6.1 Periodicals

Work in this thesis was published in the following periodical journals:

1. Jančauskas, Vytautas. Empirical Study of Particle Swarm Optimization Mutation Operators. Baltic Journal of Modern Computing, Vol. 2 (2014), No.4, pp. 199 – 214.

2. Jančauskas, Vytautas. Heterogeneous Multi-Objective Particle Swarm Optimiser with a Spread Particle. International Journal of Research Studies in Science, Engineering and Technology, Vol. 2, Issue 9, September 2015, pp. 30 – 49. ISSN 2349 – 4751.

1.6.2 Conference Presentations

Works in this thesis was presented in the following international conferences:

1. Jančauskas, Vytautas; Žilinskas, Antanas. On Multi-Objective Black Box Optimization of Expensive Objectives. 25th European Conference on Operational Research (EURO XXV), July 2012, Vilnius, Lithuania.
2. Jančauskas, Vytautas; Kaukas, Giedrius; Žilinskas, Antanas; Žilinskas, Julius. On Multi-Objective Optimization Aided Visualization of Graphs Related to Business Process Diagrams. Tenth International Baltic Conference on Databases and Information Systems (Baltic DB&IS), July 2012, Vilnius, Lithuania.
3. Jančauskas, Vytautas. Empirical Study of Particle Swarm Optimization Mutation Operators. Veszprém Optimization Conference: Advanced Algorithms (VOCAL) 2012, December 2012, Veszprém, Hungary.
4. Jančauskas, Vytautas. Optimizing Neighbourhood Distances for a Variant of Fully-Informed Particle Swarm Algorithm. VI International Workshop on Nature Inspired Cooperative Strategies for Optimization (NICSO 2013), September 2013, Canterbury, United Kingdom.
5. Jančauskas, Vytautas. Using Nature-Inspired Optimization Methods to Fit Spectra of Supernovae. 9th Wuerzburg Workshop on Supernovae and Related Topics, December 2015, Wuerzburg, Germany.

1.6.3 Peer Reviewed Conference Proceedings

Works in this thesis was published in the following peer-reviewed conference proceedings:

1. Jančauskas, Vytautas, Aušra Mackute-Varoneckiene, Audrius Varoneckas, and Antanas Žilinskas. "On the multi-objective optimization aided drawing of connectors for graphs related to business process management." In Information and Software Technologies, pp. 87–100. Springer Berlin Heidelberg, 2012.

2. Jančauskas, Vytautas, Giedrius Kaukas, Antanas Žilinskas, and Julius Žilinskas. "On Multi-Objective Optimization Aided Visualization of Graphs Related to Business Process Diagrams." In DB&Local Proceedings, pp. 71–80. 2012.

3. Jančauskas, Vytautas. "Optimizing neighbourhood distances for a variant of fully-informed particle swarm algorithm." In Nature Inspired Cooperative Strategies for Optimization (NICSO 2013), pp. 217–229. Springer International Publishing, 2014.

1.7 Thesis Structure

In the second chapter, the basic concepts behind the topic are layed out. Definitions of the problems of the real-valued global optimization and multi-objective optimization are given. Definitions for Dominance, Pareto Frontier, Pareto Set and Pareto Optimality are given too.Traditional methods for solving multi-objective optimization problems are presented. The reasons for why these methods are not suitable for non-convex problems are also given. Original Particle Swarm Optimization method is briefly described as well as several of it's variants. Several concepts important to the operation of the method are explained. They include neighbourhood topologies and Particle Swarm Optimizer convergence properties. The topic of non-dominated point archive management is covered here as being important to the implementation of multi-objective optimization methods. Existing Multi-Objective Particle Swarm Optimization methods are covered in this chapter. Careful research has been done so ,as to include the methods which have been described in the scientific literature. Test problems for multi-objective optimization methods are covered further. The list of problems given here contains test functions developed by others to test various aspects of the multi-objective optimizer performance. An overview of

existing performance indicators is given later. Performance indicators are used to assess the quality of the Pareto frontier approximations that are found by multi-objective optimization methods.

In the third chapter, an experimental analysis of existing Multi-Objective Particle Swarm Optimization methods is presented. The test problems described in the previous chapter have been used in conjunction with the performance indicators described there to evaluate the performance of these methods and look for patterns in terms of underlying implementation details of the methods and the types of problems they are particularly good (or bad) at solving. The findings of these experiments are presented in this chapter as well.

The fourth chapter is dedicated to the methods and suggestions proposed by the author. They are all related to the topic of evaluating the multi-objective optimization method performance. At first, an attempt at a novel classification of existing Multi-Objective Optimization Methods using their implementation details is made. Issues with existing performance indicators are explained with regards to measuring the uniformity of Pareto frontier coverage. Two new performance indicators are proposed that aim to solve these problems. Two new Multi-Objective Particle Swarm Optimization methods are described. They are based on the concept of using several different types of particles in the same swarm. These particles share the information via the non-dominated point archive. The methods are then compared to popular existing MOPSO methods. Results of experiments done to compare the methods are given and the conclusions are drawn. The methods compare favorably to existing approaches. Finally, a publicly available open-source software framework that has been developed as a part of the thesis is described in broad terms.

Chapter 2

Multi-Objective Particle Swarm Optimization

The problem of optimizing a real-valued function, that is finding either it's global minimum or global maximum (depending on the problem) is important and well studied. Real world problems can often be formulated as the problem of real-valued function optimization. We call $\boldsymbol{x}^* \in \mathbb{X}$, where $\mathbb{X} \subseteq \mathbb{R}^d$ functions' $f : \mathbb{X} \to \mathbb{R}$ global minimum if and only if $\forall \boldsymbol{x} \in \mathbb{X} : f(\boldsymbol{x}^*) \leq f(\boldsymbol{x})$. Functions global maximum is defined similarly with the comparison changed to $\geq$ in the previous formula. In the above, $\mathbb{X}$ is called the solution space, d is called the dimensionality of the problem and is the number of dimensions of the solution space. It is important to say that the problem of maximizing the function f is the same as minimizing the function $-1 \times f$ therefore one can restrict themselves to considering only the problem of minimization or only the problem of maximization without the loss of generality. We will consider only minimization of functions but everything said here works for maximization with small changes as well.

A lot of real world problems, however, involve minimizing several objectives at once. We will use function, criterium and objective interchangeably. These objectives are often contradictory. For example, decreasing the weight of an item will often decrease it's sturdiness and increase it's price. In a lot of problems it quickly becomes obvious that no single solution $\boldsymbol{x}^*$ exists like it does in the case of single objective optimization that will satisfy all the objectives. As such a

solution in this case is not a single vector, but a set of vectors called the Pareto set. This set has to satisfy certain properties that will be discussed in an appropriate section of this chapter. Since this set can be continuous and we cannot usually get it's explicit form via any exact methods the multi-objective optimization algorithms are designed to return an approximation of this set. It is up to the user then to select one solution from the set depending on what their priorities are at the moment. The points in these sets are expected to lay on the real Pareto front and to cover it uniformly, so that the user gets a clear idea what the actual Pareto frontier looks like for their problem.

In this chapter we present a method that is becomming more widely used to solve multi-objective optimization problems, namely Particle Swarm Optimization (PSO). We discuss this method's origins and theoretical properties, as well as how it is used to solve multi-objective optimization problems. We provide the definitions of the basic multi-objective optimization concepts, such as Pareto dominance, Pareto fronts and non-dominated sets. We also give an introduction on how several issues arising from using PSO for multi-objective problems are solved. These include stagnation to which one solution is the introduction of mutation operators and how to deal with the boundary constraints. We also discuss how non-dominated point archives, which contain solutions to multi-objective problems, are maintained.

2.1 Multi-Objective Optimization

Here we will define some general terms needed to understand the problem of Multi-Objective Optimization. As well as the concepts behind algorithms most often used to solve such problems in specific cases.

Definition 2.1.1. (Multi-Objective Optimization Problem)

A multi-objective optimization problem is the problem of minimizing several functions at once. The problem is that these functions may be contradictory (solution that gives low values for one function may give large values for other functions) standard terminology of local and global optima does not work in

this case. We define a multi-objective problem with k objectives and d decision variables below.

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{f}(\boldsymbol{x}) = (f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_k(\boldsymbol{x})) \\ \boldsymbol{x} &= (x_1, x_2, \ldots, x_d) \in \mathbb{X} \\ \boldsymbol{y} &= (y_1, y_2, \ldots, y_k) \in \mathbb{Y} \end{aligned} \tag{2.1}$$

Further on, we will call $\boldsymbol{x}$ a decision vector, $\boldsymbol{y}$ an objective vector, $\mathbb{X}$ will be called decision space and $\mathbb{Y}$ objective space. The problem is to minimize $\boldsymbol{f}(\boldsymbol{x})$ with respect to all objectives $f_1, f_2, \ldots, f_k$. Constraints may be added to the problem as in the single objective case. We will not be examining the constrained cases since the MOPSO methods surveyed and proposed in this work do not handle constraints. □

Definition 2.1.2. For any two objective vectors $\boldsymbol{u}$ and $\boldsymbol{v}$ we define the following operators.

$$\begin{aligned} \boldsymbol{u} = \boldsymbol{v} &\iff \forall i \in \{1, \ldots, k\} : u_i = v_i \\ \boldsymbol{u} \leq \boldsymbol{v} &\iff \forall i \in \{1, \ldots, k\} : u_i \leq v_i \\ \boldsymbol{u} < \boldsymbol{v} &\iff (\boldsymbol{u} \leq \boldsymbol{v}) \wedge (\boldsymbol{u} \neq \boldsymbol{v}) \end{aligned} \tag{2.2}$$

Operators $>$ and $\geq$ are defined similarly. □

Definition 2.1.3. (Dominance) For any two decision vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ we define the following relationships.

$$\begin{aligned} \boldsymbol{a} \prec \boldsymbol{b} \quad (\boldsymbol{a} \text{ dominates } \boldsymbol{b}) &\iff \boldsymbol{f}(\boldsymbol{a}) < \boldsymbol{f}(\boldsymbol{b}) \\ \boldsymbol{a} \preceq \boldsymbol{b} \quad (\boldsymbol{a} \text{ weakly dominates } \boldsymbol{b}) &\iff \boldsymbol{f}(\boldsymbol{a}) \leq \boldsymbol{f}(\boldsymbol{b}) \end{aligned} \tag{2.3}$$

Dominance is used to determine if one decision vector is "better" than another. Since simple arithmetic comparison of the function values does not work we need to extend it for the multi-objective problems. For single objective problems, we simply determine if the value of the function we are optimizing is lower for one decision than another. If said in simple words, the concept of dominance is this: decision vector $\boldsymbol{a}$ dominates decision vector $\boldsymbol{b}$ if all coordinates of $\boldsymbol{f}(\boldsymbol{a})$ are not greater than any of the coordinates $\boldsymbol{f}(\boldsymbol{b})$ and at least one coordinate of $\boldsymbol{f}(\boldsymbol{a})$ is smaller than the coresponding coordinate of $\boldsymbol{f}(\boldsymbol{b})$. □

Definition 2.1.4. (Pareto Optimality) A decision vector $\boldsymbol{x} \in \mathbb{X}_f$ is said to be nondominated regarding a set $\mathbb{A} \subseteq \mathbb{X}_f$ if an only if

$$\nexists \boldsymbol{a} \in \mathbb{A} : \boldsymbol{x} \prec \boldsymbol{a} \tag{2.4}$$

Similarly decision vector $\boldsymbol{x}$ is said to be Pareto optimal if and only if $\boldsymbol{x}$ is nondominated regarding $\mathbb{X}_f$. □

Definition 2.1.5. (Nondominated Sets and Frontiers For any set of decision vectors $\mathbb{A} \subseteq \mathbb{X}_f$ we define set $p(\mathbb{A})$ in the following way

$$p(\mathbb{A}) = \{\boldsymbol{a} \in \mathbb{A} \mid \boldsymbol{a} \text{ is nondominated regarding } \mathbb{A}\} \tag{2.5}$$

We call $p(\mathbb{A})$ a nondominated set with regards to $\mathbb{A}$, the corresponding set of objectives $\{\boldsymbol{f}(\boldsymbol{a}) \mid \boldsymbol{a} \in p(\mathbb{A})\}$ is called the nondominated frontier. Similarly we will call the set $p(\mathbb{X}_f)$ a Pareto-optimal set and it's corresponding objective vector set a Pareto-optimal frontier. □

Figure 2.1 shows a set of points in two dimensional objective space. The objectives are called f_1 and f_2. Points that are mutually non-dominated are emphasized. If the points were to correspond to all feasible solutions, the emphasized points would constitute the Pareto frontier of this multi-objective problem.

2.2 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a global optimization metaheuristic designed for the continuous problems. While heuristics are the approaches to problem solving that do not guarantee to be optimal or perfect, metaheuristics are heuristics that do not make assumptions about what kind of problems are to be solved. Evolutionary Algorithms are a commonly used metaheuristic. Metaheuristics are useful when little is known about the problem being solved. This commonly happens to the real-world problems. PSO was first proposed

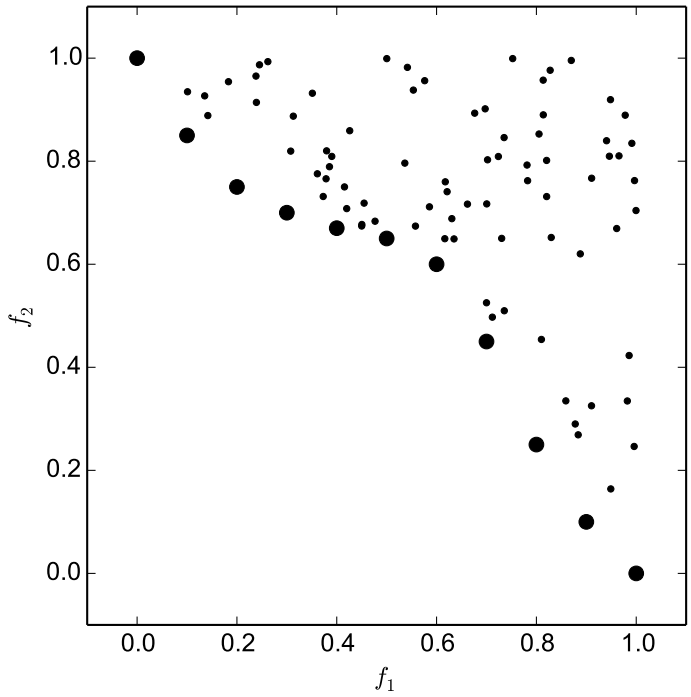


Figure 2.1: A set of points in two dimensions with mutually non-dominated points emphasized.

by James Kennedy and Russell C. Eberhart in 1995 [27]. The idea is to have a swarm of particles (points in multi-dimensional space) each having some other particles as neighbours and exchanging the information to find the optimal solutions. Particles move in the solution space of some function by adjusting their velocities to move towards the best solutions they have found, so far, and towards the best solutions found by their neighbours. These two attractors are further randomly weighted to allow more diversity in the search process. The idea behind this algorithm are the observations from the societies acting in nature. For example, one can imagine a flock of birds looking for food by flying towards other birds which are signaling a potential food source, as well as, by remembering where this particular bird itself has seen the food before and scouting areas nearby. Parts of it can also be viewed as modelling the way we ourselves solve problems - by imitating people we know, whom we see as particularly successful, but also by learning on our own. Thus, problem solving is influenced by our own experience and by the experience of the people we

know who can solve similar problems particularly well.

2.2.1 Original PSO Algorithm

This method is described in detail by J. Kennedy et al. [27]. In the original work on PSO the authors describe the thought process behind the development of the original PSO algorithm. The authors build their work on the work of C. W. Reynolds [54] which is connected to computer graphics and is meant to create convincing graphical simulations of the swarm behaviour so, that each individual's behaviour does not have to be scripted by hand. Another influence is the work of the zoologist F. Heppner et al. [20] who is concerned about modelling bird flocks and animal herds. Both of these studies were connected with describing local processes that result in the swarm behavior. This is in a way similar to other areas of research, such as that in to cellular automata (CA), where simple local rules gives a rise to the complex emergent behaviour. We see this in swarms as well — simple (we assume) behaviour of the individuals results in the complex global behavior of the swarm. We can see this in bee swarms or ant hills. Bees individually would not be able to collect food efficiently. Only through communication and the large population size they are able to function as successfully as they are. There is a belief in the research community that swarms represent an example of the final result being more than a sum of it's parts. That is the success of the social systems, which seems to be more complicated than one would expect merely through the multiplication of the simple local behaviour of the individuals. Each individual profits from being in the swarm since it shares the success of the population. Authors also extend the metaphor of swarming to the human behaviour. It is important to note the differences though — fish school and bird flock observed behaviour is physical. They use the advantages of swarming to evade predators, find food and potential mating partners. While human experience is also abstract — humans change their cognitive and experiential variables with accordance to their experience and the experience of people they are influenced by. The additional (if minor) difference is that there is no collision in the case of human opinions — while two people can have the same opinion and beliefs, two birds cannot occupy the same physical space. The metaphor behind the method is

basically this — we, when looking for solutions to problems, look for them on our own by modifying the solutions we already know, we also are influenced by successful solutions of others. Another way to look at it is to imagine a flock of birds looking for food. One bird would look around the areas where it had found the food previously, it would also take into account the information its closest neighbours signal to it about where they had previously found large amounts of food.

Suppose we have n particles in the swarm. We introduce two vectors $\boldsymbol{p}_i$ which is the best solution found by particle with index $i \in \{1, \ldots, n\}$ so far.The best solution is the one that gives the lowest value of the function being minimized. This vector is updated after each iteration of the algorithm if particle i finds a better solution. Another solution that is stored is $\boldsymbol{g}_i$ which is the best solution found by the neighbours of particle i. Who are the particle's neighbours can be determined in various ways, such as set in advance. The particles "fly" through the solution space. Their direction is then calculated in such a way that the particle would move towards it's own best personal solution $\boldsymbol{p}$ and towards it's neighbour's best solution $\boldsymbol{g}_i$. This is further randomly weighted so, that the particle explores the area around those two solutions better.

$$\boldsymbol{v}_i \leftarrow w\boldsymbol{v}_i + \rho_1 \boldsymbol{U}_{(0,1)}(\boldsymbol{p}_i - \boldsymbol{x}_i) + \rho_2 \boldsymbol{U}_{(0,1)}(\boldsymbol{g}_i - \boldsymbol{x}_i) \tag{2.6}$$

This is shown in Equation (2.6). Here the velocity vector $\boldsymbol{v}_i$ is updated so that during each iteration it moves towards personal best and neighbour's best solutions. It is important to note that the relation is recurrent. It can also be prone to unstable behavior. For example, explosion of the values of the velocity vector. Values of velocity vector $\boldsymbol{v}_i$ are clipped to be in some prespecified range. If the velocity in any coordinate exceeds a value v_{max} or goes below value $-v_{max}$ it is set to v_{max} or $-v_{max}$ accordingly. Algorithm parameter v_{max} thus sets the maximum velocity for every coordinate. Usually maximum velocity is the same for every coordinate of the solution space.

Parameters ρ_1 and ρ_2 are set to 2. The reasoning behind this is that with that value the particles will overfly the target on average half of the time and thus explore the area around the potential good solutions better. Parameter w is

the damping or "inertia" parameter. Low values of this parameter mean that particles will react to changes in $\boldsymbol{p}_i$ and $\boldsymbol{g}_i$ faster. Large values of this parameter mean that there is a lot of inertia in this regard. Common values range from zero to one.

Another important point to consider is how $\boldsymbol{g}_i$ is chosen. Personal best $\boldsymbol{p}_i$ is simply the best solution particle with index i has discovered so far. Global or neighbourhood best $\boldsymbol{g}_i$ is the best solution found by particle's with index i neighbours. This requires one to know who those neighbours are. This is often simply all the other particles of the swarm. As such $\boldsymbol{g}_i$ is the best solution found by the swarm as a whole. This is not always the case. Particles can be arranged in various topologies where neighbourhood relationships are determined by edges of the graph where particles are vertices. These neighbourhood configurations or topologies have an important influence on the performance of the swarm as discussed below.

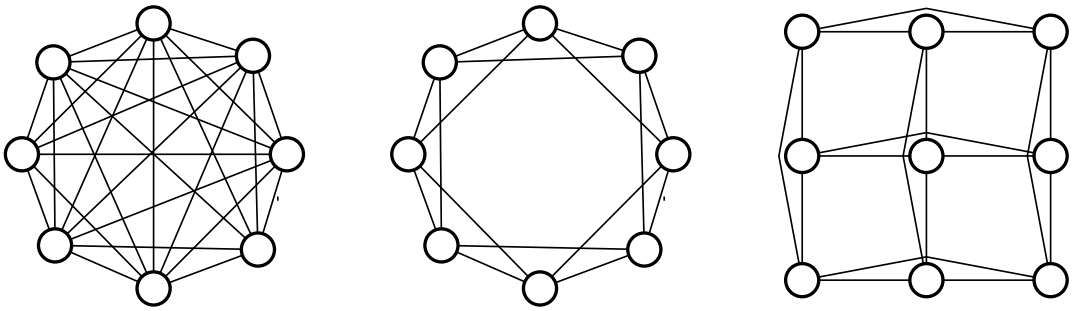
Having calculated the velocity it is simply added to the position vector $\boldsymbol{x}_i$. This is then repeated until some stopping criteria are met. Most often the algorithm is simply run for a fixed number of iterations of velocity and position updates. Initially position is most often set to randomly chosen points from the solution space while velocity vector is set to zero. The original PSO algorithm is presented in Algorithm 1.

Algorithm 1 Original PSO algorithm.

1: **for** $i \leftarrow 1, n$ **do**
2: initialize $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$
3: $\boldsymbol{p}_i \leftarrow \boldsymbol{x}_i$
4: **end for**
5: **while** stop conditions not satisfied **do**
6: $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i$
7: update $\boldsymbol{p}_i$ and $\boldsymbol{g}_i$
8: $\boldsymbol{v}_i \leftarrow w\boldsymbol{v}_i + \rho_1 \boldsymbol{U}_{(0,1)}(\boldsymbol{p}_i - \boldsymbol{x}_i) + \rho_2 \boldsymbol{U}_{(0,1)}(\boldsymbol{g}_i - \boldsymbol{x}_i)$
9: **end while**

2.2.2 PSO Topology

PSO topology is used to refer to the graph that is formed from particles as vertices and neighbourhood relations as edges. If two particles are connected by an edge they will share information. This information will then be used when choosing $\boldsymbol{g}_i$ — the global best solution for the particle with index i. When updating $\boldsymbol{g}_i$ all the neighbours are examined and the neighbour j whose $\boldsymbol{p}_j$ gives the lowest value of the function being optimized is chosen as the leader. After this the leader's $\boldsymbol{p}_j$ is set as $\boldsymbol{g}_i$. It was shown numerous times that particle swarm topology has a profound influence on the performance of PSO. This is, however, very often ignored and a simple topology where each particle is connected to every other is chosen. It is often a suboptimal choice. Popular topologies are given in Figure 2.2. Besides a simple "everyone is a neighbour of everyone else" topology or "gbest" topology shown in Figure 2.2a it is also common to connect particles in a ring where each particle is connected to some number of particles to the left and to the right as shown in Figure 2.2b. This is known as the "lbest" topology sometimes. Other configurations are possible, for example the "grid" topology where particles are aranged in a grid and connected to particles at the top, bottom and the sides. Two early studies on the influence of topology to PSO performance were performed by J. Kennedy [26] [28]. In them the author examines various different topologies when optimizing different test problems. They also include the procedure for randomly generating new topologies. The results indicate that popular topologies like "gbest" and "lbest" do not necessarily result in good performance. In topologies like "gbest" information is exchanged very quickly since each particle has instance access to any good solution found by the swarm. This could result in premature convergence to local minima. On the other hand, in topologies like "lbest" it can take a long time until good solutions propagate. This can mean that premature convergence is less alike. It is noted in the study that "grid" topology results in good performance despite it's simplicity and the authors recommend its use.



(a) "gbest" topology (b) "lbest" topology (c) "grid" topology

Figure 2.2: Three popular PSO topologies.

2.2.3 Canonical Particle Swarm Optimization

Canonical PSO was proposed by Maurice Clerc et al. [4] and is a variant of the original PSO algorithm. It guarantees convergence through the use of the constricting factor χ. It also has the advantage of not having any parameters, except for population size and ρ_1, ρ_2. Parameters ρ_1 and ρ_2 represent the influence of the personal best solution and the best solution of particles neighbours on the trajectory of that particle. Both of these parameters are usually set to 2.05 as per suggestion in the original paper. Moving the particle in the solution space is done by adding the velocity vector to the old position vector as illustrated in Equation (2.7).

$$\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i \tag{2.7}$$

Updating velocity involves taking current velocity and adjusting it so, that it will point the particle more in the direction of its personal best and the best of its most successful neighbour. It is layed out in Equation (2.8).

$$\boldsymbol{v}_i \leftarrow \chi \left(\boldsymbol{v}_i + \rho_1 \boldsymbol{U}_{(0,1)}(\boldsymbol{p}_i - \boldsymbol{x}_i) + \rho_2 \boldsymbol{U}_{(0,1)}(\boldsymbol{g}_i - \boldsymbol{x}_i) \right) \tag{2.8}$$

where

$$\chi = \frac{2}{\rho - 2 + \sqrt{\rho^2 - 4\rho}} \tag{2.9}$$

and where $\rho = \rho_1 + \rho_2$. Usually ρ_1 and ρ_2 are set to 2.05, however, the method works as long as $\rho_1 + \rho_2 > 4$. $\boldsymbol{U}(a, b)$ is a vector of random numbers from the uniform distribution ranging from a to b in value. Here $\boldsymbol{p}_i$ is the best personal solution of particle i and $\boldsymbol{g_i}$ is the solution found by a neighbour of particle i. Which particle is a neighbour of which other particle is set in advance.

2.2.4 Fully-Informed Particle Swarm Optimization

The original Fully-Informed PSO algorithm was described by Rui Mendes et al. [38] and the original procedure for velocity update is given in (2.10) formula. The difference between standard PSO and Fully-Informed PSO is that the velocity update formula takes into the account all of particles neighbours, instead of only the one with the best solution found so far.

$$\boldsymbol{v}_i \leftarrow \chi \left(\boldsymbol{v}_i + \phi \left(\boldsymbol{P}_i - \boldsymbol{x_i} \right) \right) \tag{2.10}$$

where

$$\boldsymbol{P}_i = \frac{\sum_{k \in N_i} c_k \boldsymbol{\rho}_k \boldsymbol{p}_k}{\sum_{k \in N_i} c_k \boldsymbol{\rho}_k} \tag{2.11}$$

where

$$\boldsymbol{\rho}_k = \boldsymbol{U} \left(0, \frac{\rho_{max}}{|N_i|} \right) \tag{2.12}$$

and where N is the set of particle's neighbour indices. Particle's neighbourhood topology is set up in advance and is an algorithm parameter. As in the canonical PSO algorithm, particle's position in the solution space is updated by adding

velocity vector to position vector $\boldsymbol{x}$. This variant extends the work done by Maurice Clerc et al. [4] on the value of χ - the constriction coefficient in formula (2.10), which is essential to swarm convergence. This work has been done on a particle swarm variant that only takes into account its own personal best and the personal best of its most successful neighbour, but it has been extended to use the data from all the neighbours.

Riccardo Poli et al. [51] give a slightly different update rule, where $w_k = 1$, it is shown in (2.13) formula. It is simply a special case of the method described above.

$$\boldsymbol{v}_i \leftarrow \chi \left(\frac{1}{N_i} \sum_{k \in N_i} \rho \boldsymbol{U}_{(0,1)} (\boldsymbol{p}_k - \boldsymbol{x}_i) \right) \tag{2.13}$$

Here, N_i is the set of neighbour indices of particle with index i. Coefficient χ is calculated the same way as for the canonical particle swarm. For the calculation to work $\rho > 4$ is required.

2.2.5 Bare Bones Particle Swarm

First proposed by James Kennedy [29] it is a very simple PSO variant. It is memory-less in the sense that the position vector is updated directly instead of by adding the velocity, as shown in (2.14) formula.

$$\boldsymbol{x}_j \leftarrow \boldsymbol{v}_j \tag{2.14}$$

Velocity (or in this case position) is updated using the (2.15) formula.

$$\boldsymbol{v}_j \leftarrow \boldsymbol{N} \left(\rho_1 \frac{\boldsymbol{p}_j + \boldsymbol{g}_j}{2}, \rho_2 |\boldsymbol{p}_j - \boldsymbol{g}_j| \right) \tag{2.15}$$

Here, $\boldsymbol{N}(\boldsymbol{\mu}, \boldsymbol{\sigma})$ is a vector with random numbers from the normal distribution with means and standard deviations taken from the vectors passed as arguments. Each coordinate of the argument vectors corresponds to the mean and standard

deviation for that coordinate in the resulting vector. Other than the changed rules this algorithm is implemented identically to the canonical one. This variants' performance is claimed to be comparable to the canonical algorithm but it is considerably simpler to both implement and understand.

2.3 PSO Mutation Operators

It is generally accepted that PSO converges very fast. For example, in the study done by Jakob Vesterstrom et al. [65], where they compare the performance of Differential Evolution, Particle Swarm Optimization and Evolutionary Algorithms they conclude that PSO always converges the fastest of the examined algorithms. In practice this is a double-edged sword – fast convergence is obviously attractive in an optimization algorithm, however, it is possible that it can lead the algorithm to stagnate after finding a local minimum. There are several strategies to slow down convergence and, thus, increase the amount of time that the algorithm spends in the initial exploratory stage as opposed to local search indicative of later stages of PSO operation. One solution is to use different swarm topologies since it has been shown that using a different topology can affect the swarm operation in terms of convergence speed and allow to adjust the trade-off between exploration and exploitation. Exploration here means covering as large a volume of the search space as possible. Exploitation means focusing on promising regions of the search space. See, for example, a paper by James Kennedy [30] or James Kennedy and Rui Mendes [31]. Another attempt to solve this,is to change the velocity update formula to use an inertia coefficient w that the speed it multiplied by during each iteration, see, for example, the works by Russell C. Eberhart and Yuhui Shi [14] or by Yuhui Shi and Russell C. Eberhart [59].

The third approach is to introduce a mutation operator. A mutation operator is used to modify particle positions or velocities outside the position and velocity update rules. In all of the cases examined here mutation is applied after position and velocity updates and only to particle positions. Each coordinate of each particle has a certain probability of being mutated. The probability can be calculated from Equation (2.16) if mutation $rate$ is provided. Parameter $rate$

means how many dimensions of the particle's current position will be mutated during each algorithm iteration. For example, if $rate = 1$ one dimension of one particle in the swarm will be mutated on average during each iteration.

$$probability = \frac{rate}{particles \times dimensions} \tag{2.16}$$

Five different mutation operators that are found in literature are examined here. The first one given in Equation (2.17) simply reinitializes a single dimension of a particle to a uniformly distributed random value $U_{(a_j,b_j)}$ from the permissible range. It is used to test whether it is useful to rely on the previous value x_{ij} or not, it is the only of the operators that does not rely on it. It can be found, for example, in an overview of mutation operators by Paul S. Andrews [3]. Here, and below, x_{ij} is the j-th coordinate of the particle's with index i position vector $\boldsymbol{x}_i$.

$$x_{ij} \leftarrow U_{(a_j,b_j)} \tag{2.17}$$

Another operator, also proposed by Paul S. Andrews [3] is based on the Gaussian distribution and given in Equation (2.18).

$$x_{ij} \leftarrow x_{ij} + N(0, \sigma) \tag{2.18}$$

Another operator based on the Gaussian distribution is given in Equation (2.19) and can be found in a work by Natsuki Higashi et al. [21].

$$x_{ij} \leftarrow x_{ij}(1 + N(0, \sigma)) \tag{2.19}$$

A similar operator to the one given in Equation (2.18) is given in Equation (2.20), the only difference is that this one is based on the Cauchy distribution. It is presented in a work by Andrew Stacey et al. [61].

$$x_{ij} \leftarrow x_{ij} + cauchy(a) \tag{2.20}$$

A different kind of mutation operator proposed by Zbigniew Michalewitz [39]. It was proposed for use in PSO by Susana C Esquivel et al. [17]. While the original operator changes its behaviour with regards to algorithm iteration, we used a static version to keep it in line with the other operators. It is given in Equation (2.21), where $flip$ is a random value in the range $(0, 1)$, generated before applying the operator.

$$x_{id} \leftarrow \begin{cases} x_{ij} + (b_j - x_{ij})U_{(0,1)} & \text{, if } flip < 0.5 \\ x_{ij} - (x_{ij} - a_j)U_{(0,1)} & \text{, if } flip \geq 0.5 \end{cases} \tag{2.21}$$

Here $a = \sigma = 0.1(b_j - a_j)$, where a_j is the lower bound for coordinate j and b_j is the upper bound. Mutation operators are applied to the particle's position after the particle has completed it's position and velocity updates. This moves the particle to a new, randomized position, possibly dependant on the particles previous position.

2.4 PSO Behavior and Convergence

An early analysis of PSO convergence behavior was performed by E. Ozcan et al. [46]. In it the authors perform the analysis of a trajectory of a single particle in a simplified swarm. This analysis is built on the author's previous work [45]. It is claimed that the particles, instead of "flying" in the search space "surf" it on sine waves instead. When seeking for the optimal solutions particles manipulate the waves frequency and amplitude at random. The velocity limiting parameter v_{max} seems to allow the particle to jump between the waves easier.

J. Kennedy [29] gets rid of the velocity update rule proposing a bare-bones particle swarm optimization which has more in common with random search methods. The reasoning behind this is that particles follow a cyclic-path centered around a randomly-weighted mean of the individual's and it's neighbours' best points. The author uses these observations to simplify the PSO method so, that particle picks it's position from Gaussian distribution with a mean that is the average of $\boldsymbol{p}_i$ and $\boldsymbol{g}_i$ (a midpoint between these two points) and the standard

variation is the distance between these two points. This method gives the results that are comparable if slightly worse to the canonical PSO and thus, gives support to these assumptions about PSO behavior. It is important to note that in this case the social aspects such as neighbourhood topologies are still maintained and constitute an important difference between bare-bones PSO and random search methods. One important outcomes of this is that the simplified PSO is easier to analyze using the techniques that have already been applied, for example, when analyzing the behavior and properties of simulated annealling and other random search methods.

R. Poli et al. [50] built on this work and used Markov chains to analyze the behavior of bare-bones PSO. Authors state that with some simple changes to the sampling distribution used, it is possible to make bare-bones PSO a global optimizer at least for the problems they have analyzed.

One of the first analyses of PSO behavior in terms of it's stability and convergence properties was performed by M. Clerc et al. [4]. The authors analyze PSO in a simplified form as a dynamic system. The simplifications involve getting rid of the random coefficients and simplifying velocity update rules. The results, however, apply in the general case as well. The algorithm is analyzed from the inside — the point of view of the single particle. The single particle deals with two attraction points — global and personal best solutions. The authors propose to use a convergence coefficient χ to force the PSO algorithm to converge and prevent explosive behavior. The coefficient is calculated as given in Equation (2.22). With ρ being the sum of the personal and global influence coefficients. This is the basis of what some refer to as the Canonical PSO algorithm which has been discussed in the previous section.

$$\chi = \frac{2}{\rho - 2 + \sqrt{\rho^2 - 4\rho}} \tag{2.22}$$

2.5 Traditional Approaches to Multi-Objective Optimization

One of the traditional approaches to multi-objective optimization is the weighted aggregation approach. The values of objectives, given as a solution, are multiplied by their respective weights and aggregated. There are several types of aggregation, the simplest and very common is simply taking the sum of all objectives. The weights, in this case, have to add up to one. One optimization run will give a single non-dominated point. Weights are usually chosen not by the decision maker, but algorithmically. They are usually picked so, that the whole Pareto frontier is uniformly covered. This procedure is independent on the optimization method used. The resulting weighted aggregate function can be optimized using any single objective global optimization method suitable for the resulting single objective function. The resultant function to be minimized is given in Equation (2.23), subject to $\boldsymbol{x} \in \mathbb{X}$ with $\mathbb{X}$ being feasible solution space.

$$f(\boldsymbol{x}) = w_1 f_1(\boldsymbol{x}) + w_2 f_2(\boldsymbol{x}) + \ldots + f_k(\boldsymbol{x}) \tag{2.23}$$

Values w_i are called weights, are positive and usually normalized to add up to one, that is $\sum_{i=1}^{k} w_i = 1$. It is possible and easy to show that provided that a global optimization method is used (one that is guaranteed always to find the global minimum) and weights are positive, this method will only generate Pareto optimal solutions. This is easy to prove by contradiction. Suppose a solution $\boldsymbol{a}$ minimizes function $\boldsymbol{f}$ for a given weight combination but is Pareto dominated by another solution $\boldsymbol{b}$. In that case, without loss of generality, $f_1(\boldsymbol{b}) < f_2(\boldsymbol{a})$ and $f_i(\boldsymbol{b}) \leq f_i(\boldsymbol{a})$ for $i = 2, \ldots, k$. This means that $f(\boldsymbol{a}) > f(\boldsymbol{b})$ (since the values of the weight vector are all positive) which contradicts our assumption that $\boldsymbol{a}$ minimizes function f.

A serious drawback of weighted aggregation methods is that they cannot find points on non-convex surfaces. The reason for this will hopefully become apparent, if we consider the following two objective case. For example, consider the case of minimizing $y = w_1 f_1(\boldsymbol{x}) + w_2 f_2(\boldsymbol{x})$ which can be rewritten as $f_2(\boldsymbol{x}) = -\frac{w_1}{w_2} f_1(\boldsymbol{x}) + \frac{y}{w_2}$. This constitutes, in objective space, a line with slope $-\frac{w_1}{w_2}$ and

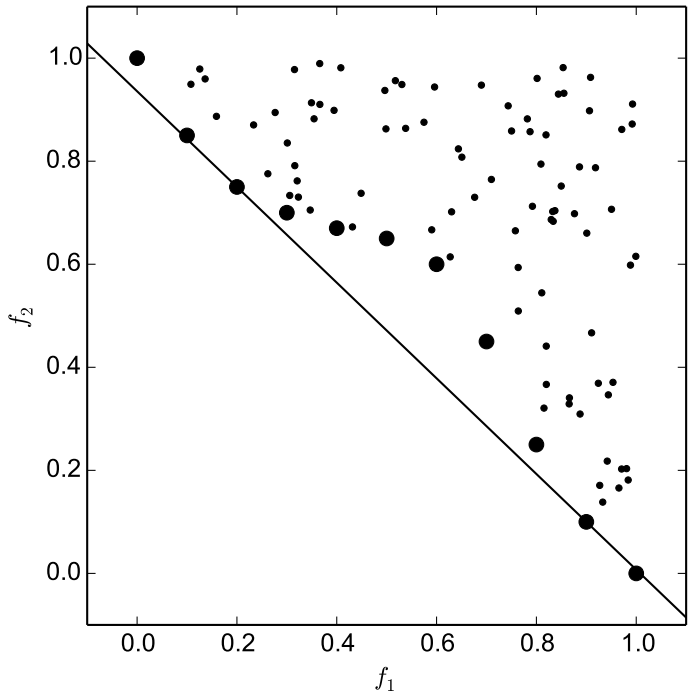


Figure 2.3: The lines with various slopes $-\frac{w_1}{w_2}$ will be moved up and down during the optimization process.

offset $\frac{y}{w_2}$. The optimization process will move this line up and down until none of the Pareto frontier points lies below it and at least one point is on it. This is illustrated graphically in Figure 2.3. It can be seen that no matter what slope we choose, we cannot find a line that would touch a point in the non-convex region in such a way that none of the other points were below it. In the case of a non-convex surface, the points on that surface will not be found by this method.

Another way of solving multi-objective problems is to convert $k - 1$ of the k objectives into inequality constraints. The remaining objective is a single objective problem that is then optimized using constrained single objective optimization methods. The problem here is that it is not easy to design the constraints appropriately. Aggregate and constraint methods were described by J. L. Cohon [8] [7]. Other traditional approaches include goal programming described by R. E. Steuer [62] and the Min Max approach as described by J. Koski [32].

2.6 Pareto Point Archive Maintenance

It is possible to construct Pareto frontier approximations by using single objective optimization algorithms. That can be done by constructing a weighted aggregate function from the objectives and optimizing the resulting single objective function. The result of such optimization is a single Pareto point. We can get different Pareto points along the Pareto frontier, if we change the weights of the aggregate function. However, it is often desireable to collect the whole Pareto frontier approximation in a single run of the algorithm. Such an approximation is often maintained by an archive maintenance algorithm that is designed to collect points while ensuring that all points in the archive are non-dominated. Pareto point archives can be un-bounded. This means taking in as many non-dominated points as are provided. They can also be bounded, in the case when the archive is full some criteria must be used to determine which point, if any, to remove from the archive to accomodate the new one. These criteria are often designed in such a way so as to ensure uniform coverage of the Pareto frontier. This means that points are prefered if they increase the uniformity of the archive. Uniformity is usually measured as deviation in Euclidean distance between the closest points in the archive or using some other metric. In Algorithm 2 we give a rough outline of one such algorithm.

Algorithm 2 Pareto point archive management. General algorithm.

```
1:  function UPDATEARCHIVE(archive, candidate)
2:      for solution ∈ archive do
3:          if dominates(solution, candidate) then
4:              return False
5:          end if
6:          if dominates(candidate, solution) then
7:              remove(archive, solution)
8:          end if
9:      end for
10:     append(archive, solution)
11:     return True
12: end function
```

Algorithm 2 describes the function that takes a Pareto point archive, a candidate solution (here a solution is some object that contains the pareto point and maybe other data) and tries to update the archive. If the candidate solution is rejected,

the algorithm returns **False**, otherwise it returns **True**. It depends on three other procedures, namely $dominates$, $remove$ and $append$. The first one checks if one solution dominates another. The second one removes the given solution from the archive. The third one will append the solution to the archive. The third procedure is particularly important. It has to decide what to do when a new solution is proposed and the archive is bounded and already full. It will have to check if the solution improves some metric that is used to decide if a solution improves the uniformity of the archive in such cases.

2.6.1 Crowding Distance

Crowding distance is one of the core concepts behind the popular NSGA-II algorithm developed by Kalyanmoy Deb et al. [12]. Each solution in a non-dominated point archive is assigned a crowding distance. Below is an outline of the algorithm to calculate this distance. If our archive A consists of N solutions $\boldsymbol{a}_1, \ldots, \boldsymbol{a}_N$ then we can define crowding distance for solution i as described in Algorithm 3.

Algorithm 3 Calculating crowding distance d_i for solution $\boldsymbol{a}_i$ in archive A.

$d_i \leftarrow 0$
for $j \in \{1, \ldots, k\}$ **do**
    $I \leftarrow sort_{f_j}(A)$
    if solution a_i is either the first or last solution in I **then**
        return ∞
    end if
    for $K \in \{1, \ldots, k\}$ **do**
        $d_i \leftarrow d_i + \frac{f_K(\boldsymbol{I}_{i-1}) - f_K(\boldsymbol{I}_{i+1})}{f_K^{max} - f_K^{min}}$
    end for
end for
return d_i

Here the number of objectives as elsewhere is k and we define I_{i-1} to be the solution that comes before the solution i after sorting the archive by one of the objectives and I_{i+1} to be the solution after i after sorting. The idea here is to sort the solutions in archive by each of the objectives and then to calculate Manhattan distance between the solution to the left and to the right (or smaller and larger in terms of that objective), scale it and then add them together to produce a

crowding distance. This is simple to explain the two objective case since it shows the meaning of neighbouring solutions clearly. This visual explanation is given in Figure 2.4 and here neighbouring solutions are called $\boldsymbol{y}_{i-1}$ and $\boldsymbol{y}_{i+1}$ and crowding distance in this case is the perimeter of the box shown in dashed line.

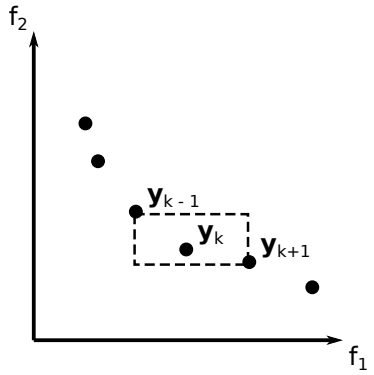


Figure 2.4: Crowding distance for two objective case.

2.6.2 ϵ-Dominance

The concept of ϵ-dominance is used to limit the size of non-dominated point archives. The dicussion of it can be found in work by Marco Laumanns et al. [33]. A decision vector $\boldsymbol{x}_1$ is said to ϵ-dominate another vector $\boldsymbol{x}_2$ (with $\boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{X}$) for some $\epsilon > 0$ if and only if the conditions in (2.24) are satisfied.

$$\begin{aligned} &\left(\forall i \in \{1, \ldots, k\} : \frac{f_i(\boldsymbol{x}_1)}{1+\epsilon} \leq f_i(\boldsymbol{x}_2)\right) \wedge \\ &\left(\exists i \in \{1, \ldots, k\} : \frac{f_i(\boldsymbol{x}_1)}{1+\epsilon} < f_i(\boldsymbol{x}_2)\right) \end{aligned} \tag{2.24}$$

Which is similar to the definitions of Pareto dominance but with the objective space vector being scaled by a scalar value. This concept is then applied to limit which solutions are accepted into the archive. Only those solutions that are not ϵ-dominated by any other point in the archive are accepted.

2.7 Overview of Existing MOPSO Methods

When designing multi-objective optimization algorithms the following problems have to be addressed for them to be effective:

1. Maximize the number of points in the approximation.
2. Minimize the distance between the approximation and the Pareto frontier.
3. Maximize the spread of solutions along the Pareto frontier. A related goal is to make sure that no areas of the Pareto frontier remain undiscovered in the approximation.

All these problems should also be kept in mind when designing quality indicators that measure the performance of multi-objective optimization algorithms. To put it more simply, a successful algorithm should give a good idea of what the Pareto frontier looks like. It is also desireable to obtain several non-dominated points with a single algorithm run. This is possible given a population based nature of PSO where each particle can be used to explore a different part of the Pareto frontier.

The three main questions that someone engineering a multi-objective PSO algorithm is going to have to solve are the following:

1. How to select personal best solution $\boldsymbol{p}_i$ for particle with index i and how to select the neighbour's best solution $\boldsymbol{g}_i$? These are needed if one hopes to use the velocity update formula of standard PSO.
2. How to store the nondominated solutions found during each iteration of the algorithm. Problems of computational efficiency and ensuring good spread of solutions should be addressed here as well. Nondominated point archives are often used for this.
3. How to ensure uniform spread of the solutions? How to make sure that no area of the Pareto frontier is missing in the approximation?

Personal best $\boldsymbol{p}_i$ is very often updated similar to how it's handled in standard single objective PSO. In this case dominance is used instead of checking if the new solution gives a lower value of the function being minimized. If the new solution dominates the old one $\boldsymbol{p}_i$ can be updated, for example. This is not the only posibility though. When selecting $\boldsymbol{g}_i$, it is often done in such a way that $\boldsymbol{g}_i$ corresponds to poorly explored areas of the real Pareto frontier. This is done either by using crowding distance, niching, hypervolume contribution or other methods. Nondominated point archives are often used to address problem number two and they are mostly no different than the ones used for multi objective evolutionary algorithms. Problem number three is mostly solved by picking global best in order to correspond to poorly covered areas of the Pareto frontier although other approaches also exist.

Margarita Reyes-Sierra et al. [53] provide an overview of multi-objective PSO algorithms available at the time.

In this chapter we provide summary of state of the art as well as order research in to using Particle swarm optimization algorithms to solve multi-objective problems. Most algorithms discussed in this section and indeed proposed in literature can be fit in to the template given in Algorithm 4 with the only differences between algorithms being how personal and global bests $\boldsymbol{p}_i$ and $\boldsymbol{g}_i$ are updated. Therefore where possible we will discuss algorithms in terms of how various parts of this template are filled in.

Algorithm 4 General multi-objective particle swarm algorithm.

for $i \leftarrow 1, n$ **do**
    initialize $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$
    $\boldsymbol{p}_i \leftarrow \boldsymbol{x}_i$
end for
while stop conditions not satisfied **do**
    update non-dominated point archive $\mathbf{A}$
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i$
    update $\boldsymbol{p}_i$ and $\boldsymbol{g}_i$
    $\boldsymbol{v}_i \leftarrow w\boldsymbol{v}_i + \rho_1 \boldsymbol{U}_{(0,1)}(\boldsymbol{p}_i - \boldsymbol{x}_i) + \rho_2 \boldsymbol{U}_{(0,1)}(\boldsymbol{g}_i - \boldsymbol{x}_i)$
end while

Let us briefly discuss what is going on in Algorithm 4. First of all for each particle i among n particles we initialize $\boldsymbol{x}_i$ and $\boldsymbol{v}_i$. Usually $\boldsymbol{x}_i$ is set to random values within permissible ranges for each coordinate so as to stay in the feasible

region while $\boldsymbol{v}_i$ is set to zero vector. This is done in the loop from line 1 to line 4 in the pseudo-code. Then follows the main loop of the MOPSO algorithm. First we update the non-dominated point archive $\mathbf{A}$. This is done before other updates so as to incorporate initial particle positions. After this we update particle position by adding the velocity vector to it as per standard PSO algorithm. Then we update personal best $\boldsymbol{p}_i$ and global best $\boldsymbol{g}_i$. These two updates usually are the main thing that is different between distinct multi-objective PSO algorithms. Usually updating $\boldsymbol{p}$ involves establishing dominance relationships between current $\boldsymbol{p}_i$ and new particle position $\boldsymbol{x}_i$. Personal best can be updated, for example, if new position strongly or weakly dominates old position. Global best is usually selected from the non-dominated point archive. Selection rules often try to emphasize exploring regions in objective space where there are fewer solution so as to uniformly cover the Pareto front. After $\boldsymbol{p}_i$ and $\boldsymbol{g}_i$ are selected velocity update rule from standard PSO is used. This is repeated until some criteria are met. Often a limit on the number of iterations is set and algorithm terminates once the main loop is repeated a fixed number of times.

2.7.1 C. A. C. Coello et al. (2002)

C. A. C. Coello et al. [6] propose a Pareto dominance based approach. The main idea is dividing the decision space in to a fixed number of hypercube with even sides. Basically this means covering the space in a hypergrid. The number of hybercubes is set by the user. The reason for this is estimating most crowded areas of the decision space in terms of how many solutions occupy those area. A solution in a hypercube is prefered when selecting global best if the hypercube contains fewer solutions. A solution in a full non-dominated point archive will be selected for replacement by a new solution if it lies within a hypercube that contains many solutions. This techniques is employed to an end similar to that of crowding distance measure used in other methods.

2.7.1.1 Changes to PSO update rules

The changes to PSO velocity and position update rules are given in the list below.

Parameter	Description	Value
D	Number of objective space division points	30 to 50
ρ_1	Personal experience influence coefficient	1.0
ρ_2	Social influence coefficient	1.0
w	Inertia coefficient	0.4

Table 2.1: Parameter values for C. A. C. Coello et al. (2002) method.

- Personal best solution $\boldsymbol{p}_i$ is updated if the new solution dominates the old personal best.
- Solution space is divided in to a fixed number of hypercube depending on the value of parameter D. There will be d^D hypercubes if the decision space is d-dimensional. For those hypercubes that contain more than one solution contained in the archive A assign a value of $\frac{10}{N}$ where N is the number of particles in the hypercube. Pick a hypercube where that value is the lowest. Select an archive solution that is contained within said hypercube at random. This solution will be the global best $\boldsymbol{g}_i$ for that iteration. All particles share the same global best.

2.7.1.2 Parameters

Parameters and their values as specified by the authors are summarized in Table 2.1. The personal and social influence coefficients and inertia coefficient are similar to those used in other MOPSO variants. Parameter D specified how many intervals should each dimension be divided in to. The number of hypercubes as used in the method to measure population diversity is D^m, where m is the number of objectives in a given multi-objective optimization problem.

Other than the above mentioned differences, the method followed the outline of our generic MOPSO algorithm.

2.7.2 X. Hu et al. (2002)

Proposed by Xiaohui Hu et al. [22] this algorithm uses what the authors call "dynamic neighbourhood". Only problems with two objectives are considered in the work. The basic idea behind the algorithm is to use the first objective to find the neighbours for a given particle and to use the second objective to select the best one among those neighbours. A new personal best solution for some particle replaces the old one only if the old one is Pareto dominated by the new one. The name "dynamic neighbourhood" comes from the fact that during each iteration of the PSO algorithm a particle has to recalculate who it's neighbours are.

2.7.2.1 Changes to PSO update rules

Equation (2.25) describes distance between particle i and particle j in terms of the value of the first objective. The algorithm requires, for each particle i in the swarm, to find m closest particles in terms of distance d_{ij}.

$$d_{ij} = |f_1(\boldsymbol{x}_i) - f_1(\boldsymbol{x}_j)| \tag{2.25}$$

The differences in finding global and personal best solutions are outlined in the list below. Other than that, velocity $\boldsymbol{v}_i$ and position $\boldsymbol{x}_i$ are updated as in classical single objective PSO.

1. Update personal best solution $\boldsymbol{p}_i$ only if the new solution Pareto dominates the old one.
2. Find m closest particles to particle i in terms of the first objective f_1. Among those m closest particles find the one which position gives the lowest value of the second objective f_2. If this particle has index j then global best $\boldsymbol{g}_i$ will be solution $\boldsymbol{x}_j$.

Parameter	Description	Value
m	Neighbourhood size	2
ρ_1	Personal experience influence coefficient	1.49445
ρ_2	Social influence coefficient	1.49445
w	Inertia coefficient	0.5 + $U_{0,0.5}$

Table 2.2: Parameter values for X. Hu et al. (2002) method.

2.7.2.2 Parameters

The authors used the following values for the swarm parameters $w = 0.5 + 0.5U_{(0,1)}$, $\rho_1 = \rho_2 = 1.49445$ and v_{max} was set to "dynamic range of the particle on each dimension". The number of neighbours selected using the first objectives is set to 2.

2.7.3 K. E. Parsopoulos et al. (2002)

K.E. Parsopoulos et al. [47] propose three different strategies for multi-objective optimization using particle swarm algorithms. The first two use forms of weighted aggregation approach. The third one is based on VEGA (Vector Evaluated Genetic Algorithms) as developed by J.D. Schaffer [56]. Here we only consider the two dynamic aggregation approaches. This is because very similar vector evaluated approaches were proposed by other researchers and are discussed in their respective sections.

2.7.3.1 Changes to PSO update rules

The weighted aggregation based algorithms rely on the two formulas given in (2.26) and (2.27), both well known approaches. They are given here and examined in the K.E. Parsopoulos' paper only in two objective case. If the objectives are f_1 and f_2 then $f(x) = w_1 f_1(x) + w_2 f_2(x)$ is the aggregated objective.

$$w_1(t) = 0.5(1.0 + sign\,(sin\,(2\pi t/F))), w_2(t) = 1 - w_1(t) \tag{2.26}$$

Parameter	Description	Value
m	Neighbourhood size	2
ρ_1	Personal experience influence coefficient	0.5
ρ_2	Social influence coefficient	0.5
w	Inertia coefficient	1.0 to 0.0
F	Aggregator parameter	100, 200

Table 2.3: Parameter values for K. E. Parsopoulos et al. (2002) method.

$$w_1(t) = 0.5(1.0 + sin\,(2\pi t/F)), w_2(t) = 1 - w_1(t) \tag{2.27}$$

Formula (2.26) is known as Bang-Bang Weighted Aggregation (BWA) and (2.27) is known as Dynamic Weighted Aggregation (DWA). All the other parts of the algorithm are unchanged from the single objective function. Global and personal bests are updated using aggregated values of the multi-objective function just like they would be in the case of the single objective problem. Solutions are inserted in to the non-dominated point archive after every iteration.

2.7.3.2 Parameters

Parameter values as given by the authors are presented in Table 2.3. Parameter t is the current iteration index and F is the weight change frequency parameter. Parameter F was set to 100 in the BWA case and to 200 in the DWA case. Other PSO parameters are as follows: $\rho_1 = \rho_2 = 0.5$, $w = 1.0$ in the start and decreased linearly to $w = 0.4$ in the end.

2.7.4 S. Mostaghim et al. (2003)

In a paper Called "Strategies for Finding Good Local Guides in Multi-Objective Particle Swarm Optimization (MOPSO)" Sanaz Mostaghim and Jürgen Teich [40] discuss strategies for selecting the global attractor for the swarm (global best particle) from among Pareto-optimal solutions collected in the archive. In

this algorithm personal best solution is updated only if new solution dominates the old solution. This algorithm does not take in to account swarm topology.

2.7.4.1 Changes to PSO update rules

The changes to standard PSO velocity and position update rules can be summarized as in the list below.

1. Personal best solution $\boldsymbol{p}_i$ is updated if the old solution is dominated by the new solution.
2. Global best solution $\boldsymbol{g}_i$ for a given particle is chosen in such a way that the global best's sigma value is closest to the sigma value of position of particle i. The procedure is given in Algorithm 5.

Algorithm 5 Selecting the best global leader for each particle i via the Sigma method.

```
1:  function FindGlobalBest(Archive, x_i)
2:      m = |Archive|
3:      for j ← 1 to m do
4:          σ_j ← Sigma(y_j)
5:      end for
6:      σ_i ← Sigma(f(x_i))
7:      d ← ||σ_1 − σ_i||
8:      for j ← 2 to m do
9:          d_tmp ← ||σ_j − σ_i||
10:         if d_tmp ≤ d then
11:             d ← d_tmp
12:             k ← j
13:         end if
14:     end for
15:     return g_k
16: end function
```

Sigma value can be assigned to any point in the Pareto point space and is calculated in such a way that points that lie on the same line $f_1 = af_2$ get the same σ value. Furthermore points that lie on lines that are close to one another (have a similar angle of inclination) will have similar σ values. The reasoning

behind this heuristic is that by choosing points with similar σ values particles will move straight towards the Pareto frontier. If we have a problem with 2 objectives a point we will call point i with coordinates $(f_{1,i}, f_{2,i})$ if $f_{1,i} = f_{2,i}$ then $\sigma = 0$ in such case. If $f_{1,i} = 0$ then $\sigma = 1.0$ and if $f_{2,i} = 0$ then $\sigma = -1.0$. In essence σ represents the angle between the line $f_2 = \frac{f_{2,i}}{f_{1,i}} f_1$ and the line $f_1 = f_2$.

The calculation of sigma value can be extended to more dimensions than two. For three dimensions the calculation is given in Equation (2.29).

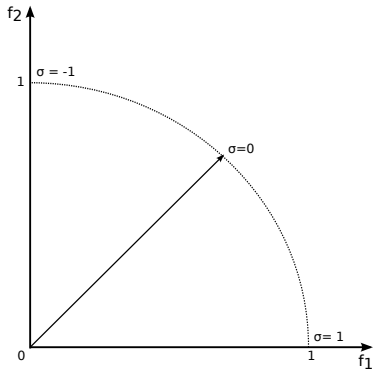


Figure 2.5: σ values for points on the unit circle. The same principles is generalized for points not on the unit circle and when there are more than one objective.

$$\sigma = \frac{f_1^2 - f_2^2}{f_1^2 + f_2^2} \tag{2.28}$$

$$\boldsymbol{\sigma} = \begin{pmatrix} f_1^2 - f_2^2 \\ f_2^2 - f_3^2 \\ f_3^2 - f_1^2 \end{pmatrix} / \left(f_1^2 + f_2^2 + f_3^2\right) \tag{2.29}$$

$$\sigma = \frac{(K_2 f_1)^2 - (K_1 f_2)^2}{(K_2 f1)^2 + (K_1 f_2)^2} \tag{2.30}$$

2.7.4.2 Parameters

Parameters and their values as proposed by the authors are given in Table 2.4. Parameters are similar to the ones used in other MOPSO variants. We chose

Parameter	Description	Value
ρ_1	Personal experience influence coefficient	1.0
ρ_2	Social influence coefficient	1.0
w	Inertia coefficient	0.4
m	Mutation probability	0.01, 0.05

Table 2.4: Parameter values for S. Mostaghim et al. (2003) method.

to use mutation probability (the probability that mutation will be applied to a particular coordinate of the position vector) to be 0.05 but 0.01 is said to have been applied by the authors as well.

2.7.5 J. E. Fieldsend et al. (2002)

In a paper titles A Multi-Objective Algorithm based upon Particle Swarm Optimizsation, an Efficient Data Structure and Turbulance Jonathan E. Fieldsend and Sameer Singh [18] propose a PSO variant for multi-objective optimization based on dominated trees - a data structure for maintenance of unconstrained Pareto archives. The dominated tree consists of a list of $L = |Z|/D$ composite points ordered by the weak dominance relation.

2.7.6 X. Li (2003)

Xiaodong Li [36] proposes an algorithm incorporating ideas from NSGA-II in to multi-objective particle swarm oprtmization. The differences between our generic MOPSO algorithm and this variant are summarized in the list below.

1. Global best $\boldsymbol{g}_i$ is updated by taking all mutually non-dominated particles from the current population in terms of their position $\boldsymbol{x}_i$, sorting them by either decreasing crowding distance or increasing niche count and taking a position at random from the top 5% of that list. Note that this way the global best $\boldsymbol{g}_i$ can be different for different particles.

2. Personal best $\boldsymbol{p}_i$ is updated by first doubling the population of the PSO to include the particle with updated $\boldsymbol{p}_i$ and with a non-updated $\boldsymbol{p}_i$. This population is then sorted using "non-dominated sorting" and retaining particles that make up the first half of the population after sorting. This way if the particle with updated $\boldsymbol{p}_i$ survives sorting and culling then $\boldsymbol{p}_i$ is, in effect, updated, otherwise not.

3. Coefficients ρ_1 and ρ_2 are both set to 2.0 and inertia coefficient w is linearly decreased from 1.0 to 0.4 during MOPSO evolution. Weighing the influence of personal and global best is again done using scalar random values rather than vectors used in single objective PSO.

Non-dominated sorting is done on a population of particles. First particles that are mutually non-dominating are identified and removed from the population. These particles compose the first non-dominated front. In the remaining population non-dominated particles are again identified and removed from the population. These particles compose the second non-dominated front. This is repeated until there are no more particles left in the population. In effect this groups particles in the swarm in to several non-dominated fronts. The result of this sorting is a list where particles from the first front come first, particles from the second front come second, etc. We use this concept to decide which particles will have their personal bests updated. For this purpose a population consisting of particles with both updated personal bests and non-updated personal bests is sorted in this way and only particles from the first half after sorting survive. The concept of non-dominated sorting is easiest understood graphically. We give an example of solutions sorted this way in Figure 2.6.

2.7.7 X. Hu et al. (2003)

Xiaohui Hu et al. [23] propose a particle swarm with extended memory for multiobjective optimization. It is similar to the dynamic neighbourhood by X. Hu et al. [22] that we discussed in a previous section. The variant described only works for problems with two objectives.

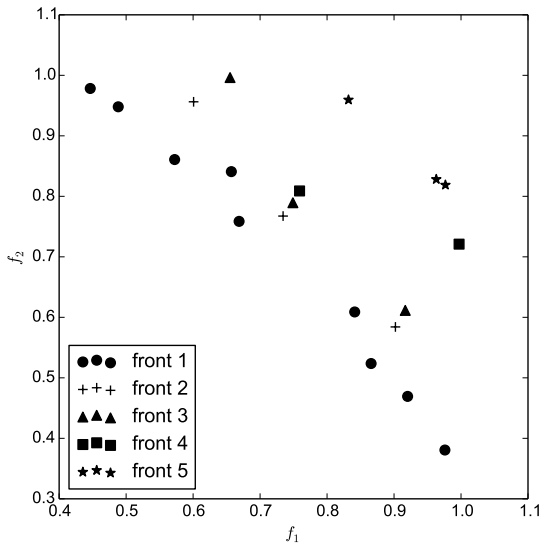


Figure 2.6: Sorting objective space solution in to separate non-dominated fronts.

2.7.7.1 Changes to PSO update rules

The selection of $\boldsymbol{g}_i$ and $\boldsymbol{p}_i$ is explained below.

- Personal best solution $\boldsymbol{p}_i$ is updated if it dominates any of the particles in the swarm.
- Global best solution $\boldsymbol{g}_i$ is chosen by first finding m particles closest to particle with index i in terms of the first objective. Then among those particles the best particle in terms of the second objective is chosen.

2.7.7.2 Parameters

The authors used the following values for the swarm parameters $w = 0.5 + 0.5U_{(0,1)}$, $\rho_1 = \rho_2 = 1.49445$ and v_{max} was set to "dynamic range of the particle on each dimension". The number of neighbours selected using the first objectives is set to 2.

Parameter	Description	Value
m	Neighbourhood size	2
ρ_1	Personal experience influence coefficient	1.49445
ρ_2	Social influence coefficient	1.49445
w	Inertia coefficient	0.5 + $U_{0,0.5}$

Table 2.5: Parameter values for X. Hu et al. (2003) method.

2.7.8 C. A. C. Carlos et al. (2004)

Coello Coello A. Carlos et al. [5] propose a Pareto dominance based approach. They use a mutation algorithm that is given in Algorithm 6 and which is also used by several other MOPSO methods. The only difference between this method and the one used in C. A. C. Coello et al. [6] is the use of this mutation operator. They also propose to use a simple constraint handling mechanism. The solution with the lowest amount of constraint violations is prefered if both solutions violate constraints, the solution that does not violate constraints is prefered if only one violates constraints, if both solutions do not violate constraints then the dominance relation is used. We, however, do not examine problems with constraints. Everything else is discussed in a section dedicated to that method.

2.7.8.1 Parameters

Parameters and their values as proposed by their authors are given in Table 2.6. The only new parameter is mutation rate m_r. Please not that mutation rate parameter in this case is not used directly. That is it is not the probability that a particular coordinate will be mutated. It's meaning can be understood from Algorithm 6.

2.7.9 K. E. Parsopoulos et al. (2004)

Konstantinos E. Parsopoulos et al. [48] developed a PSO variant for multi-objective problems similar to the VEGA approach in Genetic Algorithms. In this

Parameter	Description	Value
D	Number of objective space division points	30 to 50
ρ_1	Personal experience influence	1.0
ρ_2	Social influence	1.0
w	Inertia coefficient	0.4
m_r	Mutation rate	0.5

Table 2.6: Parameter values for C. A. C. Coello et al. (2004) method.

approach several subswarms are used. There are as many subswarms as there are objectives. The working of this PSO variant is best explained by looking at the changed velocity update Equation (2.31).

$$\boldsymbol{v}_{j,i} \leftarrow \chi \left(\boldsymbol{v}_{j,i} + \boldsymbol{\rho_1} \otimes (\boldsymbol{p}_{j,i} - \boldsymbol{x}_{j,i}) + \boldsymbol{\rho_2} \otimes (\boldsymbol{g}_{n(j),i} - \boldsymbol{x}_{j,i}) \right) \tag{2.31}$$

In the equation above j is the index of the subswarm and i is the index of a particle. Function $n(j)$ defines which subswarm does subswarm j takes it's global best solution from. An example of such a function is given in Equation (2.32) in this case subswarms are connected in a ring. Other configurations are of course possible.

$$n(i) = \begin{cases} i+1 & \text{if } i < M \\ 1 & \text{if } i = M \end{cases} \tag{2.32}$$

Usually each subswarm is connected in a `gbest` topology. However it is also possible to use other topologies. Whether it is advantagious to do would be an interesting area of research. This approach is trivially parallelizable with each subswarm capable of running as a separate process.

The authors used four test problems. In all four cases they note that their algorithm performs better than VEGA. They however did not test it against any other Multi-Objective optimization approaches, such as different Multi-Objective PSO variants.

Parameter	Description	Value
ρ_1	Personal experience influence	2.05
ρ_2	Social influence	2.05
χ	Constriction coefficient	0.729

Table 2.7: Parameter values for K. E. Parsopoulos et al. (2004) method.

2.7.9.1 Changes to PSO update rules

The method uses the constriction coefficient variant of PSO proposed by M. Clerc et al. [4]. Other then that the only difference from single objective PSO methods is that global best is taken from the neighbouring swarm.

Updating personal and global best solutions are performed as given in the list below.

- Personal best $\boldsymbol{p}_i$ is updated by checking if the new solution has the lower value of the objective that is assigned to the subswarm the particle is in.
- Global best $\boldsymbol{g}_i$ is taken from the neighbouring swarm. The particle from the neighbouring swarm with the lowest value of that swarms objective is taken as the global best.

2.7.9.2 Parameters

Parameter values are given in Table 2.7. They seem to be taken from the single objective version of this PSO method.

2.7.10 S. Mostaghim et al. (2004)

Sanaz Mostaghim et al. [41] propose a sub-swarm based approach to multi-objective optimization using PSO. This method divides the optimisation process in to two stages. Both stages use the Sigma method by S. Mostaghim et al. [40] which was discussed in a previous section. The basic idea of the method is to

assign special vectors to each solution. This way the similarity between two solutions will be measured by measuring how close those vectors are to each other in terms of Euclidean distance. How these vectors are calculated and the rationale behind it is explained in a section about the method. This new algorithm builds on the ideas behind it. During the first stage the algorithm works the same as in the original method. After running it an approximation of the real PF is the end result. This approximated PF A is then used for the second stage. The difference between our generic MOPSO algorithm and the second stage is shown below. We do not present an explanation of the first stage since it is identical to the Sigma method we discussed in an above section.

- Local best $\boldsymbol{p}_i$ is updated if the new solution $\boldsymbol{x}_i$ dominates the old $\boldsymbol{p}_i$.
- Global best $\boldsymbol{g}_i$ is selected by finding a solution in the approximation A (returned by the first stage of the algorithm) that is closest in terms of sigma value.

It can be seen that during the second stage the particles will arrange themselves in to subswarms. This is in the sense that groups of particles will tend to select one of the solutions in A over another as their global best attractor. So all the particles in a subswarm will have the same global guide. The rationale here is that the second stage will explore around the solutions found by the first stage. This way better coverage of the real PF is achieved. Each subswarm will explore the area around one of the points in the approximation A. The size of the nondominated point archive is limited during the first stage. During the second stage the size of the archive is unlimited.

Mutation is applied. The mutation operator is of the form $x_{ij} = x_{ij} + U_{(-1,1)}x_{ij}$ and is applied with probability 0.07. Inertia weight $w = 0.4$ and coefficients $\rho_1 = \rho_2 = 1.0$ as in the original method. Authors restrict the archive size during the first stage to 50 and don't restric the archive size during the second stage.

2.7.11 C. R. Raquel et al. (2005)

Carlo R. Raquel et al. [52] propose a crowding distance based approach. This is a very simple approach that uses crowding distance to select global best. It

also employs an elaborate mutation operator in order to maintain diversity. The solutions found are stored in a non-dominated point archuive. The size of non-dominated point archive is fixed. As it is fixed there may come a time that a new solution needs to be added to the archive but it has already reached it's size limit. In this case the archive is sorted by decreasing value of crowding distance and a solution from the bottom 10% of the sorted list is removed. The candidate solution is then added to the archive.

2.7.11.1 Changes to PSO update rules

Changes to PSO velocity and position update rules are given in the list below.

- Personal best $\boldsymbol{p}_i$ is updated if new solution $\boldsymbol{x}_i$ dominates the old $\boldsymbol{p}_i$.
- Global best $\boldsymbol{g}_i$ is updated by sorting the solutions in nondominated point archive by decreasing crowding distance and taking a solution at random from the top 10% of the sorted list.

Mutation operator used in this variant is given in Algorithm 6. Here $flip(p)$ evaluates to truth with probability p, m_r is mutation rate, x_{ij} is the j-th coordinate of i-ith particle's position, a_j is lower boundary constraint for coordinate j in decision space and b_j is upper boundary constraint. The notation here is consistent with rest of this thesis. Note that mutation rate here has a specific meaning, different than in most other cases. It is the probability that mutation operator will be applied to the given particle as opposed to mutation operator being applied to a value of a coordinate in decision space. Mutation is only applied if current iteration number is lower than maximum iterations times mutation probability.

2.7.11.2 Parameters

Parameters and their values as proposed by the authors are given in Table 2.8. They are similar to ones used by other methods. It is important to observe that mutation rate m_r does not mean mutation probability but instead is a parameter used in mutation algorithm as given in Algorithm 6.

Algorithm 6 Mutation operator used in Carlo R. Raquel et al. [52] algorithm.

if $flip\left(\left(1-\frac{t}{t_{max}}\right)^{\frac{5}{m_r}}\right)$ **then**
  $j \leftarrow U_{\{1,2,\ldots,d\}}$
  $range \leftarrow (b_j - a_j)\left(1-\frac{t}{t_{max}}\right)^{\frac{5}{m_r}}$
  $ub \leftarrow x_{ij} + range$
  $lb \leftarrow x_{ij} - range$
  if $ub > b_j$ **then**
    $ub \leftarrow b_j$
  end if
  if $lb < a_j$ **then**
    $lb \leftarrow a_j$
  end if
  $x_{ij} \leftarrow U_{(lb,ub)}$
end if

Parameter	Description	Value
w	Inertia coefficient	0.4
ρ_1	Personal experience influence coefficient	1.0
ρ_2	Social influence coefficient	1.0
m_r	Mutation rate	0.5

Table 2.8: Parameter values for C. R. Raquel et al. (2005) method.

2.7.12 J. E. Alvarez-Benitez et al. (2005)

Julio E. Alvarez-Benitez et al. [2] discuss several variants of PSO based exclussively on Pareto dominance concepts. In their algorithms personal best for each particle $\boldsymbol{p}_i$ is updated if $\boldsymbol{f}(\boldsymbol{x}_i) \preceq \boldsymbol{f}(\boldsymbol{p}_i)$ or if $\boldsymbol{f}(\boldsymbol{x}_i) \not\prec \boldsymbol{f}(\boldsymbol{p}_i)$ and $\boldsymbol{f}(\boldsymbol{p}_i) \not\prec \boldsymbol{f}(\boldsymbol{x}_i)$. Or to put it in words if the new value in objective space weakly dominates the old personal best or the two are mutually non-dominated. The authors propose three strategies to select global best value g_i we described the three in the list below.

ROUNDS The basic idea of this method is for particles to move towards those archive members that dominate the fewest current particle positions (but at least one) in hopes that those members represent sparsely populated areas of objective space. First of all for each member of the non-dominated

solution archive $\boldsymbol{a} \in A$ we calculate how many particle positions it dominates. So for vector $\boldsymbol{a} \in A$ we assign a set $X_{\boldsymbol{a}} = \{\boldsymbol{x} \mid \boldsymbol{x} \in \mathbf{X} \wedge \boldsymbol{a} \prec \boldsymbol{x}\}$ then we select $\boldsymbol{a}$ so as to minimize $|X_{\boldsymbol{a}}|$ and assign it's corresponding solution space vector as $\boldsymbol{g}_i$ for a particle chosen at random from $X_{\boldsymbol{a}}$. After this vector $\boldsymbol{a}$ is removed from consideration until all remaining particles get $\boldsymbol{g}_i$ assigned to them. This is repeated each iteration of the algorithm.

RANDOM Here for particle position $\boldsymbol{x}_i \in \mathbf{X}$ we assign a list of solutions $A_{\boldsymbol{x}_i}$ from archive $\mathbf{A}$ that dominate $\boldsymbol{x}_i$. From this list we then choose one at random with equal probabilities. If $\boldsymbol{x}_i \in \mathbf{A}$ then $A_{\boldsymbol{x}_i}$ is empty (due to the fact that points in the archive $\mathbf{A}$ cannot dominate each other by definition) in which case we simply choose a point from $\mathbf{A}$ at random, again with equal probabilities.

PROB This method can be seen as a combination of the two previous ones. In this case, for vector $\boldsymbol{x}_i$ we assign a global best similarly to the way it is done in RANDOM approach, however instead of uniform probabilities we assign a probability to each vector $\boldsymbol{a} \in A_{\boldsymbol{x}_i}$ inversely proportional to $|X_{\boldsymbol{a}}|$. To put it simply we proceed the same way as in RANDOM but instead of giving equal probabilities to each solution that dominates a given particle we assign probabilities to those solutions based on how many particles in the swarm they dominate. So an archive solution that dominates some particle $\boldsymbol{x}_i$ is less likely to become that particle's global best $\boldsymbol{g}_i$ the more particles in the swarm it dominates. The reasoning behind this strategy that authors state is that archive solutions near the areas of Pareto front that contain few solutions are likely to dominate fewer particles and therefore to explore those areas better we want to emphasize them.

Particle swarm parameters are set as $\rho_1 = \rho_2 = 1$ and $w = 0.5$ which is again different from what is reported as good parameters for single objective optimization.

To recap what was said before - these algorithms are identical to our basic multi-objective PSO algorithm given in Algorithm 4 with personal best $\boldsymbol{p}_i$ updated if new solution $\boldsymbol{x}_i$ weakly dominates $\boldsymbol{p}_i$ or if the two are mutually non-dominating and global best $\boldsymbol{g}_i$ updated according to one of the strategies ROUNDS, RANDOM or PROB.

2.7.13 M. R. Sierra et al. (2005)

Proposed by Margarita R. Sierra et al. [60] in a paper called "Improving PSO-based Multi-Objective Optimization using Crowding, Mutation and ϵ-Dominance". The method uses two separate archives in place of one. One archive is the same size as the number of particles in the swarm. This archive is used to select global leaders from. If after adding a new solution to this archive it's size exceeds that of the particle swarm population, the worst members of the archive in terms of crowding distance (a concept discussed in the introductory chapter) are removed until the archive has the required number of members. There is another archive that is maintained differently. This is the archive that will be reported as the end-result after an algorithm run. This archive uses the ϵ-dominance relation to update it's solution set. This approach was discussed in the introductory chapter of this thesis in a section about pareto point maintenance. The value of parameter ϵ will influence the maximum size of the archive. Authors fail to give the value of parameter ϵ used to achieve the results in their experiments.

2.7.13.1 Changes to PSO update rules

Changes to PSO velocity and position update rules are given in the list below. After each position update mutation is applied.

- Personal best position $\boldsymbol{p}_i$ is if the particles new position $\boldsymbol{x}_i$ dominates $\boldsymbol{p}_i$ or if the two are mutually non-dominating.
- Two solution are selected at random from the swarm archive (the one that has the same number of solutions as the number of particles in the swarm) and pick the one that has larger value of crowding distance. This solution will be used as the global best solution $\boldsymbol{g}_i$.

The authors propose to divide the particle swarm population in to three sub-populations. The only difference between the sub-populations are the mutation operators applied to them. They share the same non-dominated point archives

and otherwise act identically. The sub-populations are of equal size. The first sub-population has no mutation applied to it all, the second one uses uniform mutation where the range of mutation is fixed and the third one uses non-uniform mutation where the allowed range decreases over time. What any of this actually means and what the ranges are is not explained in the paper. The probility with which mutation is applied to each coordinate is set to $\frac{1}{d}$ where d is problem dimensionality. Since the precise nature of mutation operators applied is not explained in the paper we simply reset the coordinates value to a random value with uniform probability from valid range for that coordinate with probability $\frac{1}{d}$.

Uniform mutation is defined in Equation (2.33) and can be found in a book by Kalyanmoy Deb [11]. In this case a coordinate is simply initialized at random with uniform probability distribution from the allowed range for that coordinate. Here a_j and b_j are lower and upper bounds for coordinate j in the solution space.

$$x_{ij} \leftarrow U_{(0,1)}(b_j - a_j) \tag{2.33}$$

Non-Uniform mutation is defined in Equation (2.34) and can also be found in a book by Kalyanmoy Deb [11]. In this case the range that mutation can move coordinate positions to is gradually decreased during the evolution of the swarm.

$$x_{ij} \leftarrow x_{ij} + \tau(b_j - a_j)\left(1 - U_{(0,1)}^{\left(1-\frac{t}{t_{max}}\right)^c}\right) \tag{2.34}$$

Here, τ is assigned at random to either -1 or 1 with equal probability during each evaluation of the mutation operator, parameter t_{max} is the maximum number of iterations and c is a user specified parameter.

2.7.13.2 Parameters

Parameter and their values as recommended by the authors are given in Table 2.9. Here w is defined as a uniformly distributed number from the interval $(0.1, 0.5)$,

Parameter	Description	Value
w	Inertia coefficient	$U_{(0.1,0.5)}$
ρ_1	Personal experience influence coefficient	$U_{(1.5,2.0)}$
ρ_2	Social influence coefficient	$U_{(1.5,2.0)}$

Table 2.9: Parameter values for M. R. Sierra et al. (2005) method.

ρ_1 and ρ_2 are uniformly distributed numbers from the interval $(1.5, 2.0)$. It is not completely clear if w, ρ_1 and ρ_2 are assigned during each iteration or at the initialization phase of the algorithm. We assume that the authors mean they are randomized during each iteration for each particle. It is interesting to note that the authors use scalar random values instead of commonly accepted practice of using random number vectors for r_1 and r_2 of the same dimensionality as the solution space.

2.7.14 M. Salazar-Lechuga et al. (2005)

Maximino Salazar-Lechuga et al. (2005) [55] propose a fitness sharing based approach. Fitness sharing seems to be identical with niching used to determine which individuals are in the well explored regions of Pareto front by counting how many other individuals fall within a given distance to them. Fitness sharing means, in essence, penalizing individuals that have other individual surrounding them in terms of Euclidean distance between solutions.

$$fs_i = \frac{10}{\sum_{j=0}^{n} share_{ij}} \tag{2.35}$$

$$share_{ij} = \begin{cases} 1 - \left(\frac{d_{ij}}{\sigma_{share}}\right)^2 & \text{if } d_{ij} < \sigma_{share} \\ 0 & \text{otherwise.} \end{cases} \tag{2.36}$$

$$d_{ij} = \|\boldsymbol{x}_i - \boldsymbol{x}_j\| \tag{2.37}$$

In above equations we denote fitness sharing coefficient of solution $\boldsymbol{x}_i, i \in$

$\{1, \ldots, |A|\}$ as fs_i. Such coefficient is assigned to each solution in the nondominated point archive A. It is inversely proportional to the number of solutions that are within a hypersphere with radius σ_{share} with it's center at $\boldsymbol{x}_i$. Futhermore there is a fitness sharing penalty the closer other solutions are to $\boldsymbol{x}_i$. Archive A for this algorithm is size bounded. New solutions will be accepted after the archive becomes full if their fitness sharing value is larger than the smallest fitness sharing value in the archive. The candidate solution will then replace the solution with the smallest fitness sharing value.

2.7.14.1 Changes to PSO update rules

We summarize differences between this algorithm and our generic MOPSO in the list below.

1. Global best $\boldsymbol{g}_i$ is selected via roulette wheel selection from the nondominated point archive using fitness sharing to calculate probabilities.
2. Personal best $\boldsymbol{p}_i$ is updated if the new position dominates $\boldsymbol{p}_i$.

2.7.14.2 Parameters

Parameter values and their meaning are summarized in Table 2.10. Social and personal influence coefficients ρ_1, ρ_2 and inertia coefficient w are set to values commonly used in MOPSO variants. Parameter σ_{share} is used to calculate the diameter of a hypersphere which is used to penalize particles when calculating velocity updates. The authors state that they selected the values for this parameter empirically depending on the problem. We used a value of 0.01 since this seems to be used for problems with large non-dominated point archives.

2.7.15 P. K. Tripathi et al. (2007)

Praveen Kumar Tripathi et al. [63] propose to use a variant of multi-objective PSO with time variant inertia and accelaration coefficients. The parameters w,

Parameter	Description	Value
σ_{share}	Method specific parameter	0.01, 0.1, 0.2, 1.0, 2.0
ρ_1	Personal experience influence	1.0
ρ_2	Social influence	1.0
w	Inertia coefficient	0.4

Table 2.10: Parameter values for M. Salazar-Lechuga et al. (2005) method.

ρ_1 and ρ_2 are updated as given in equations below.

$$w_t = (w_i - w_f)\frac{t_{max} - t}{t_{max}} + w_f \tag{2.38}$$

Here, w_t is inertia during iteration t and t_{max} is the maximum number of iterations. Inertia changes during the course of the PSO evolution from w_i to w_i linearly. Personal and social influence parameters ρ_1 and ρ_2 are treated similarly as given in the two equations below.

$$c_{1t} = (c_{1f} - c_{1i})\frac{t}{t_{max}} + c_{1i} \tag{2.39}$$

$$c_{2t} = (c_{2f} - c_{2i})\frac{t}{t_{max}} + c_{2i} \tag{2.40}$$

2.7.15.1 Changes to PSO update rules

Mutation is used in this MOPSO. The authors use the scheme given in Equation (2.41), with $flip$ being a binary random event with equally possible results of 0 and 1 and x_{ij} being the j-th coordinate of particle i's position and Δ defined in Equation (2.42).

$$x_{ij} = \begin{cases} x_{ij} + \Delta(t, b_j - x_{ij}), \text{ if } flip = 0, \\ x_{ij} + \Delta(t, x_{ij} - a_j), \text{ if } flip = 1. \end{cases} \tag{2.41}$$

Parameter b in Equation (2.42) is set to 5, other parameters have the same meaning as elsewhere.

$$\Delta(t, z) = z\left(1 - U_{(0,1)}^{\left(1-\frac{t}{t_{max}}\right)^b}\right) \tag{2.42}$$

Personal best solution $\boldsymbol{p}_i$ is updated if the new solution dominates the old one. Global best solution $\boldsymbol{g}_i$ is chosen from the archive in a way that maximizes what the authors call particular solutions density "density". It is similar in concept to crowding distance discussed before. If our problem has k objectives we will sort the archive by the non-decreasing value of each objective in turn, measure the distance between a solution and the solution to it's right and use that instead of as in crowding distance the distance between the solution to the left and the solution to the right. The reason for using this instead of using crowding distance is unclear from the paper. After calculating densities as described above the algorithm uses roulette selection to determine global best.

Let us recap the differences between this algorithm and our general multi-objective particle swarm optimization algorithm:

1. Parameters w, ρ_1 and ρ_2 are adjusted during the course of PSO evolution using equations (2.38), (2.39) and (2.40).
2. Personal best is updated if the new position dominates old personal best.
3. Global best is chosen from the archive using roulette selection using above described density calculations for probabilities.
4. Instead of using random value vectors to randomly weigh personal and swarm influence in velocity update as is customarily done in single objective PSO it uses scalars.

2.7.15.2 Parameters

Parameters and their values are summarized in Table 2.11. These six parameters (w_i, w_f, c_{1i}, c_{1f}, c_{2i}, c_{2f}) were set as follows by the authors of the paper: $w_i = 0.7$,

Parameter	Description	Value
c_{1i}	Initial personal experience influence	2.5
c_{1f}	Final personal experience influence	0.5
c_{2i}	Initial social influence	2.5
c_{2f}	Final social influence	0.5
w_i	Initial inertia coefficient	0.7
w_f	Initial inertia coefficient	0.4
m	Mutation probability	0.05

Table 2.11: Parameter values for P. K. Tripathi et al. (2007) method.

$w_f = 0.4$, $c_{1i} = 2.5$, $c_{1f} = 0.5$, $c_{2i} = 0.5$ and $c_{2f} = 2.5$. In this way inertia is decreased over the course of the evolution of the algorithm. The influence of particle's personal experience is also decreased over time and the influence of social experience is increased over time. This, in theory, should mean that the swarm emphasizes global search during the first stages of it's evolution and local search nearing the end.

2.7.16 W. Peng et al. (2008)

Wei Peng et al. [49] propose a decomposition based multi-objective particle swarm optimization algorithm. In this variation of MOPSO the multiobjective problem $\boldsymbol{f}$ is reduced to a problem in Equation (2.43). Basically you want to minimize the largest objective when objectives are weighted using a vector $\boldsymbol{\lambda}$ which is called the weight vector. This may seem similar to weighted aggregate approaches, however in our case each particle will get a different weight vector $\boldsymbol{\lambda}$. This in effect means that each particle minimizes a different function.

$$g\left(\boldsymbol{x} \mid \boldsymbol{\lambda}\right) = \max_{1 \leq i \leq m} \{\lambda_i | f_i(\boldsymbol{x}) - z_i^*|\} \tag{2.43}$$

The main remaining point is how to calculate weight vectors. The way the authors propose to calculate it is by taking combinations with replacement of values from the set $\left\{\frac{0}{H}, \frac{1}{H}, \ldots, \frac{H}{H}\right\}$ where H is a user chosen parameter. Parameter H will also decide the number of particles in the swarm with $N = C_{H+m-1}^{m-1}$. In other words, weight vectors $\boldsymbol{\lambda}$ are all possible vectors of length m where

each individual weight takes one of the values from the aforementioned set. Reference point $\boldsymbol{z}^*$ is calculated by taking lowest values of each objective when solution $\boldsymbol{x}$ belongs to objective space. Vector $\boldsymbol{x}^* = \{z_1^*, \ldots, z_m^*\}$ where $z_i^* = \min\{f_i(\boldsymbol{x}) \mid \boldsymbol{x} \in \Omega\}$.

2.7.16.1 Changes to PSO update rules

The changes to how personal best $\boldsymbol{p}_i$ and global best $\boldsymbol{g}_i$ for particle with index i are selected are given in the list below. Otherwise the velocity and position update rules are the same as in the case of single objective PSO. In the velocity update rule the difference between personal best and current position and between global best and current position is randomly scaled by a random scalar value. This is different from single objective PSO but common in multi-objective PSO. After each iteration polynomial mutation is used as described in H. Li et al. [34]. Mutation is applied to each coordinate with constant probability.

- Update of the personal best $\boldsymbol{p}_i$ is simple. If $g\left(\boldsymbol{x}_i \mid \boldsymbol{\lambda}_i\right) \leq g\left(\boldsymbol{p}_i \mid \boldsymbol{\lambda}_i\right)$ then $\boldsymbol{p}_i$ is updated, otherwise not. This simply means if the value of the aggregate function is lower then the personal best it is updated, just like in the single objective case.
- We define $B(i)$ as the set of indices of neighbours of particle with index i. The neighbours of particle with index i are those T particles whose weights vectors are closest in terms of Euclidean distance. If weight vectors are set in advance and don't change during the evolution of the swarm then these neighbourhood relations can be calculated in advance. Then for each particle i and for each neighbour $j \in B(i)$ we update $\boldsymbol{g}_j \leftarrow \boldsymbol{g}_i$ if $g\left(\boldsymbol{p} \mid \boldsymbol{\lambda}_j\right) \leq g\left(\boldsymbol{g}_j \mid \boldsymbol{\lambda}_j\right)$. In other words, we pick the neighbour whose personal best solution give the lowest value of the aggregate funciton when evaluated with the weight vector of particle i.

2.7.16.2 Parameters

Parameter values for numerical parameters used in this method are summarized in Table 2.12. Inertia weights and social and personal influence coefficients used

Parameter	Description	Value
N	Population size	100
M	Archive size	500
ρ_1	Personal experience influence	2.0
ρ_2	Social influence	2.0
w	Inertia coefficient	0.4
T	Neighbourhood size	30
p_m	Polynomial mutation parameter	0.05

Table 2.12: Parameter values for W. Peng et al. (2008) method.

are relatively common in PSO. Neighbourhood size is the number of particles that will be searched when selecting global best solution. As mentioned before particle's neighbourhood is constructed to include those particles whose weight vectors are closest in terms of Euclidean distance to the particle in question.

2.7.16.3 Other comments

Due to the nature of this method the number of particles depends on the parameter H. For test problems with two objectives we used value of $H = 25$ which means there are 351 particles in the swarm. This means that to not exceed $300,000$ function evaluations we had to run the swarm for 855 iterations. For problems with three objectives we used value of $H = 12$ which means there are 364 particles in the swarm and we ran the swarm for 824 iterations. For the five objective case we used $H = 7$ and in turn 462 particles and ran the swarm for 649 iterations.

2.7.17 U. Wickramasinghe et al. (2008)

Upali Wickramasinghe et al. [67] propose a variant of MOPSO that uses ideas from differential evolution. Namely the differential evolution operator is used to create global best $\boldsymbol{g}_i$ for each particle i. Otherwise this variant is similar to the one described by Xiadong Li [36]. We enumerate the differences between this variant and our generic multi-objective particle swarm optimization algorithm below.

1. Global best $\boldsymbol{g}_i$ is chosen by first selecting three particle positions $\boldsymbol{x}_{r_1}$, $\boldsymbol{x}_{r_2}$ and $\boldsymbol{x}_{r_3}$ so that $i \neq r_1 \neq r_2 \neq r_3$ at random. Then differential evolution operator is applied to form $\boldsymbol{g}_i$.

2. Personal best solutions are updated by non-dominated sorting a population consisting of both updated and not-updated particles and then keeping the first half. This procedure is explained in more detail in Section 2.7.6.

3. Parameters ρ_1 and ρ_2 are set to 2.05, χ is set to 0.7298 while inertia coefficient w is simply set to one. Differential evolution parameter CR is set to 0.2 and parameter F is set to 0.4, this is different from parameters traditionally used in single objective differential evolution where CR is usually set to 0.9 and F is set to 0.5.

Differential evolution operator is used to generate new solutions from existing ones. For each solution $\boldsymbol{x}_i$ we need three other solutions $\boldsymbol{x}_{r_1}$, $\boldsymbol{x}_{r_2}$ and $\boldsymbol{x}_{r_3}$ distinct from each other as well as from the current solution $\boldsymbol{x}_i$. For a particle i we form it's $\boldsymbol{g}_i$ using Equation (2.44).

$$g_{i,j} = \begin{cases} x_{r_1,j} + F\left(x_{r_2,j} - x_{r_3,j}\right) & \text{if } \left(U_{(0,1)} < CR \vee j = j_{rand}\right) \\ x_{i,j} & \text{otherwise} \end{cases} \tag{2.44}$$

2.7.18 A. J. Nebro et al. (2009)

Antonio J. Nebro et al. [42] propose a variant of multi-objective PSO algorithm that is similar to the one proposed by Margarita R. Sierra et al. [60] but with certain modifications. This variant uses a separate archive for storing particles that will be used as personal best. This archive has the upper size limit set to the number of particles in the swarm and uses crowding distance discussed previously to maintain fixed size.

2.7.18.1 Changes to PSO update rules

The main difference between this algorithm and the one developed in work by Margarita R. Sierra et al. [60] is that it uses constriction coefficient proposed by Maurice Clerc et al. [4]. One other difference is that this algorithm returns it's leaders archive as the end result while the one it is based on uses a separate, possibly larger, archive for the end result approximation.

Again the authors use scalar values for r_1 and r_2 instead of more usual vectors of uniformly distributed random values of the same dimensionality as the solution space. The velocity is further multiplied by the aforementioned constriction factor calculations which we reproduce in Equation (2.45). Since ρ_1 and ρ_2 are chosen randomly they expand the definition of the constriction coefficient to take in to account cases where $\rho_1 + \rho_2 < 4$ by assigning value of χ equal to one in such cases.

$$\chi = \frac{2}{2 - \rho - \sqrt{\rho^2 - 4\rho}} \tag{2.45}$$

$$\rho = \begin{cases} \rho_1 + \rho_2 & \text{if } \rho_1 + \rho_2 > 4 \\ 1 & \text{if } \rho_1 + \rho_2 \leq 4 \end{cases} \tag{2.46}$$

Furthermore we set limits to each coordinate of velocity vector equal to half the allowed range for that coordinate as shown in Equation (2.47).

$$v_{ij} \leftarrow \begin{cases} \delta_j & \text{if } v_{ij} > \delta_j \\ -\delta_j & \text{if } v_{ij} \leq -\delta_j \\ v_{ij} & \text{otherwise.} \end{cases} \tag{2.47}$$

In this equation δ_j is the limit on velocity and is defined in Equation (2.48).

$$\delta_j = \frac{b_j - a_j}{2} \tag{2.48}$$

In other words δ_j is half the allowed range for coordinate j in the decision space.

Parameter	Description	Value
w	Inertia coefficient	0.5
ρ_1	Personal experience influence coefficient	$U_{(1.5,2.5)}$
ρ_2	Social influence coefficient	$U_{(1.5,2.5)}$
η_m	Polynomial mutation operator parameter	20

Table 2.13: Parameter values for A. J. Nebro et al. (2009) method.

This variant in addition to uniform and non-uniform mutation operators used in algorithm it is based on also uses polynomial mutation operator that can be found in a book by Kalyanmoy Deb [11] and which we define in Equation (2.49).

$$x_{ij} \leftarrow x_{ij} + (b_j - a_j)\bar{\delta}_j \tag{2.49}$$

Where parameter $\bar{\delta}_i$ is calculated from the polynomial function instead of uniform distribution $P(\delta) = 0.5(\eta_m + 1)(1 - |\delta|)^{\eta_m}$ as follows:

$$\bar{\delta}_j = \begin{cases} (2U_{(0,1)})^{\frac{1}{\eta_m+1}} - 1 & \text{if } U_{(0,1)} < 0.5, \\ 1 - (2(1 - U_{(0,1)}))^{\frac{1}{\eta_m+1}} & \text{if } U_{(0,1)} \geq 0.5 \end{cases}$$

Parameter η_m is set to 20. Particles using this additional mutation operator constitute 15% of the population. It is not entirely clear how other mutations are distributed but we take it to be 30% uniform, 30% non-uniform, 15% polynomial and 15% no mutation to be consistent with the method on which this one is based.

2.7.18.2 Parameters

Parameter values as proposed by the authors are given in Table 2.13. Social and personal influence coefficients are uniformly distributed random numbers. Polynomial mutation parameter η_m is set to 20.

2.7.19 Y. Wang et al. (2009)

Yujia Wang et al. (2009) [66] propose a variant of PSO using preference order ranking. It uses the concept of Pareto efficiency, which is a generalization of Pareto dominance. It also has to be stated that if a solution is of efficiency order k it is also of efficiency order $k+1$. This was proven by I. Das [9]. As such we will be interested in minimal efficiency order of a solution.

Definition 2.7.1. (Pareto efficiency of order k)

A point $\boldsymbol{x}^* \in \Omega$ is said to be Pareto efficient of order k if $\boldsymbol{f}(\boldsymbol{x}^*)$ is not dominated by any member of $\boldsymbol{f}(\Omega)$ for any of the k-element subsets of the objectives. In other words, a point is efficient of order k if it is Pareto optimal in all the $\binom{m}{k}$ subspaces of Ω obtained by considering only k objectives at a time. If $k = m$ then the concept reduces to that of Pareto optimality.

□

The method uses an unusual velocity update rule given in the Equation (2.50).

$$\boldsymbol{v}_i \leftarrow w_i \boldsymbol{v}_i + (1 - r_2)\rho_1 r_1(\boldsymbol{p}_i - \boldsymbol{x}_i) + (1 - r_2)\rho_2(1 - r_1)(\boldsymbol{g}_i - \boldsymbol{x}_i) \tag{2.50}$$

The inertia coefficient w_i is updated in such a way that it follows an exponential curve as given in Equation (2.51).

$$w_i \leftarrow w_i f_w \tag{2.51}$$

- Personal best solution $\boldsymbol{p}_i$ is updated if the new solution dominates the old personal best solution.
- Global best solution $\boldsymbol{g}_i$ is selected using preference order ranking scheme. The solutions in the archive A are sorted by their minimal efficiency order. This is done by first identifying all the $\binom{m}{k}$ subsets of the decision space for all $k \leq m$ and then finding the lowest value of k for which a solution

Parameter	Description	Value
w_0	Initial value of the inertia coefficient	1.4
f_w	Inertia coefficient update coefficient	0.975
ρ_1	Personal experience influence	1.5
ρ_2	Social influence	2.5

Table 2.14: Parameter values for Y. Wang et al. (2009) method.

is k efficient. The solution with the highest efficiency order is chosen as global best.

The efficiency order concept is also used when the archive becomes full. The solutions with lowest value of efficiency order are removed from the archive in order to make space. Parameter values used in the method and mentioned in the description above were set to the values given in Table 2.14 and are the same as used by the authors. More information on the preference order ranking can be found in a work of F. di Pierro et al. [13].

2.7.20 N. Al Moubayed et al. (2010)

Noura Al Moubayed et al. [1] propose a decomposition based multi-objective particle swarm optimisation algorithm. In the proposed method Tchebyscheff method for weighted aggregation is used. That is for a given weight vector $\boldsymbol{\lambda}$ the problem is that of minimizing function given in Equation (2.52). In essense this means minimizing the worst objective.

$$g(\boldsymbol{x} \mid \boldsymbol{\lambda}, \boldsymbol{z}^*) = \max_{1 \leq i \leq m} \{\lambda_i | f(\boldsymbol{x})_i - z_i^*|\} \tag{2.52}$$

Weight vectors $\boldsymbol{\lambda}_i$ are initialized to random values from the uniform distribution. The index i is the index of a particle that this weight vector belongs to. The values are further scaled so that they add up to one. Weight vectors are assigned particles so that they give the lowest values of the aggregate function. That is from a set of n generated weight vectors we choose one for a particle if it gives the lowest value of the aggregate function for that particle among all the

weight vectors. Each particle gets a unique weight vector. Weight vectors are not reinitialized after each iteration and stay the same during the evolution of the swarm. The method also makes use of a non-dominated point archive. The non-dominated point archive is updated by taking in to account crowding distance measure. The measure is used to decide which solutions to delete if the archive is full.

2.7.20.1 Changes to PSO update rules

The changes to personal and global best selection are given in the list below. In the velocity update rule the differences between local and global best and current position are scaled by random scalars from the range $(0.1, 1.0)$ for some reason. Everything else is the same as we outlined in our generic MOPSO algorithm.

- Personal best $\boldsymbol{p}_i$ is updated the same way as in single objective optimization but using the value of the aggregated function instead. That is personal best is updated if $g(\boldsymbol{x}_i \mid \boldsymbol{\lambda}_i, \boldsymbol{z}^*) \leq g(\boldsymbol{p}_i \mid \boldsymbol{\lambda}_i, \boldsymbol{z}^*)$.
- Global best $\boldsymbol{g}_i$ is selected at random from the neighbourhood. The neighbourhood is defined as the N particles that have the values of weight vectors closest to the particle in question with regards to Euclidean distance.

2.7.20.2 Parameters

Parameters and their values for this as used by the authors are summarized in Table 2.15. Neighbourhood of particle with index i is defined by N particles that are closest to the particle i in terms of Euclidean distance between weight vectors. Social, global influence coefficients and inertia coefficient are all random numbers in the given ranges. This is relatively unusual in the context of PSO.

Parameter	Description	Value
N	Neighbourhood size	30
ρ_1	Personal experience influence	$U_{(1.5,2.0)}$
ρ_2	Social influence	$U_{(1.5,2.0)}$
w	Inertia coefficient	$U_{(0.1,0.5)}$

Table 2.15: Parameter values for N. al Moubayed et al. (2010) method.

2.7.21 A. Elhossini et al. (2010)

Ahmed Elhossini et al. [15] propose a variant of PSO based on the strength Pareto approach. In the case particle's strength is defined as the number of particles it dominates in the swarm. Formally we state this in Equation (2.53) with P being the set of current particle positions and A the solutions in the non-dominated point archive.

$$s_i = |\{\boldsymbol{x}_j \mid \boldsymbol{x}_j \in P \cup A \wedge \boldsymbol{f}(\boldsymbol{x}_i) \prec \boldsymbol{f}(\boldsymbol{x}_j)\}| \tag{2.53}$$

The raw fitness value r_i for particle with index i is then calculated as the sum of the strength of particles dominating it. This is stated formally in Equation (2.54).

$$r_i = \sum_{\boldsymbol{x}_j \in P \cup A, \boldsymbol{f}(\boldsymbol{x}_j) \prec \boldsymbol{f}(\boldsymbol{x}_i)} s_j \tag{2.54}$$

Further factor is added d_i is added to the raw fitness value to take into account population diversity. Value d_i is calculated in such a way that solutions in crowded areas of Pareto front are penalized. It is given formally in Equation (2.55). Here σ_i^k is defined as the distance to the k-th nearest particle to particle with index i. It is inverted since we want lower value of d_i to mean that particle with index i is in an unexplored area of the Pareto set. Value k is set to the square root of the size of population plus the size of the archive.

$$d_i = \frac{1}{\sigma_i^k + 2} \tag{2.55}$$

Finally the fitness value F_i for a particle with index i is calculated as given in Equation (2.54).

$$F_i = r_i + d_i \tag{2.56}$$

The ideas behind this MOPSO variant (more specifically the strength Pareto approach) are borrowed from SPEA2 method by E. Zitzler et al. [73].

- Personal best solution $\boldsymbol{p}_i$ is updated if one of the three conditions stated below are satisfied:
  1. The new solution $\boldsymbol{x}_i$ dominates old personal best solution $\boldsymbol{p}_i$.
  2. The new solution does not dominate the old personal best solution, but the new solution has lower values for the majority of objectives.
  3. If both conditions above are not satisfied and the new solution has the same number of lower objectives as the old personal best, personal best is updated with a 0.5 probability.
- Global best solution $\boldsymbol{g}_i$ is selected using tournament selection from the non-dominated point archive. Four solutions are taken at random from the archive and the solution with index i that has the lowest value of F_i is chosen. It is important to note that for this algorithm a different global best solution is chosen for each particle. This in practice means that the aforementioned tournament selection process will be repeated for each particle to select it's potentially unique $\boldsymbol{g}_i$.

Furthermore the method employs mutation and crossover operators. The operators are applied to both velocity and position vectors. The crossover vector is different in the way it is used in most evolutionary algorithms in that it is applied simultaneously to both velocity and position vectors. A single crossover point is used and the same point is used for both velocity and position vectors.

2.7.22 S. Z. Martínez et al. (2011)

S. Z. Martínez et al. [69] propose a decomposition based approach to multi objective particle swarm optimization. The authors propose to use Penalty Boundary Intersection (PBI) approach instead of Tchebycheff approach for calculating weighted aggregate for a given weight vector $\boldsymbol{\lambda}$ and reference vector $\boldsymbol{z}^*$. The problem then becomes that of minimizing single objective problem g defined in Equation (2.57).

$$g\left(\boldsymbol{x} \mid \boldsymbol{\lambda}, \boldsymbol{z}^*\right) = d_1 + \theta d_2 \tag{2.57}$$

$$d_1 = \frac{\left\|(\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{z}^*)^T \boldsymbol{\lambda}\right\|}{\|\boldsymbol{\lambda}\|} \tag{2.58}$$

$$d_2 = \left\|(\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{z}^*) - d_1 \frac{\boldsymbol{\lambda}}{\|\boldsymbol{\lambda}\|}\right\| \tag{2.59}$$

Double bars $\|.\|$ denote Euclidean norm. Weight vectors are selected from the $\left\{\frac{0}{H}, \frac{1}{H}, \ldots, \frac{H}{H}\right\}$ as in the previous decomposition based method by W. Peng et al. [49]. Weight vectors $\boldsymbol{\lambda}$ are all possible vectors of length m where each individual weight takes one of the values from the aforementioned set. The method uses an age threshold mechanism. If a particle does not imporve its personal solution for a specified number of iterations T_a its position, velocity, age and personal best solution are reinitialized. Position is reinitialized using the rule in Equation (2.60).

$$x_{ij} = N\left(\frac{g_{ij} - p_{ij}}{2}, |g_{ij} - p_{ij}|\right) \tag{2.60}$$

2.7.22.1 Changes to PSO update rules

The changes to personal and global best selection rules are given in the list below.

- Personal best $\boldsymbol{p}_i$ is updated if $g(\boldsymbol{x}_i \mid \boldsymbol{\lambda}_i, \boldsymbol{z}^*) \leq g(\boldsymbol{p}_i \mid \boldsymbol{\lambda}_i, \boldsymbol{z}^*)$. This is identical to how single objective personal update works, but the function optimized depends on the weight vector $\boldsymbol{\lambda}_i$.
- Global best $\boldsymbol{g}_i$ is selected at random from a set of global bests. The algorithm for calculating the set of potential global bests is simple: for each weight vector $\boldsymbol{\lambda}_i$ find the solution $\boldsymbol{x}^*$ that gives the lowest value of $g(\boldsymbol{x}^* \mid \boldsymbol{\lambda}_i, \boldsymbol{z}^*)$. Here $\boldsymbol{x}^* \in P \cup G$ where P is the corrent swarm population and G is the previous set of potential global bests.

If a particle's position coordinate goes outside the corresponding lower or upper boundary vector $\boldsymbol{a}$ or $\boldsymbol{b}$ coordinate equations (2.61) and (2.62) are used to update position and velocity. Boundary violating coordinate is simply set to the value of the corresponding boundary vector coordinate. Velocity is scaled by scalar value γ and multiplied by -1 so that the particle moves away from the boundary. Parameter $\gamma = 1$ in the study in question.

$$x_{ij} = \begin{cases} a_{ij} & \text{, if } x_{ij} < a_{ij} \\ b_{ij} & \text{, if } x_{ij} > b_{ij} \end{cases} \tag{2.61}$$

$$v_{ij} = -\gamma v_{ij} \tag{2.62}$$

2.7.22.2 Parameters

Parameters and their values as used by the authors of the method are summarized in Table 2.16. Inertia weight as well as social and personal experience coefficients are random number from given ranges. This is unusual in PSO in general but is used by several authors in MOPSO methods. Age threshold parameter T_a is used to determine when position and velocity reinitialization are to be applied. Parameter θ is a parameter of the aggregate function.

Parameter	Description	Value
θ	PBI penalty value	5
T_a	Age threshold	2
ρ_1	Personal experience influence	$U_{(1.2,2.0)}$
ρ_2	Social influence	$U_{(1.2,2.0)}$
w	Inertia coefficient	$U_{(0.1,0.5)}$

Table 2.16: Parameter values for M. Zapotecas et al. (2011) method.

2.7.22.3 Other comments

Due to the nature of this method the number of particles depends on the parameter H. For test problems with two objectives we used value of $H = 25$ which means there are 351 particles in the swarm. This means that to not exceed $300,000$ function evaluations we had to run the swarm for 855 iterations. For problems with three objectives we used value of $H = 12$ which means there are 364 particles in the swarm and we ran the swarm for 824 iterations. For the five objective case we used $H = 7$ and in turn 462 particles and ran the swarm for 649 iterations.

2.7.23 A. J. Nebro et al. (2013)

Antonio J. Nebro et al. [44] propose variants of their own SMPSO algorithm. That variant was described in section 2.7.18. These variants differ from their original algorithm which was described here previously in the selection strategies for global best (leader) particle. The variants are given below.

SMPSO_r In this case the global leader is simply chosen at random from the nondominated point archive A. This is included as a sanity check, any more complicated variant is noth worth it if it cannot outperform this one.

cellSMPSO This variant is based on concepts from MOCell algorithm by Antonio J. Nebro et al. [43]. The description of this variant is not very clear. For 50% of the particles the original SMPSO leader selection scheme is used. For the other half eight neighboring particles in the swarm are considered

and the leader is selected from their personal bests, presumably at random. We will also assume that the eight neighbors are selected by their closeness in terms of Euclidean distance in solution space from the particle in question.

SMPSOhv This variant uses contribution to the hypervolume when selecting leader solutions from the nondominated point archive. It is not clear exactly what do the authors mean by contribution to the hypervolume. It incorporates this measure in two ways. First of all if the nondominated point archive becomes full (the number of points in it reaches the limit set by the user) the point that contributes the least to the hypervolume is removed and the new point replaces it. When selecting the global best particle two particles are chosen at random from the archive and the one with larger contribution to the hypervolume is selected.

SMPSOhv_r This variant differs from the previous way in that the leader is selected at random. The archive updates work as in the variant below. It is similar to **SMPSO_r** except the archive uses hypervolume contribution coefficient instead of crowding distance.

Everything else is the same as in the algorithm described in section 2.7.18 so it is not discussed here in detail. The changes to that algorithm are fully explained in the list above.

2.7.24 K. S. Lim et al. (2013)

Kian Sheng Lim et al. [37] propose an approach based on vector evaluated swarm optimization. Each objective gets a separate swarm. Each swarm optimizes it's objective. Swarms get global bests from their neighbour swarms. Neighbours are defined by a ring topology where the neighbour is the preceding swarm if they are ordered in a list and the first swarm has the last swarm as its neighbour. Personal bests are updated as in standard single objective PSO via simple arithmetic comparison. In this variant global best is chosen in a different way than other vector evaluated particle swarm algorithms. It is chosen from the nondominated point archive A. Suppose swarm S_j optimizes objective j.

It will then choose the solution from nondominated point archive A that is lowest in terms of objective $j-1$. This is different from other vector evaluated PSOs in that dominance relations are also taken in to account. This variant uses polynomial mutation operator as given in Equation (2.63). Here we show how a position vector of a particle $\boldsymbol{x}$ is updated.

$$x_i \leftarrow x_i + (b_i - a_i)\delta_i \tag{2.63}$$

where

$$\delta_i \leftarrow \begin{cases} (2r_i)^{\frac{1}{\eta_m+1}} - 1 & \text{if } r_k < 0.5 \\ 1 - (2(1-r_k))^{\frac{1}{\eta_m+1}} & \text{if } r_k \geq 0.5 \end{cases} \tag{2.64}$$

2.7.24.1 Changes to PSO update rules

The changes to personal and global best update rules are summarized in the list below.

- Personal best $\boldsymbol{p}_i$ is updated if the new solution gives lower value for that subswarms objective.
- Global best $\boldsymbol{g}_i$ is chosen from the non-dominated point archive. It is the solution that has the lowest value of the neighbouring subswarms objective.

2.7.24.2 Parameters

Parameters and their values as used by the authors are summarized in Table 2.17. Algorithm parameter w is reduced linearly from 1 to 0.4 during the evolution of the swarm. Parameters ρ_1 and ρ_2 are set to random values between 1.5 and 2.5. Distribution parameter for polynomial mutation η_m is set to 0.5. Mutation probability is proportional to the dimensionality of the problem. It is designed so that one coordinate per particle would be mutated each iteration on average.

Parameter	Description	Value
p_m	Mutatation probability per coordinate	$\frac{1}{d}$
η_m	Polynomial mutation distribution parameter	0.5
ρ_1	Personal experience influence	$U_{(1.5,2.5)}$
ρ_2	Social influence	$U_{(1.5,2.5)}$
w	Inertia coefficient	1.0 to 0.4

Table 2.17: Parameter values for K. S. Lim et al. (2013) method.

2.7.25 I. C. Garcia et al. (2014)

Ivan Chaman Garcia et al. [19] propose a hypervolume-based multi-objective particle swarm optimizer. The differences between our generic MOPSO and this algorithm are described below.

1. Global best $\boldsymbol{g}_i$ is selected at random from the top 2% of the archive sorted by decreasing hypervolume contribution coefficient. This way solutions corresponding to least explored sections of the Pareto frontier are hopefully selected.
2. Personal best $\boldsymbol{p}_i$ is selected at random from the remaining 98% of solutions in the nondominated point archive.
3. Parameter w is set to 0.5, ρ_1 and ρ_2 are set to 1. Mutation rate m_r is set to 0.5.

This algorithm uses the same mutation operator as Carlo R. Raquel et al. [52]. This mutation operator is given in Algorithm 6. It is important to note that mutation rate m_r in this case signifies a parameter in mutation algorithm. It does not stand for the probability with which mutation is applied to each coordinate of particle's position.

The concept of hypervolume contribution is defined as the amount of hypervolume that a solution contributes. Namely how much of the hypervolume is covered by that particular solution and is not intersected by any other solution. Computation of hypervolume also needs a reference point. This reference point (called nadir point in case of minimization) is updated during each iteration by

simply taking the lowest value of each objective. How to calculate this point is shown in Equation (2.65).

$$\boldsymbol{z}_{ref} = \left(\min_{\boldsymbol{x} \in PF} f_1(\boldsymbol{x}), \ldots, \min_{\boldsymbol{x} \in PF} f_k \right) \tag{2.65}$$

Hypervolume contribution is illustrated graphically in Figure 2.7 where the areas of the gray paches to the bottom left of the PF points correspond to the hypervolume contribution of that point.

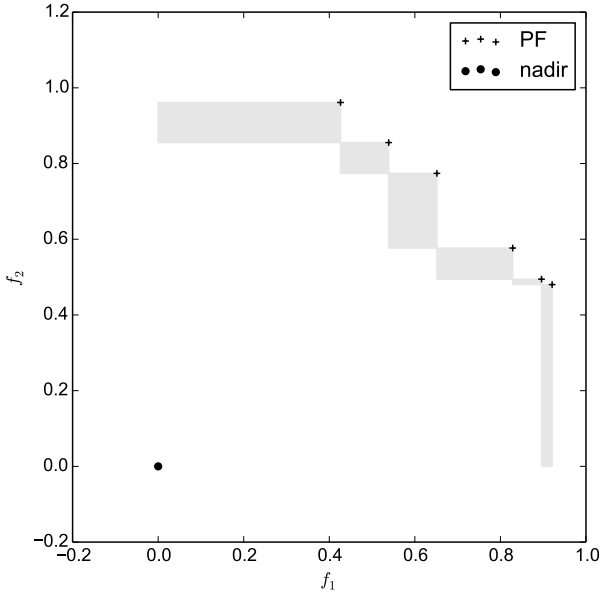


Figure 2.7: Hypervolume contribution. The areas of gray patches to the bottom left from the PF approximation points correspond to their hypervolume contribution in the two objective case.

2.8 Conclusions

In this chapter the problem of multi-objective optimization was formally described. In the problem with several, possibly contradictory, objectives the goal is to find a set of non-dominated points called the Pareto frontier of that problem. The solutions which correspond to points on the Pareto frontier are then used

to make a decision based on the current priorities of the decision maker. Exact methods for solving multi-objective problems are not known in the general case. Exact methods exist for some subsets of the problem, for example, for certain multi-objective problems with convex Pareto frontiers. When nothing or little is known about the problem one has to rely on optimization metaheuristics, such as Genetic Algorithms or Particle Swarm Optimization. We discuss the limitations of several popular multi-objective optimization methods such as the weighted aggregate approach. From these limitations it is apparent that the use of metaheuristics is sometimes necessary.

We also provide a review of existing multi-objective PSO methods. They are analyzed with regards to their implementation details. These details are: the core idea behind them, how they ensure Pareto point diversity in the approximation and whether mutation operators are used. There were attempts in classifying existing MOPSO methods. See, for example, one by Margarita Reyes-Sierra et al. [53], however, the approach used there considers only very broad principles behind the methods and the relations between methods and the ideas used in them are not very clear. Therefore, a new classification method would be of benefit that takes into consideration the findings of the review. The methods in literature were never experimentally evaluated with regards to one another using an extensive set of test-problems. This means that the relative performance of these methods compared to one another is not known. This means there is a need for such a study to help researchers to understand the relative merits of these methods better.

To measure the performance of optimization methods it is necessary to have a diverse set of test problems. These test problems should model a wide variety of possible real-world problems. Such test problems are described here. This includes many diffent types of problems – including non-convex and discontinuous. The problems have 2, 3 and 5 objectives. The Pareto frontiers for these problems are apriori known. This allows to quantify the optimizer performance easily by comparing the approximations to real Pareto frontiers.

Methods designed to solve multi-objective problems usually return a discrete approximation of that frontier. This approximation can be evaluated if the Pareto Frontier of the problem is known in advance in terms of similarity to that frontier.

Similarity can be measured using a variety of metrics.They can alternatively be evaluated relatively to one another (for example by using hypervolume comparison). Several performance indicators that were designed to this end were described in this chapter. We will argue at length in the following chapter that they are inadequate when measuring Pareto solution uniformity – that is how extensively and uniformly the points in the approximation cover the Pareto frontier. Therefore, new performance indicators are needed.

Another area of understanding MOPSO methods that is currently lacking ,is what kind of MOPSO approaches work best for what type of multi-objective problems. The problems can be different in their Pareto frontier (discontinuous or discrete, convex, non-convex, etc.), number of objectives and other properties. There are several diffent core principles which are used to implement new MOPSO methods. From the experimental analysis with different types of problems it should be clearer what kinds of methods work best with which kinds of problems.

It is apparent from the survey of existing methods that heterogeneous approaches are absent. These approaches show great promise in the field of single objective Particle Swarm Optimization. Therefore, it seems to be a good idea to investigate their use in multi-objective optimization. Heterogeneous approaches work by using different types of particles (particles with different velocity update rules, etc.) in the same swarm, solving the same problem. Since they all share the information (via the same non-dominated point archive or otherwise) it can be hoped that the particles will outweight each other's disadvantages when confronted with a particular problem. One of the issues here is to decide how the types of particles will be chosen, whether they may be changed dynamically during the optimization run and if so, what are the rules for changing a particlesÂŠtype.It will be answered in the following chapters where two new heterogeneous MOPSO methods are proposed.

Most sources in literature do not include the information about publicly available implementations of these methods. Because of this, there is no easy way to independently test existing MOPSO methods. Therefore, it seems to be of importance to provide such implementations in an open-source code base. It seems that a good solution to this problem is to provide a software framework

that will allow users to implement new particle swarm optimization methods easily and test them. Such a framework is proposed in this thesis and allows users to implement new MOPSO methods with minimal changes to the code, test them using numerous included test problems and evaluate the results using included performance indicators.

Chapter 3

Experimental Analysis of Existing MOPSO Methods

In this chapter an experimental evaluation of existing MOPSO methods is presented. The goal of the study is to identify which types of methods are suitable for which problems. Also to systematize the knowledge of existing MOPSO methods. To this end experiments were performed for each of the methods described in the previous section. All the algorithms were tested with all test problems from the previous section. Multiple runs (30 runs for each method and problem combination) were performed. Results have been collected and they are presented graphically in this section.

3.1 Experimental Procedure

We used the same procedure to test each algorithm configuration.An algorithm configuration in this case means an optimization method implementation along with its parameters — for example, PSO with topology description and c_1 and c_2 values, mutation rates.

For each test problem repeat the following 30 times:

1. Run the algorithm for 300,000 function evaluations. One function evalu-

ation will consist of evaluating every objective of the problem.

2. Collect an approximation of the Pareto frontier obtained by the algorithm and store it in a file.

3. Calculate the mean values of performance indicators I_{GD}, I_{IGD}, I_{SP} and I_N and store them in tables.

4. Calculate the mean values of performance indicators I_{UNI^1} and I_{UNI^2}.

The above procedure summarizes the protocol followed for every algorithm. In the end we have the results of 30 runs for every given problem and algorithm combination. We will analyze these results in the following chapter. While the test problems have already been described in detail, here we will provide a reference and a short discussion of their properties.

Name	Objectives	RS Size	Geometry
UF01	2	1000	Convex
UF02	2	1000	Convex
UF03	2	1000	Convex
UF04	2	1000	Concave
UF05	2	21	Discrete
UF06	2	1000	Concave
UF07	2	1000	Concave
UF08	3	10000	Concave
UF09	3	10000	Concave
UF10	3	10000	Concave
UF11	5	5000	Concave
UF12	5	5000	Concave
UF13	5	5000	Convex

Table 3.1: Test problem reference table.

The reference is given in Table 3.1. It can be seen that the problems cover a wide range and include the problems with 2, 3 and 5 objectives. It also includes convex, concave and discrete problems with non-convex problems making up the majority. This makes sense since convex problems can be solved using weighted aggregation of objectives and ,thus more involved methods are seldom required. One of the problems with this set is the low number of reference points for 5 objective problems.

3.2 Test Problems for Multi-Objective Optimization

In this section we describe the test problems used in the work to perform experiments. Their definitions and properties are briefly described. For each of these problems Pareto Set and Pareto Frontier are both known which is convenient for our study. In all the problems below the dimensionality d is set to 30. In all these cases the problems is that of minimizing the objectives. These problems were compiled by Quingfu Zhang et al. [71] for use in the CEC 2009 multi-objective optimization algorithm competition.

3.2.1 CEC 2009 Unconstrained Problem 1

The two objectives are:

$$f_1(\vec{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} \left(x_j - sin\left(6\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.1}$$

$$f_2(\vec{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} \left(x_j - sin\left(6\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.2}$$

With J_1 and J_2 defined below.

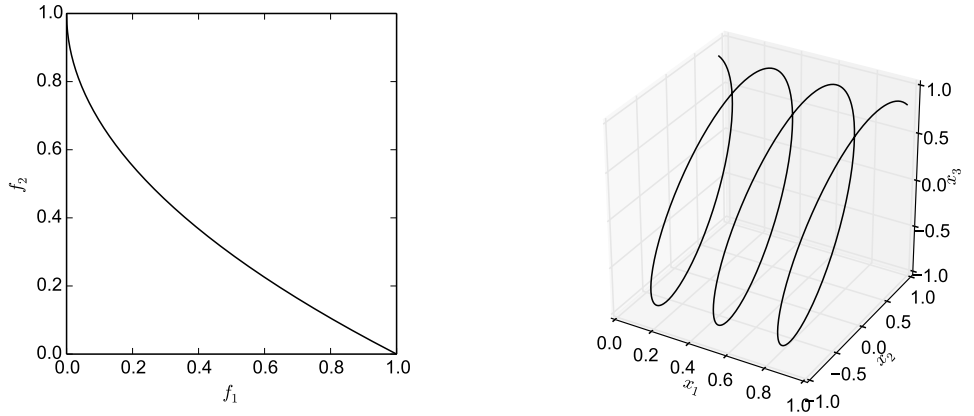
$$J_1 = \{j \mid j \text{ is odd and } 2 \leq j \leq d\}$$

$$J_2 = \{j \mid j \text{ is even and } 2 \leq j \leq d\}$$

The search space for this problem is $[0, 1] \times [-1, 1]^{d-1}$. Pareto Frontier and Pareto Set (PF and PS) are given in the two equations below.

$$f_2 = 1 - \sqrt{f_1}, \quad 0 \leq f_1 \leq 1$$

$$x_j = sin\left(6\pi x_1 + \frac{j\pi}{d}\right), \quad j = 2, \ldots, d, \quad 0 \leq x_1 \leq 1$$



(a) Pareto front.

(b) Pareto set when $d = 3$.

Figure 3.1: Pareto set and Pareto front of Unconstrained Problem 1.

3.2.2 CEC 2009 Unconstrained Problem 2

The two objectives are:

$$f_1(\vec{x}) = x_1 + \frac{2}{|J_1|} \sum_{j \in J_1} y_j^2 \tag{3.3}$$

$$f_2(\vec{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|} \sum_{j \in J_2} y_j^2 \tag{3.4}$$

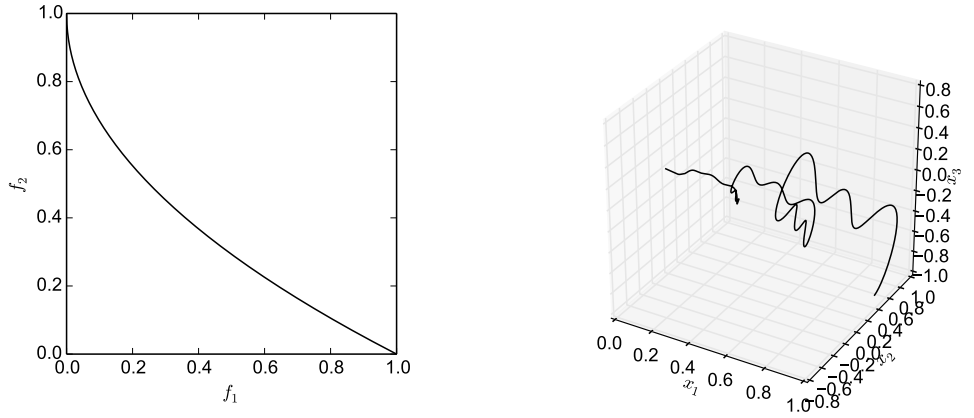
$$y_j = \begin{cases} x_j - \left(0.3x_1^2 cos\left(24\pi x_1 + \frac{4j\pi}{d}\right) + 0.6x_1\right) cos\left(6\pi x_1 + \frac{j\pi}{d}\right) & j \in J_1 \\ x_j - \left(0.3x_1^2 cos\left(24\pi x_1 + \frac{4j\pi}{d}\right) + 0.6x_1\right) sin\left(6\pi x_1 + \frac{j\pi}{d}\right) & j \in J_2 \end{cases} \tag{3.5}$$

With J_1 and J_2 defined below.

$$J_1 = \{j \mid j \text{ is odd and } 2 \leq j \leq d\}$$

$$J_2 = \{j \mid j \text{ is even and } 2 \leq j \leq d\}$$

The search space for this problem is $[0, 1] \times [-1, 1]^{d-1}$. Pareto Frontier and Pareto



(a) Pareto front.

(b) Pareto set when $d = 3$.

Figure 3.2: Pareto set and Pareto front of Unconstrained Problem 2.

Set (PF and PS) are given in the two equations below.

$$f_2 = 1 - \sqrt{f_1}, \quad 0 \leq f_1 \leq 1$$

$$x_j = \begin{cases} \left(0.3x_1^2 cos\left(24\pi x_1 + \frac{4j\pi}{d}\right) + 0.6x_1\right) cos\left(6\pi x_1 + \frac{j\pi}{d}\right) & j \in J_1 \\ \left(0.3x_1^2 cos\left(24\pi x_1 + \frac{4j\pi}{d}\right) + 0.6x_1\right) sin\left(6\pi x_1 + \frac{j\pi}{d}\right) & j \in J_2 \end{cases}$$

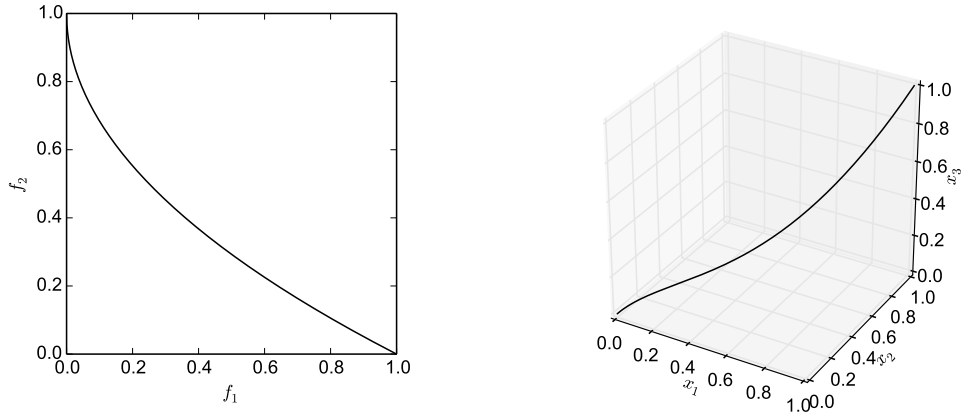
3.2.3 CEC 2009 Unconstrained Problem 3

The two objectives are:

$$f_1(\vec{x}) = x_1 + \frac{2}{|J_1|}\left(4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1} cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right) \tag{3.6}$$

$$f_2(\vec{x}) = 1 - \sqrt{x_1} + \frac{2}{|J_2|}\left(4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2} cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right) \tag{3.7}$$

$$y_j = x_j - x_1^{0.5\left(1.0+\frac{3(j-2)}{d-2}\right)}, \quad j = 2, \ldots, d \tag{3.8}$$



(a) Pareto front.

(b) Pareto set when $d = 3$.

Figure 3.3: Pareto set and Pareto front of Unconstrained Problem 3.

J_1 and J_2 are defined in the same way as in previous problems.

The search space for this problem is $[0, 1]^d$. Pareto Frontier and Pareto Set (PF and PS) are given in the two equations below.

$$f_2 = 1 - \sqrt{f_1}, \quad 0 \leq f_1 \leq 1$$

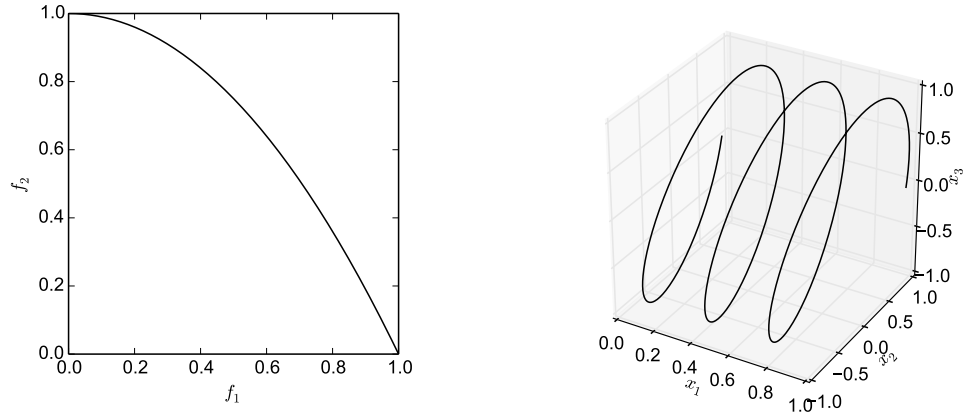
$$x_j = x_1^{0.5\left(1.0+\frac{3(j-2)}{d-2}\right)}, \quad j = 2, \ldots, d, \quad 0 \leq x_1 \leq 1$$

3.2.4 CEC 2009 Unconstrained Problem 4

The two objectives f_1 and f_2 are given in the equations below.

$$f_1(\vec{x}) = x_1 + \frac{2}{|J_1|}\left(\sum_{j \in J_1} h(y_j)\right) \tag{3.9}$$

$$f_2(\vec{x}) = 1 - x_1^2 + \frac{2}{|J_2|}\left(\sum_{j \in J_2} h(y_j)\right) \tag{3.10}$$



(a) Pareto front.

(b) Pareto set when $d = 3$.

Figure 3.4: Pareto set and Pareto front of Unconstrained Problem 4.

$$y_i = x_j - sin\left(6\pi x_1 + \frac{j\pi}{d}\right), \quad j = 2, \ldots, d \tag{3.11}$$

$$h(t) = \frac{|t|}{1 + e^{2|t|}} \tag{3.12}$$

J_1 and J_2 are defined in the same way as in previous problems.

The search space for this problem is $[0, 1] \times [-2, 2]^{d-1}$. Pareto Frontier and Pareto Set (PF and PS) are given in the two equations below.

$$f_2 = 1 - f_1^2, \quad 0 \leq f_1 \leq 1$$

$$x_j = sin\left(6\pi x_1 + \frac{j\pi}{d}\right), \quad j = 2, \ldots, d, \quad 0 \leq x_1 \leq 1$$

3.2.5 CEC 2009 Unconstrained Problem 5

The two objectives f_1 and f_2 are given in the equations below.

$$f_1(\vec{x}) = x_1 + \left(\frac{1}{2N} + \epsilon\right)|\sin(2N\pi x_1)| + \frac{2}{|J_1|}\sum_{j\in J_1} h(y_j) \tag{3.13}$$

$$f_2(\vec{x}) = 1 - x_1 + \left(\frac{1}{2N} + \epsilon\right)|\sin(2N\pi x_1)| + \frac{2}{|J_2|}\sum_{j\in J_2} h(y_j) \tag{3.14}$$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{d}\right), \quad 2, \ldots, d \tag{3.15}$$

$$h(t) = 2t^2 - \cos(4\pi t) + 1 \tag{3.16}$$

In the equations above N is an integer and $\epsilon > 0$, J_1 and J_2 are defined the same way as in the problems above. We set these values to $N = 10$ and $\epsilon = 0.1$, same as in CEC 09 algorithm contest.

The search space for this problem is $[0, 1] \times [-1, 1]^{d-1}$. The PF for this problem consists of $2N + 1$ Pareto Optimal solutions that have the form:

$$\left(\frac{i}{2N}, 1 - \frac{i}{2N}\right), \quad i = 0, 1, \ldots, 2N$$

3.2.6 CEC 2009 Unconstrained Problem 6

The two objectives f_1 and f_2 are given in the equations below.

$$\begin{aligned} f_1(\vec{x}) = x_1 + \max\left\{0, 2\left(\frac{1}{2N} + \epsilon\right)\sin(2N\pi x_1)\right\} + \\ + \frac{2}{|J_1|}\left(4\sum_{j\in J_1} y_j^2 - 2\prod_{j\in J_1}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right) \end{aligned} \tag{3.17}$$

$$f_2(\vec{x}) = 1 - x_1 + \max\left\{0, 2\left(\frac{1}{2N} + \epsilon\right)\sin\left(2N\pi x_1\right)\right\} + \\ + \frac{2}{|J_2|}\left(4\sum_{j\in J_2} y_j^2 - 2\prod_{j\in J_2}\cos\left(\frac{20y_j\pi}{\sqrt{j}}\right) + 2\right) \tag{3.18}$$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{d}\right), \quad j = 2, \ldots, d \tag{3.19}$$

In the equations above N is an integer and $\epsilon > 0$, J_1 and J_2 are defined the same way as in the problems above. We set these values to $N = 2$ and $\epsilon = 0.1$, same as in CEC 09 algorithm contest.

The search space for this problem is $[0, 1] \times [-1, 1]^{d-1}$. The PF for this problem consists of one isolated point $(0, 1)$ and N disconnected parts of the form:

$$f_2 = 1 - f_1, \quad f_1 \in \bigcup_{i=1}^{N}\left[\frac{2i-1}{2N}, \frac{2i}{2N}\right]$$

3.2.7 CEC 2009 Unconstrained Problem 7

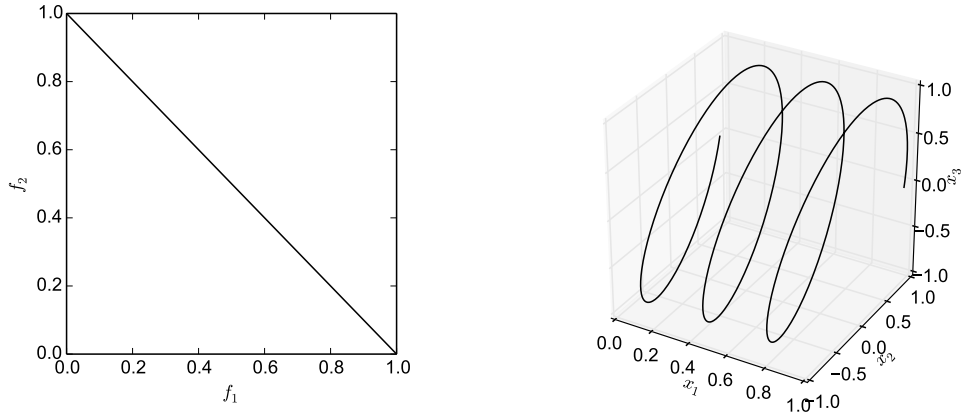
The two objectives f_1 and f_2 are given in the equations below.

$$f_1 = \sqrt[5]{x_1} + \frac{2}{|J_1|}\sum_{j\in J_1} y_j^2 \tag{3.20}$$

$$f_1 = 1 - \sqrt[5]{x_1} + \frac{2}{|J_2|}\sum_{j\in J_2} y_j^2 \tag{3.21}$$

$$y_j = x_j - \sin\left(6\pi x_1 + \frac{j\pi}{d}\right), \quad j = 2, \ldots, d \tag{3.22}$$

In these equations J_1 and J_2 are defined the same way as in the problems above.



(a) Pareto front.

(b) Pareto set when $d = 3$.

Figure 3.5: Pareto set and Pareto front of Unconstrained Problem 7.

The search space for this problem is $[0, 1] \times [-1, 1]^{d-1}$. The PF and PS are given in the equations below.

$$f_2 = 1 - f_1, \quad 0 \leq f_1 \leq 1$$

$$x_j = \sin\left(6\pi x_1 + \frac{j\pi}{d}\right), \quad j = 2, \ldots, d, \quad 0 \leq x_1 \leq 1$$

3.2.8 CEC 2009 Unconstrained Problem 8

The three objectives of this problem f_1, f_2 and f_3 are given in the equations below.

$$f_1(\vec{x}) = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \\ + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.23}$$

$$f_2(\vec{x}) = \cos(0.5x_1\pi)\sin(0.5x_2\pi)+ \\ +\frac{2}{|J_2|}\sum_{j\in J_2}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.24}$$

$$f_3(\vec{x}) = \sin(0.5x_1\pi)+ \\ +\frac{2}{|J_3|}\sum_{j\in J_3}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.25}$$

Here J_1, J_2 and J_3 are defined as given in the equations below.

$$J_1 = \{j \mid 3 \leq j \leq d \text{ and } j-1 \text{ is a multiple of } 3\}$$

$$J_2 = \{j \mid 3 \leq j \leq d \text{ and } j-2 \text{ is a multiple of } 3\}$$

$$J_3 = \{j \mid 3 \leq j \leq d \text{ and } j \text{ is a multiple of } 3\}$$

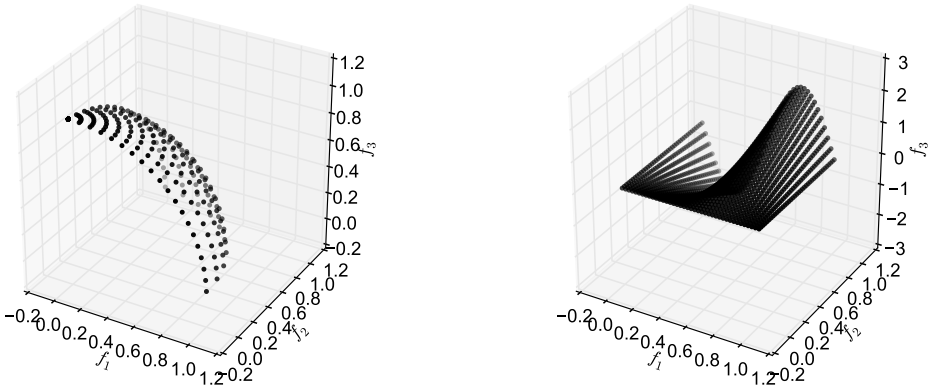
The search space for this problem is $[0,1]^2 \times [-2,2]^{d-2}$. The PF and PS are given in the equations below.

$$f_1^2 + f_2^2 + f_3^3 = 1, \quad 0 \leq f_1, f_2, f_3 \leq 1$$

$$x_j = 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right), \quad j = 3,\ldots,d$$

3.2.9 CEC 2009 Unconstrained Problem 9

The three objectives of this problem f_1, f_2 and f_3 are given in equations below.



(a) Pareto front.

(b) Pareto set when $d = 3$.

Figure 3.6: Pareto set and Pareto front of Unconstrained Problem 8.

$$f_1(\vec{x}) = 0.5\left(\max\left\{0, (1+\epsilon)\left(1-4\left(2x_1-1\right)^2\right)\right\} + 2x_1\right)x_2 + \\ + \frac{2}{|J_1|}\sum_{j\in J_1}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.26}$$

$$f_2(\vec{x}) = 0.5\left(\max\left\{0, (1+\epsilon)\left(1-4\left(2x_1-1\right)^2\right)\right\} - 2x_1 + 2\right)x_2 + \\ + \frac{2}{|J_2|}\sum_{j\in J_2}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.27}$$

$$f_3(\vec{x}) = 1 - x_2 + \\ + \frac{2}{|J_3|}\sum_{j\in J_3}\left(x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right)\right)^2 \tag{3.28}$$

Sets J_1, J_2 and J_3 are defined the same way as in Unconstrained Problem 8 and $\epsilon = 0.1$.

The search space for this problem is $[0,1]^2 \times [-2,2]^{d-2}$.

The PF has two parts where the first part is

$$\begin{aligned} 0 \leq f_3 \leq 1, \\ 0 \leq f_1 \leq \frac{1}{4}(1 - f_3), \\ f_2 = 1 - f_1 - f_3 \end{aligned}$$

and the second part is

$$\begin{aligned} 0 \leq f_3 \leq 1, \\ \frac{3}{4}(1 - f_3) \leq f_1 \leq 1, \\ f_2 = 1 - f_1 - f3. \end{aligned}$$

The PS also has two disconnected parts where the first part is

$$x_1 \in [0, 0.25] \cup [0.75, 1], \quad 0 \leq x_2 \leq 1$$

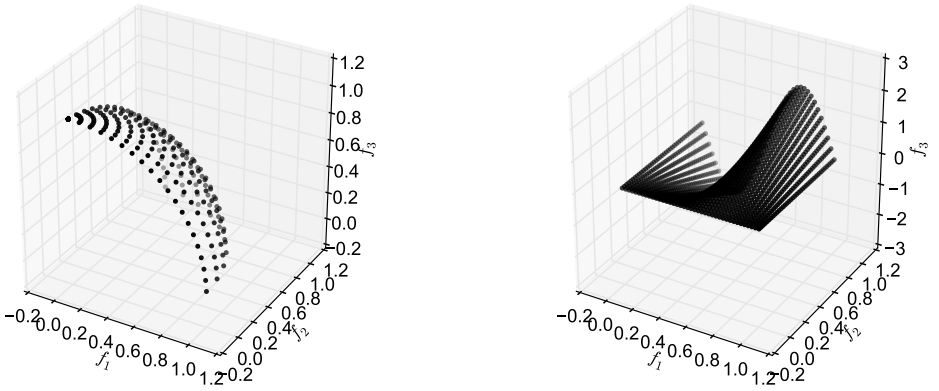
and the second one

$$x_j = 2x_2 \sin\left(2\pi x_1 + \frac{j\pi}{d}\right), \quad j = 3, \ldots, d.$$

3.2.10 CEC 2009 Unconstrained Problem 10

The three objectives of this problem f_1, f_2 and f_3 are given in equations below.

$$f_1(\vec{x}) = \cos(0.5x_1\pi)\cos(0.5x_2\pi) + \frac{2}{|J_1|}\sum_{j \in J_1}\left(4y_j^2 - \cos(8\pi y_j) + 1\right) \tag{3.29}$$



(a) Pareto front.

(b) Pareto set when $d = 3$.

Figure 3.7: Pareto set and Pareto front of Unconstrained Problem 10.

$$f_2(\vec{x}) = \cos(0.5x_1\pi)\sin(0.5x_2\pi) + \frac{2}{|J_2|}\sum_{j\in J_2}\left(4y_j^2 - \cos(8\pi y_j) + 1\right) \tag{3.30}$$

$$f_3(\vec{x}) = \sin(0.5x_1\pi) + \frac{2}{|J_3|}\sum_{j\in J_3}\left(4y_j^2 - \cos(8\pi y_j) + 1\right) \tag{3.31}$$

$$y_j = x_j - 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right), \quad j = 3,\ldots,d \tag{3.32}$$

Here, sets J_1, J_2 and J_3 are defined the same way as in previous two problems.

The search space for this problem is $[0,1]^2 \times [-2,2]^{d-2}$. The PF and PS are given in the equations below. The PF and PS for this problem are given below.

$$f_1^2 + f_2^2 + f_3^3 = 1, \quad 0 \leq f_1, f_2, f_3, \leq 1$$

$$x_j = 2x_2\sin\left(2\pi x_1 + \frac{j\pi}{d}\right), \quad j = 3,\ldots,d$$

3.2.11 CEC 2009 Unconstrained Problem 11

This is one of the three five objective problems we used and that was used in CEC 2009 contest. This one is based on a problem from the famous DTLZ test problem suite that can be found for example in this work by V. L. Huang et al. [24] or Simon Huband et al. [25].

$$
\begin{aligned}
f_1(\vec{x}) &= \begin{cases} (1+g(\vec{x}_M)) \prod_{i=1}^{M-1} \cos\left(z'_i \frac{\pi}{2}\right) + 1 & \text{if } z_i \geq 0 \\ S(psum_1)\,(1+g(\vec{x}_M)) \left(\prod_{i=1}^{M-1} \cos\left(z'_i \frac{\pi}{2}\right) + 1\right) & \text{otherwise} \end{cases} \\
f_2(\vec{x}) &= \begin{cases} (1+g(\vec{x}_M)) \prod_{i=1}^{M-2} \cos\left(z'_i \frac{\pi}{2}\right) \sin\left(z'_{M-1} \frac{\pi}{2}\right) + 1 & \text{if } z_i \geq 0 \\ S(psum_2)\,(1+g(\vec{x}_M)) \left(\prod_{i=1}^{M-2} \cos\left(z'_i \frac{\pi}{2}\right) \sin\left(z'_{M-1} \frac{\pi}{2}\right) + 1\right) & \text{otherwise} \end{cases} \\
f_3(\vec{x}) &= \begin{cases} (1+g(\vec{x}_M)) \prod_{i=1}^{M-3} \cos\left(z'_i \frac{\pi}{2}\right) \sin\left(z'_{M-2} \frac{\pi}{2}\right) + 1 & \text{if } z_i \geq 0 \\ S(psum_3)\,(1+g(\vec{x}_M)) \left(\prod_{i=1}^{M-3} \cos\left(z'_i \frac{\pi}{2}\right) \sin\left(z'_{M-2} \frac{\pi}{2}\right) + 1\right) & \text{otherwise} \end{cases} \\
&\vdots \\
f_{M-1}(\vec{x}) &= \begin{cases} (1+g(\vec{x}_M)) \cos\left(z'_1 \frac{\pi}{2}\right) \sin\left(z'_2 \frac{\pi}{2}\right) + 1 & \text{if } z_i \geq 0 \\ S(psum_{M-1})\,(1+g(\vec{x}_M)) \left(\cos\left(z'_1 \frac{\pi}{2}\right) \sin\left(z'_2 \frac{\pi}{2}\right) + 1\right) & \text{otherwise} \end{cases} \\
f_M(\vec{x}) &= \begin{cases} (1+g(\vec{x}_M)) \sin\left(z'_1 \frac{\pi}{2}\right) + 1 & \text{if } z_i \geq 0 \\ S(psum_M)\,(1+g(\vec{x}_M)) \left(\sin\left(z'_1 \frac{\pi}{2}\right) + 1\right) & \text{otherwise} \end{cases}
\end{aligned}
$$

Where

$$
g(\vec{x}_M) = \sum_{x_i \in \vec{x}_M} (z_i - 0.5)^2 \tag{3.34}
$$

$$
z'_i = \begin{cases} -\lambda_i z_i, & z_i < 0 \\ z_i, & 0 \leq z_i \leq 1 \\ \lambda_i z_i, & z_i > 1 \end{cases} \tag{3.35}
$$

$$p_i = \begin{cases} -z_i, & z_i < 0 \\ 0, & 0 \leq z_i \leq 1 \\ z_i - 1, & z_i > 1 \end{cases} \tag{3.36}$$

$$\vec{z} = \mathbf{M}\vec{x} \tag{3.37}$$

$$S(psum) = \frac{2}{1 + e^{-psum}} \tag{3.38}$$

Data vector λ and matrix $\mathbf{M}$ used in this problem can be found in files.

3.2.12 CEC 2009 Unconstrained Problem 12

The definition of this problem is the same as Problem 11, but $g(\vec{x}_M)$ is defined differently. Definition is given in Equation (3.39).

$$g(\vec{x}_M) = 100 \left(|\vec{x}_M| + \sum_{x_i \in \vec{x}_M} (z_i - 0.5)^2 - \cos(20\pi(z_i - 0.5)) \right) \tag{3.39}$$

3.2.13 CEC 2009 Unconstrained Problem 13

The problem is, given $\boldsymbol{z} = \{z_1, z_2, \ldots, z_d\}$, to minimize the equation given below where $m \in \{1, \ldots, M\}$ is the objective index and M is the number of objectives.

$$f_m(\boldsymbol{x}) = Dx_M + S_m h_m(x_1, \ldots, x_{M-1}) \tag{3.40}$$

Further explanation of this problem and it's parameter is complicated and will not be included here, but can be found in Q. Zhang et al. [71].

3.3 Quality Indicators

The purpose of quality indicators is to assign numerical values to various aspects of optimizer performance. Usually what is measured is the similarity between an approximated Pareto front produced by an optimization algorithm to a reference Pareto front produced using a priori knowledge of what the real Pareto front for that particular problem is. These reference sets are usually produced using the explicit function of the Pareto front and you are expected to assume for them to contain enough points that furthermore uniformly cover the real Pareto front. In practice however it is not always the case that either of those presuppositions is true. Popular quality indicators are given below. Measuring optimizer performance in the case of single objective optimization is simple. If minimizing a function and one solution gives lower value of that function then another that solution in almost unambigously better. Although it can be argued that a solution that will lead you to find the global minimum is better than a solution that will lead you to find a local minimum even if the former gives a larger function value. However comparing the end result of an optimizer run is trivial - lower is always better. Further aspects of performance such as convergence time can be measured by tracking best solution found after each algorithm iteration. When measuring multi-objective optimization algorithm performance the matter is more complicated since you have to compare two sets instead of just single values. Measuring distance between two sets is relatively simple with several quality indicators based on calculating average distance between points in one set and the closest point in another set. These include GD, IGD and Δ_p indicators. Popular quality indicators like IGD try to assign a single number to the abstract notion of optimizer performance. Good overviews of quality indicators for multi-objective optimiziation are provided by Eckart Zitzler et al. [74] and Jin Wu et al. [68]

We have selected four indicators to be used in our experiments. While this is but a small sample of indicators that are available we feel this is a good representation. We also propose two novel performance indicators designed to measure solution spread in approximations if reference set is available, which in our case it is. The existing performance indicators we selected are popular and used in numerous publications. For example IGD was the sole measure of

merit used for CEC 2009 multi-objective algorithm competition.

3.3.1 Generational Distance (GD)

Generational distance is defined as given in Equation (3.41). Here, and in the explanations of other metrics, A is the approximate set to the PF, P^* is a set of uniformly distributed points along the PF (in the objective space) and $d(v, P^*)$ is the minimum distance between point v and one of the points in P^*.

$$GD(A, P^*) = \frac{\sum_{v \in A} d(v, P^*)}{|A|} \tag{3.41}$$

This metric measures how close the approximate set is to the real PF. It is important to note that as long as the approximate set A lies close to the real PF this metric will report a value close to zero even if large portions of the PF are missing in the approximation. This is a well known measure and can be found for example in David A. Van Veldhuizen et al. [64].

3.3.2 Inverted Generational Distance (IGD)

This metric is, in a sense, the opposite of the previous one. It measures how close actual PF is to the approximated one. For this we simply swap the sets in Equation (3.41) places as given in Equation (3.42).

$$IGD(A, P^*) = \frac{\sum_{v \in P^*} d(v, A)}{|P^*|} \tag{3.42}$$

This metric will give lower values for approximations that are close to the actual PF and furthermore no large portions of the PF can be missing in approximation. Because of this, this metric is sometimes used as the only figure of merit when comparing multi-objective optimization algorithms on problems where PF is known. This metric can be said to measure both closeness to the real PF and uniformity of approximation. Description of the use of this metric can be found, for example, in Hui Li et al. [35]. There is however a downside when using this

metric as the only measure of merit for multi-objective optimization algorithms. It is not clear to what extent do the two properties influence the value of this metric. It is also not clear what is their exact relationship when calculating IGD. Therefore it may be a better idea to instead use two separate metrics when possible. One to measure the uniformity of spread of solutions and one to measure how close the approximation is to the actual PF in terms of average Euclidean distance. Measuring distance is trivial with the help of GD while measuring spread is not as straightforward and most commonly used metrics have serious downsides.

3.3.3 Averaged Hausdorff Distance (Δ_p)

Olver Schütze et al. [58] propose to use what they call Averaged Hausdorff Distance to address some perceived issues with GD and IGD quality indicators. First of all they propose to change GD and IGD to use power mean to average the distances instead of arithmetic mean used in GD and IGD. The new indicators are called GD_P and IGD_p where p is a parameter. The formulas for calculating those indicators become then as given in equalities (3.43) and (3.44). Finally the result of this quality indicator is the larger of the two quantities (GD_p and IGD_p). This metric tends to penalize outliers with increasing values of p.

$$GD_p(A, P^*) = \frac{\sum_{v \in A} d(v, P^*)^p}{|A|^p} \tag{3.43}$$

$$IGD_p(A, P^*) = \frac{\sum_{v \in P^*} d(v, A)^p}{|P^*|^p} \tag{3.44}$$

$$\Delta_p(A, P^*) = \max\left(GD_p(A, P^*), IGD_p(A, P^*)\right) \tag{3.45}$$

3.3.4 Hypervolume Indicator (I_H)

The description of this indicator can be found in, for example, work by Eckart Zitzler [72]. Let us first consider hypervolume indicator in two objective case.

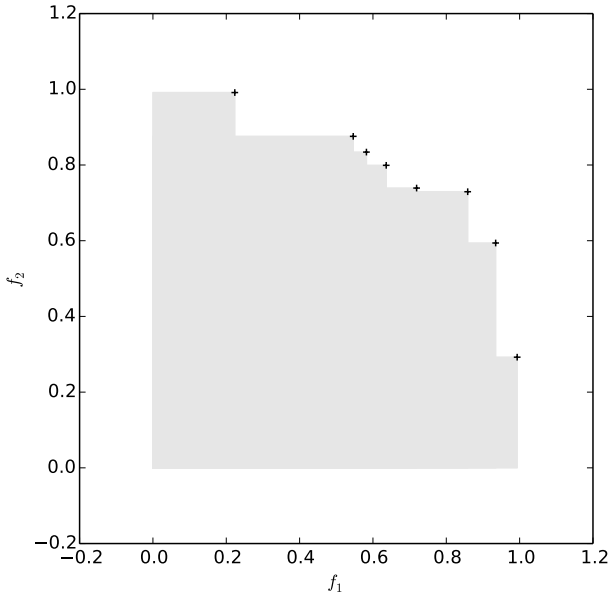


Figure 3.8: The result of hypervolume indicator I_H in this case would be the area shown in grey.

In this case this indicator is the volume of the union of all rectangles defined by points $(\min(f_1), \min(f_2))$ and $(f_1(\vec{a}), f_2(\vec{a}))$ where $\vec{a}$ is each of the vectors belonging to approximation A and $\min(f_i)$ is the minimum value for the objective f_i or at least a value that we know the objective cannot be lower than. An example of what this might look like is given in Figure 3.8, with $\min(f_1)$ and $\min(f_2)$ set to zero.

It should be readily apparent how to translate this concept to more than one objective. Indeed one just has to replace the rectangles with corresponding hyperrectangles defined as Cartesian product of intervals $(\min(f_1), f_1(\vec{a})) \times (\min(f_2), f_2(\vec{a})) \times \ldots \times (\min(f_k), f_k(\vec{a}))$ for a problem with k objectives and calculate hypervolume of the union of those products. This boils down to integration. This is probably simplest to do in practice with Monte Carlo integration algorithms. We have used an algorithm described below.

Here we have defined $\min(f_i)$ as before and $\max(f_i)$ as the maximum value of objective i in approximation archive A. Value $volume$ is defined by the volume of the bounding box defined by value $\min(f_i)$ and $\max(f_i)$ which can be calculate easily by multiplying $(\max(f_1) - \min(f_1)) \cdots (\max(f_k) - \min(f_k))$ that is the

Algorithm 7 Calculating hypervolume indicator I_H via Monte Carlo integration.

$inside \leftarrow 0$
for $iteration \leftarrow 1, N$ **do**
    $sample \leftarrow \left(\left(U_{\min(f_1)}, U_{\max(f_1)}\right), \ldots, \left(U_{\min(f_k)}, U_{\max(f_k)}\right)\right)$
    for $\vec{a} \in A$ **do**
        if $\forall j \in [1, 2, \ldots, k] : a_j < sample_j$ **then**
            $inside \leftarrow inside + 1$
            break
        end if
    end for
end for
return $volume \times \frac{inside}{N}$

viable ranges for each objective. What the algorithm does is sample the allowed range and counting the points that fall within the union of hyperrectangles defined by points in the archive. It then returns the ratio between number of samples that happen to fall within the hyperrectangles and the total number of samples N multiplied by the hypervolume taken up by the allowed range.

3.3.5 Pareto Spread Indicator (I_{OS})

This indicator consists of measuring the ratio between the hyperrectangles formed by the good and bad point of the real Pareto front and extreme points in the approximation. Good and bad points are defined here as estimates of the ideal and maximum point. That is we pick the point that has the lowest values of all objectives as the good point and the point that has the highest values of all objectives as the bad point. Extreme points are calculated from an approximation A and contain the best and the worst values for each objective. This performance indicator was proposed by Jin Wu et al. [68].

$$I_{OS}(A) = \frac{\prod_{i=1}^{k} |\max(f_i) - \min(f_i)|}{\prod_{i=1}^{k} |pb_i - pg_i|} \tag{3.46}$$

In the equation (3.46) we define $\max(f_i)$ to be the largest value for objective i in approximation A and corresponding $\max(f_i)$ to be the smallest value. Symbols pb_i and pg_i denote coordinate i of bad and good points as defined earlier. Plainly

speaking this indicator measures how big is the smallest box in to which all points in the approximation fit compared to the box defined by good and bad points. In essence this measures the extent of coverage of the real Pareto front by the approximation.

3.3.6 Number of Distinct Choices (I_{NDC_μ})

Proposed by J. Wu et al. [68]. This counts the number of points in the approximation with the condition that said points are distinct enough. This indicator uses a parameter called μ to divide the space in to hyperrectangles. The number of hyperrectangles will be $\frac{1}{\mu^d}$ assuming d-dimensional objective space. Two points are considered similar if they lie in the same hyperrectangle.

$$NT_\mu = \begin{cases} 1 & \exists \boldsymbol{y} \in A : \boldsymbol{y} \in T_\mu(\boldsymbol{q}) \\ 0 & \forall \boldsymbol{y} \in A : \boldsymbol{y} \notin T_\mu(\boldsymbol{q}) \end{cases} \tag{3.47}$$

Quantity NT_μ will be zero if there is no solution falling in to an area represented by point $\boldsymbol{q}$. It is equal to one otherwise. We then simply sum this value for each possible area taking in to the account the value of parameter μ. The value of this indicator is the numebr of hyperrectangles with at least one solution in them.

$$NDC_\mu(A) = \sum_{l_d=0}^{v-1} \dots \sum_{l_2=0}^{v-1} \sum_{l_1=0}^{v-1} NT_\mu(\boldsymbol{q}, A) \tag{3.48}$$

Here $\boldsymbol{q} = (q_1, q_2, \dots q_d)$ and $q_i = \frac{l_i}{v}$ where $v = \frac{1}{\mu}$. In the equations and discussion above the point $\boldsymbol{q}$ lies at an intersection of the grid lines formed when dividing the objective space in to hyperrectangles.

3.3.7 Cluster (I_{CL})

Proposed by J. Wu et al. [68]. It is defined as the ratio between the number of solutions in the approximation and the number of distinct solutions in the

approximation. The number of distinct solutions I_{NDC_μ} is defined in the way described in the previous subsection. If all solutions are distinct then the value of the indicator is equal to 1 and get's higher as fewer solutions are distinct. Generally the smaller the value of this indicator the better since it means there is no superfluous information in the approximation. This indicator is designed to measure clustering in the approximation.

$$I_{CL}(A) = \frac{I_N(A)}{I_{NDC_\mu}(A)} \tag{3.49}$$

3.3.8 Spacing Indicator (I_{SP})

Originally proposed by Jason R. Schott [57] and sometimes referred to in literature as Schott's metric. This metric measures the deviation in the distance between nearest neighbours in terms of Manhattan distance. The formula to calculate this metric is given in Equation (3.50).

$$SP(A) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(\bar{d} - d_i)^2} \tag{3.50}$$

Here $\bar{d}$ is the mean value of all $d_i, i \in \{1, \ldots, n\}$ and d_i is defined in Equation (3.51).

$$d_i = \min_{j \in \{1,\ldots,n\}} \left(\left| f_1^i(\vec{x}) - f_1^j(\vec{x}) \right| + \ldots + \left| f_m^i(\vec{x}) - f_m^j(\vec{x}) \right| \right) \tag{3.51}$$

This definition of d_i is for problems with two objectives but it can easily be extended for problems with more objectives. The problem with this metric is that it is not sensitive to large gaps in the approximated set. It only looks at the distance to the closest neighbour. This means that if an approximation consists of tightly packed clusters of points with large gaps between them the value of this metric will be low. However a solution like this can only be regarded as poor provided that the actual PF is continuous. For this reason this metric should be used carefully if at all.

3.4 Experimental Comparison and Analysis

Here we present the result of experiments for a large number of different MOPSO variants. All these variants can be found described in literature and they have been implemented and tested using the same procedure. The full results in the form of tables, as well as visually,are given in the appendix. Here we summarize those results with the help of two of the performance indicators I_{GD} and I_{IGD}. These have been chosen because they are commonly used and informative. After this we also present the results for every swarm analyzed when using our proposed performance indicators. The results in the first part are displayed in the two dimensional plane with the x axis representing the value of I_{GD} and y axis representing the value of I_{IGD}. One would expect the most successful methods to be represented in the bottom left of the plot. Low values of I_{GD} indicate that a method arrives at solutions that lie on or close to the real Pareto frontier. Low values of I_{IGD} indicate that values lie on or close to the real Pareto frontier plus that no large parts of the real Pareto frontier are missing in the approximation. It is probably useful to remind the reader that I_{GD} measures the average distance between an approximation point and the closest point to it in the reference set (a discretized representation of the real Pareto frontier). Indicator I_{IGD} measures what is in a sense the opposite of that — the average distance between a point in the reference set and its closest point in the approximation. Successful methods will minimize both of these indicators. If a method gives low value of I_{GD} and a (comparatively) large value of I_{IGD} in the plots below ,it means that while the method successfully finds points on or close to the real Pareto frontier, the approximation omits some parts of the Pareto frontier. Therefore, plotting them against each other gives additional information that would not be readily apparent just by examining those values in the table.

The first thing that stands out in the tables is that the K. E. Parsopoulos [47] approach stands out as being the worst in all the test cases it has been tried on. This seems to suggest that such a simple approach does not work well on complicated, high-dimensional problems we have tested on.

The approach proposed by C. A. C. Coello et al. [6] does not seem to give good results for the chosen metrics either. Its performance is consistently poor for

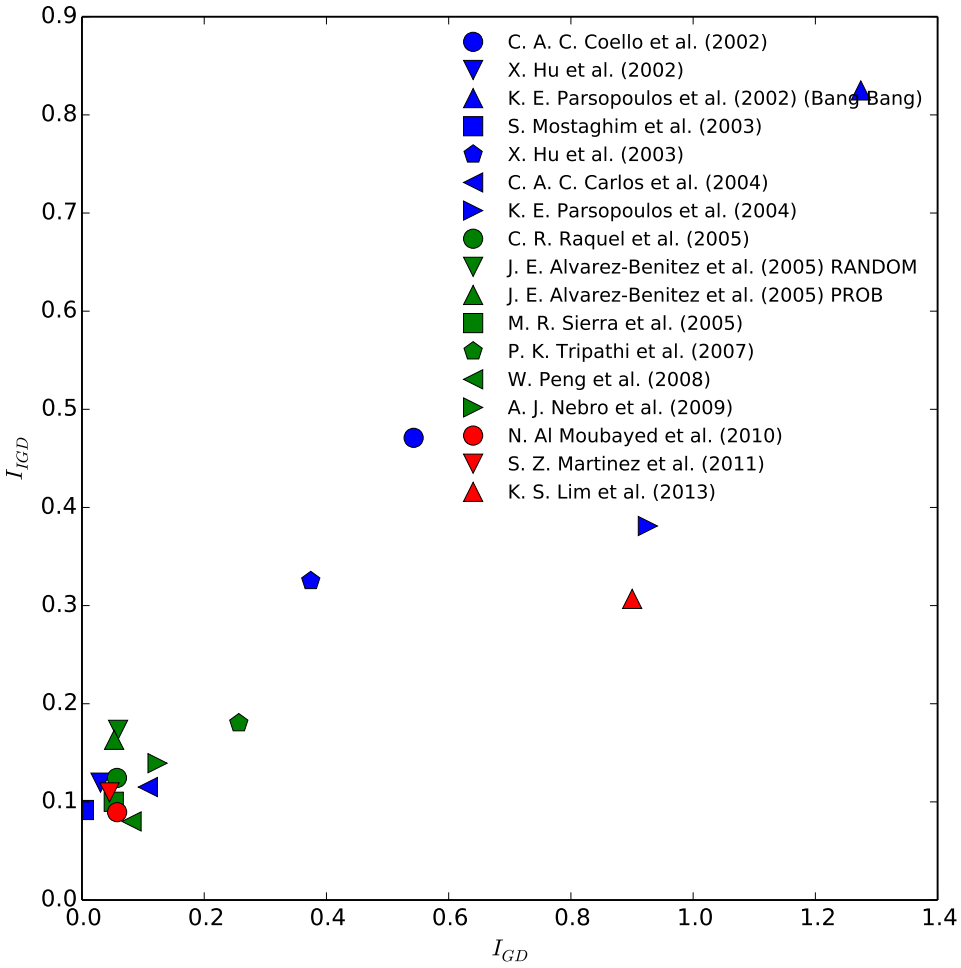


Figure 3.9: Overview of the results for problem UF01. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

problems with 2, 3 and 5 objectives. It is a simple approach that always takes the particle that is deemed to be in the least crowded area of the approximation as the global best.

The dynamic neighbourhood approach by X. Hu et al. [22] seems to do well on all two objective problems. The main problem with this approach is that it is not immediately clear how to generalize it to the problems with more than two objectives.

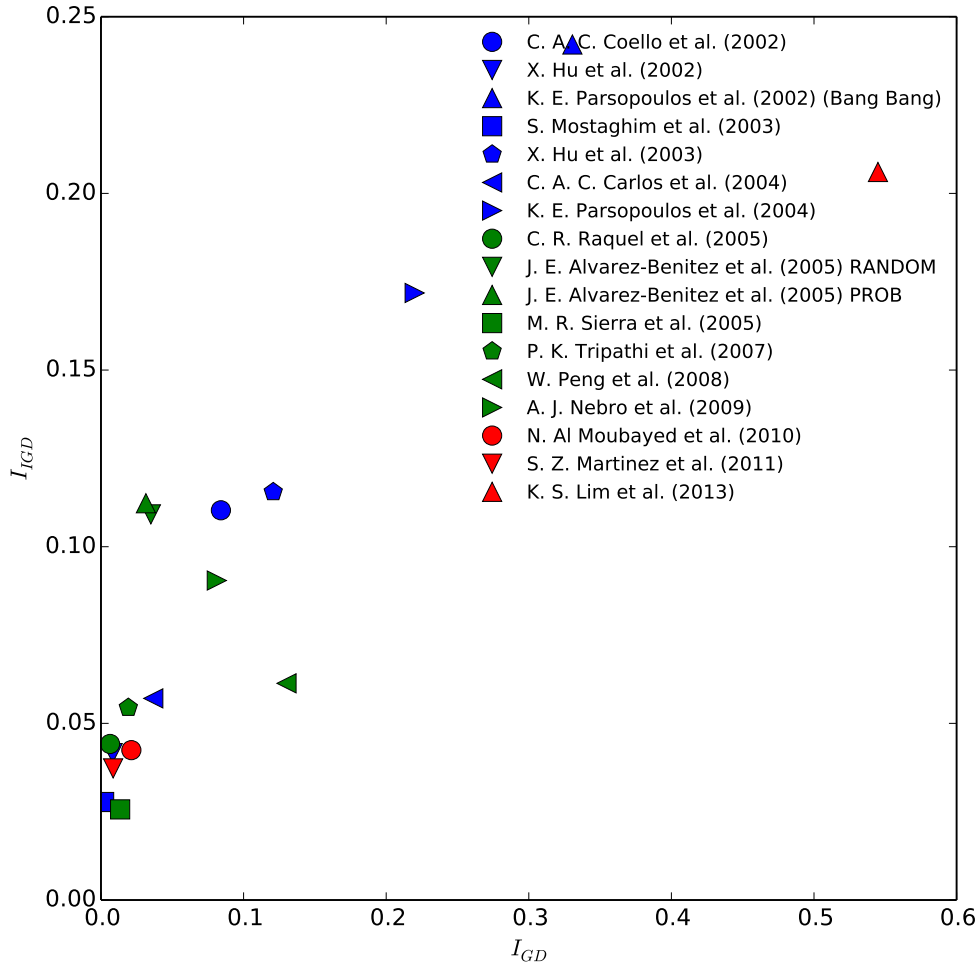


Figure 3.10: Overview of the results for problem UF02. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

The sigma particle approach proposed by S. Mostaghim et al. [40] seems to work well on all problems. The only exception is UF13, in which it does poorly in terms of the I_{GD} indicator. This approach was used also in the our proposed heterogeneous swarms. The fact that it usually outperforms more modern approaches is quite remarkable.

It is interesting to see that in a lot of cases the X. Hu et al. [23] method seems to do worse than the X. Hu et al. [22], which it was supposed to improve upon. With every test problem it has been evaluated with, it seems to be worse in

terms of both I_{GD} and I_{IGD} performance indicators. The proposed changes to the original dynamic neighbourhood method do not seem to provide any kind of marked improvement and, in fact, they seem to be detrimental to the performance of the method.

The method proposed by C. A. C. Coello et al. [5] differs from the C. A. C. Coello et al. [6] only by the inclusion of a mutation operator everything else staying the same. Therefore, it is interesting to see that this simple change to the method provides a marked improvement in performance. With all the test problems the results in terms of both I_{GD} and I_{IGD} are markedly better when using the mutation operator. This seems to indicate that adding a mutation operator is a good way of improving the performance of MOPSO methods.

The vector evaluated method of K. E. Parsopoulos et al. [48] does not seem to be performing well on any of the tested problems, except for UF13. It is not clear why this is the case.

C. R. Raquel et a. [52] method works well on all problems except for UF05, which is a problem with a discrete Pareto frontier. It makes sense since this method makes the assumption that the Pareto frontier is continuous.

The performance of the J. E. Alvarez-Benitez et al. [2] family of methods is comparable to other algorithms in most cases. It always performs on average worse than most methods tried on the indicators I_{GD} and I_{IGD}. On problems UF08 and UF09 the value of the I_{GD} is on far with the best methods tried. This indicates that these methods may prematurely converge without exploring the search space well. One notable exception is the problem UF12 where these methods do exceptionally well. Orders of magnitude better on the I_{IGD} indicator compared to others.It should be investigated, why this is so.

The M. R. Sierra et al. [60] method performs very well on problems UF01 and UF02 yet seems to perform rather poorly with problem UF03, which as identical Pareto frontier, but a more complicated Pareto set. It generally performs very well compared to other methods. This may be why this is a very popular method often used to benchmark other MOPSO methods against.

The P. K. Tripathi et al. [63] does very poorly on problem UF04. It also does poorly on problems UF05, UF08, UF12, UF13. It does comparatively well on

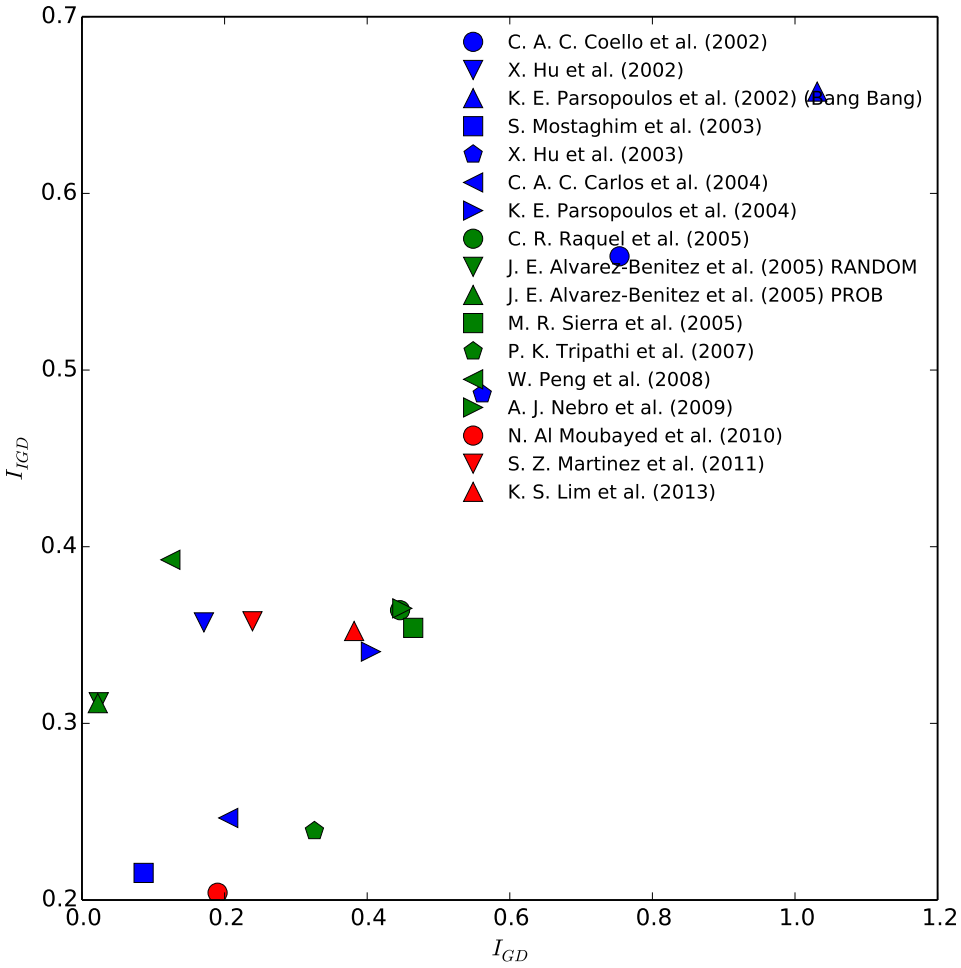


Figure 3.11: Overview of the results for problem UF03. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

problem UF11. Problem UF05 can be explained by it being a problem with a discrete Pareto frontier while this method, like most others, make an assumption that the Pareto frontier is mostly continuous.

The decomposition based MOPSO method by W. Peng et al. [49] performs very well on problem UF05. It works comparatively well on all problems with none being exceptionally worse than other methods.

A. J. Nebro et al. [42] proposed a method that is supposed to be an improvement

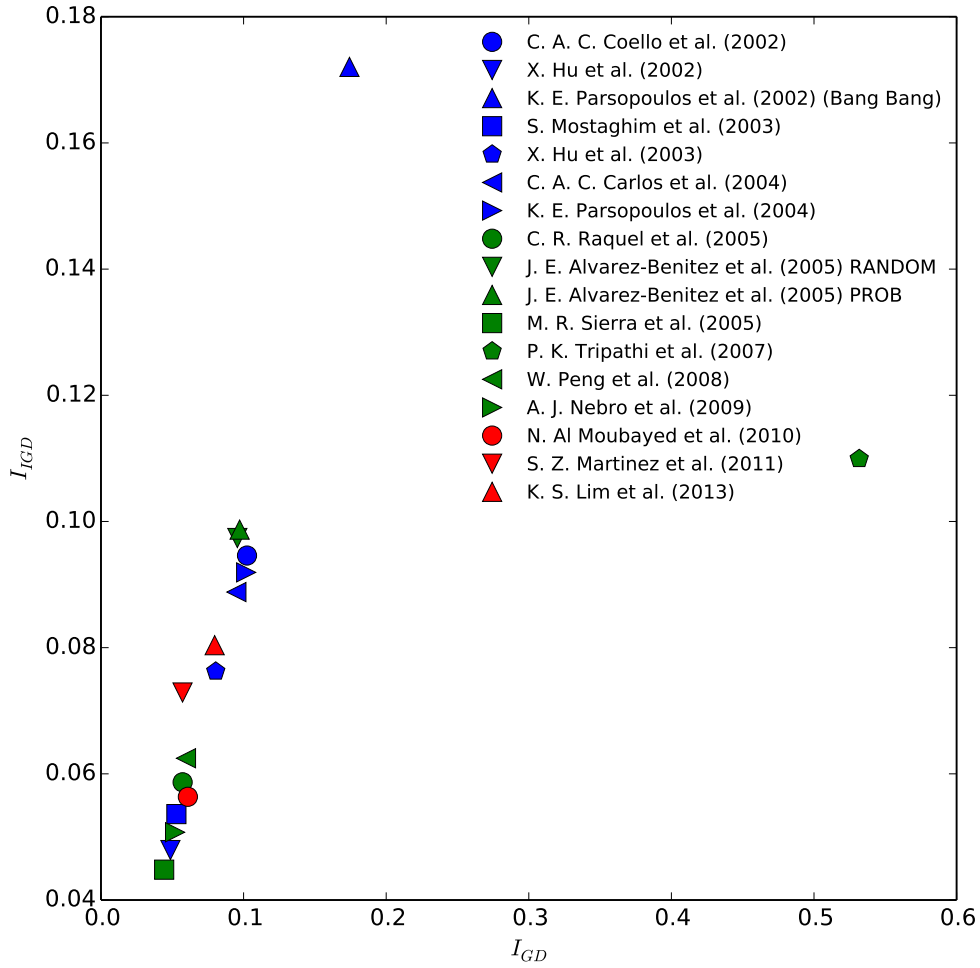


Figure 3.12: Overview of the results for problem UF04. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

upon the one proposed by M. R. Sierra et al. [60]. In all cases we have tried, it performs on average worse than that method though. Therefore,the assumptions made by the authors about how to improve the performance seem to be wrong.

N. Al Moubayed et al. [1] proposed a decomposition based method. Like all the other decomposition based methods it performs well on the UF05 test problem. It also performs well on the UF07 problem which is also a problem with a discontinuous Pareto frontier. It also performs exceptionally well on problems with more than two objectives. With the only exception being the UF12 problem.

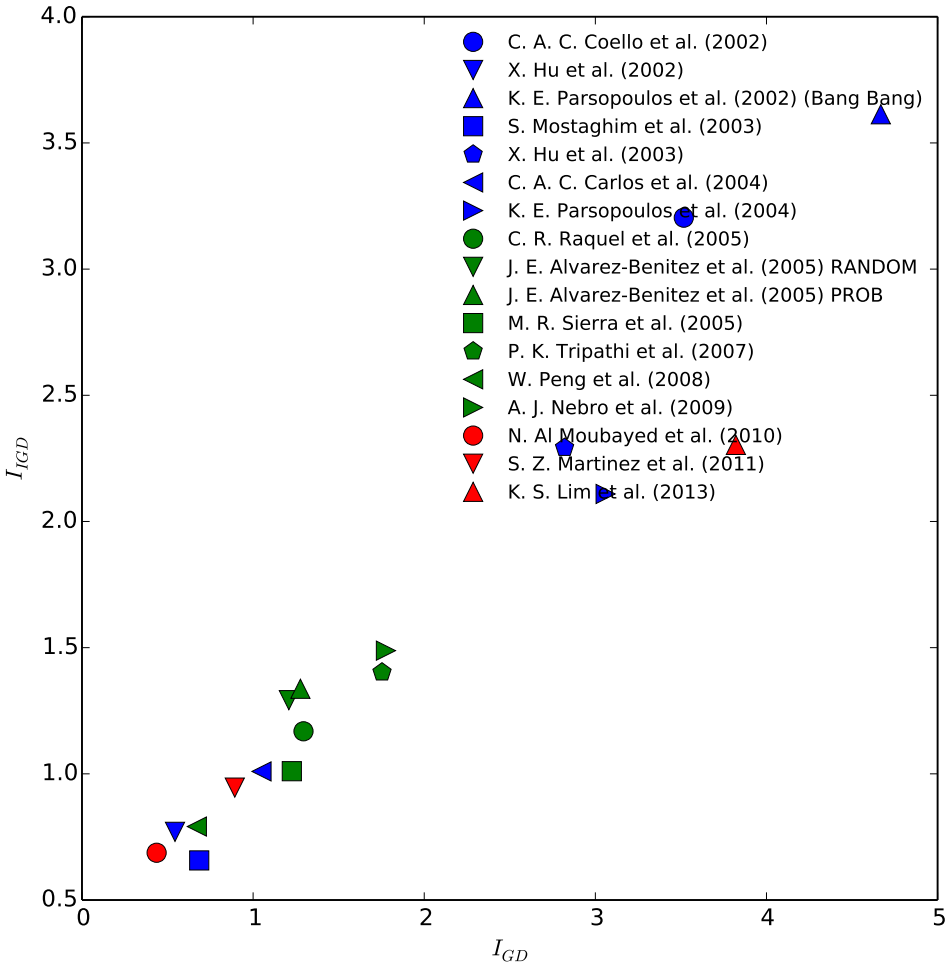


Figure 3.13: Overview of the results for problem UF05. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

S. Z. Martínez et al. [69] proposed a decomposition based method. Like the other methods based on decomposition it performs well on the UF05 test problem. Other than that it performs slightly worse than the other decomposition methods.

K. S. Lim et al. [37] method performs very poorly on all problems we have tested it on. This seems to indicate, along with the data collected from the K. E. Parsopoulos et al. [48] method when applied to these problems, that the vector evaluated methods do not fare well in comparison to other methods

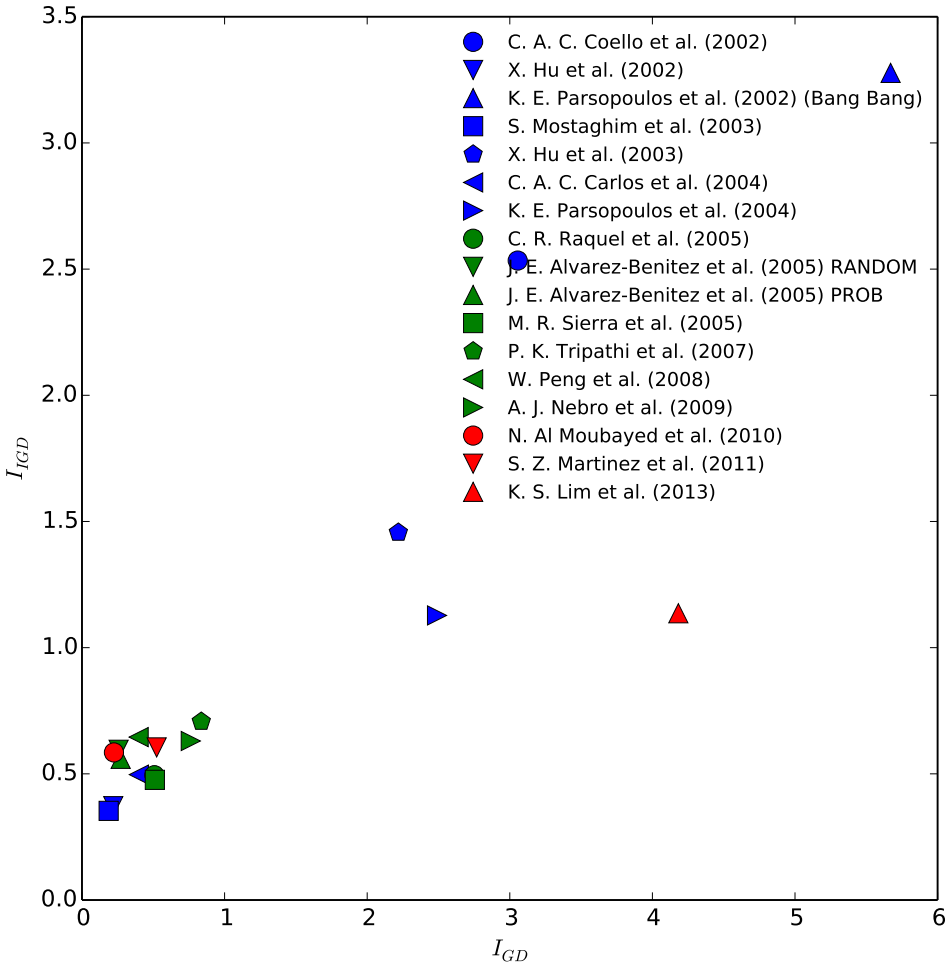


Figure 3.14: Overview of the results for problem UF06. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

when applied to high-dimensional multi-objective problems of the kind used here.

3.5 Conclusions

In this chapter we have experimentally evaluated known MOPSO methods. The following conclusions can be ascertained from this analysis:

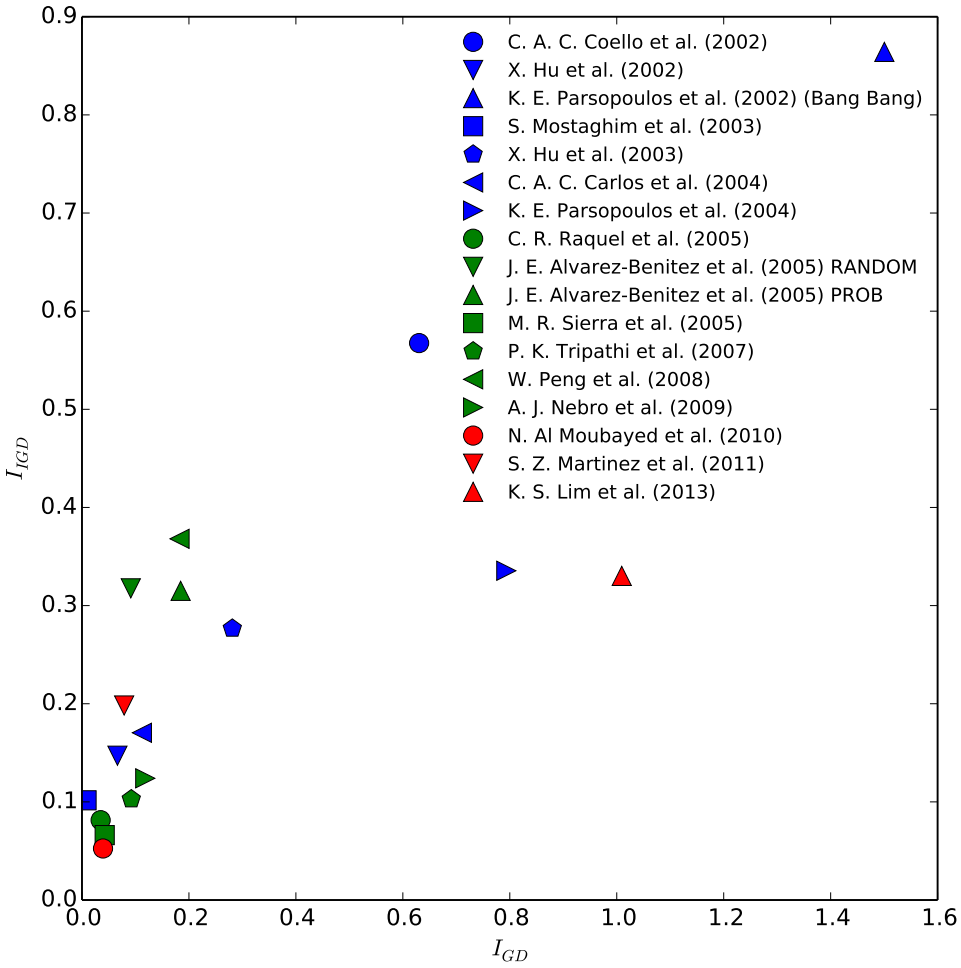


Figure 3.15: Overview of the results for problem UF07. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

- It has long been known that the use of mutation operators can improve the performance of single objective PSOs. However, comparative studies that contrast the methods that use mutation operators and those that do not, have not been performed in the field of multi-objective PSO. Because of the large number of MOPSO methods that have been surveyed in this research both with and without mutation, conclusions about its use can be made. It can be seen from the data that mutation can significantly improve the performance of MOPSO methods.

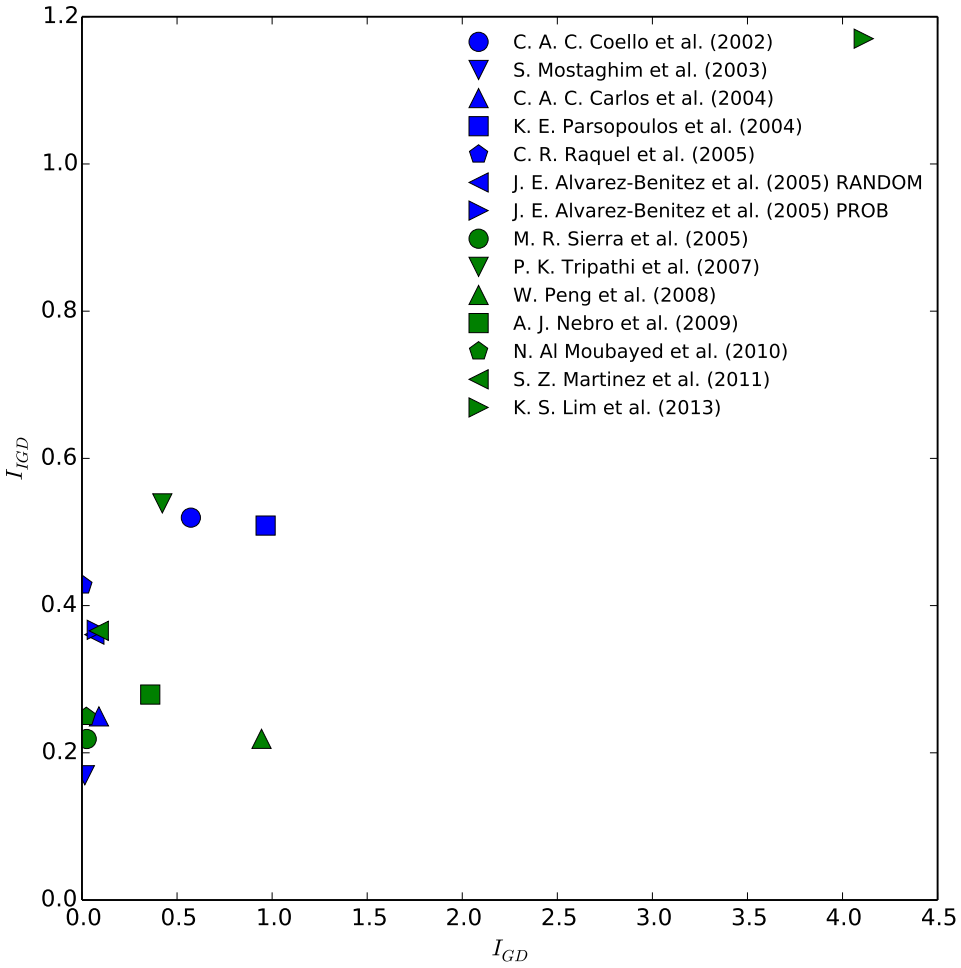


Figure 3.16: Overview of the results for problem UF08. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

- Decomposition based approaches work well on problems with discontinuous Pareto frontiers. This may be due to the fact that they do not make assumptions about the Pareto frontier being continuous the way most MOPSO approaches do.

- Vector evaluated MOPSO methods have been proposed early on for the use with multi-objective problems. In vector evaluated approaches there are several populations of individuals each optimizing a single objective. Information is exchanged between populations. In the simplest case one

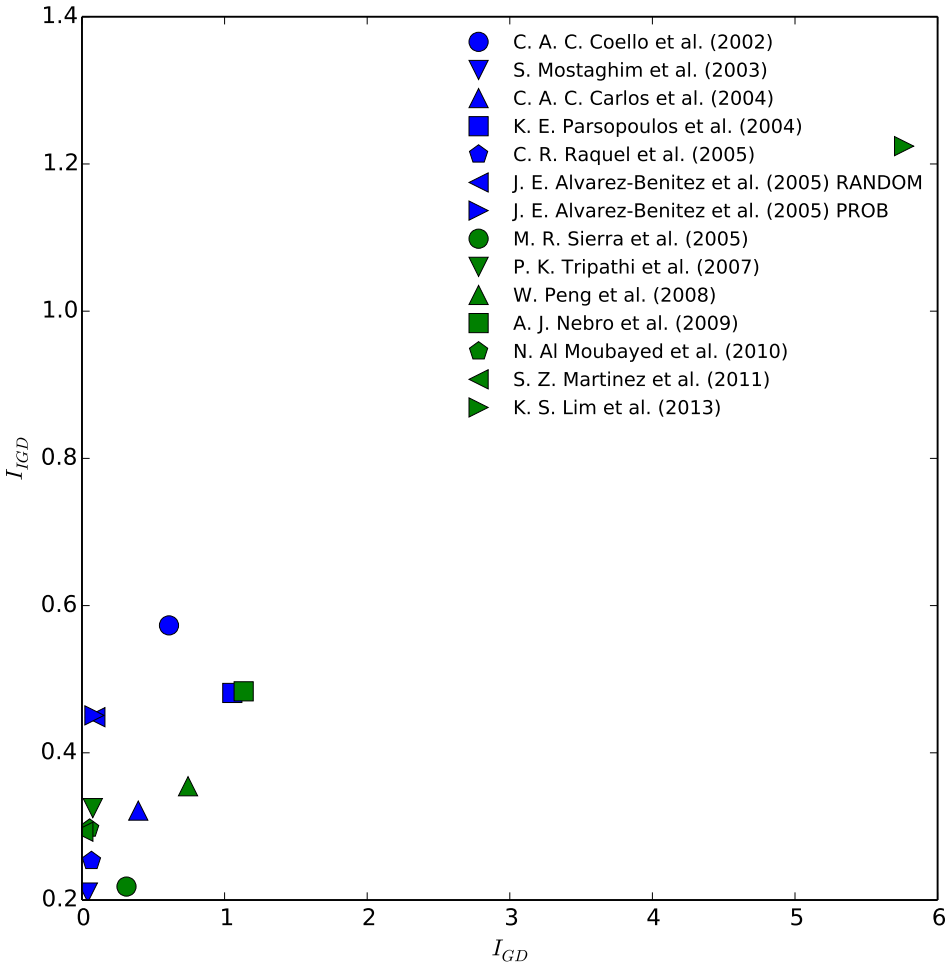


Figure 3.17: Overview of the results for problem UF09. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

population will take the best solution found by another population as its global best solution. Since they optimize different objectives ,this will allow the swarms to explore solutions that optimize both objectives. However, from the experimental results that have been gathered in chapter 3, it can be seen that vector evaluated MOPSO methods do not work well compared to other MOPSO approaches. Several vector evaluated approaches have been tested and none of them come close in terms of results to other approaches.

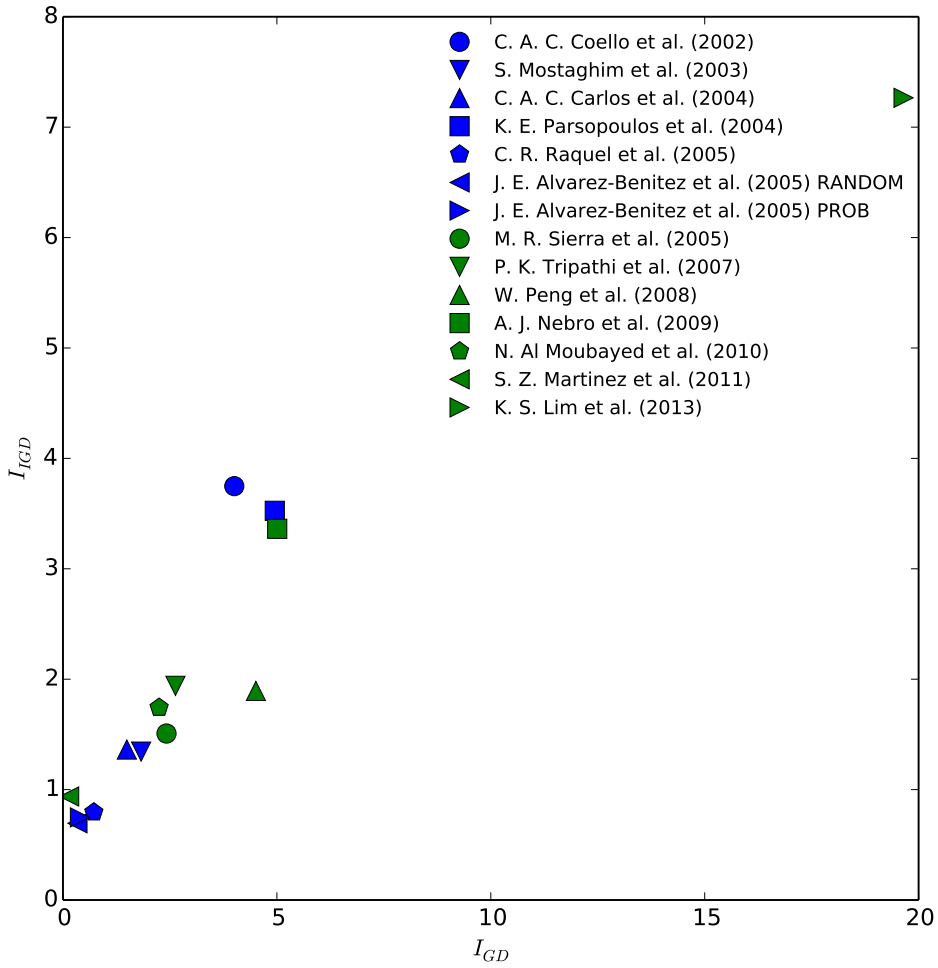


Figure 3.18: Overview of the results for problem UF10. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

There seems to be a distinct lack of heterogenous approaches in the methods discussed, so far. Heterogeneous methods use different types of particles sharing information. Particles usually differ in their velocity and position update rules. They may also differ in how they select the best neighbour and how they update their personal best solution. The latter two differences are particularly important in MOPSO methods,where there are a lot of different ways how the neighbourhood best is selected and how personal best is updated. Using heterogeneous approaches may solve some problems with existing methods.

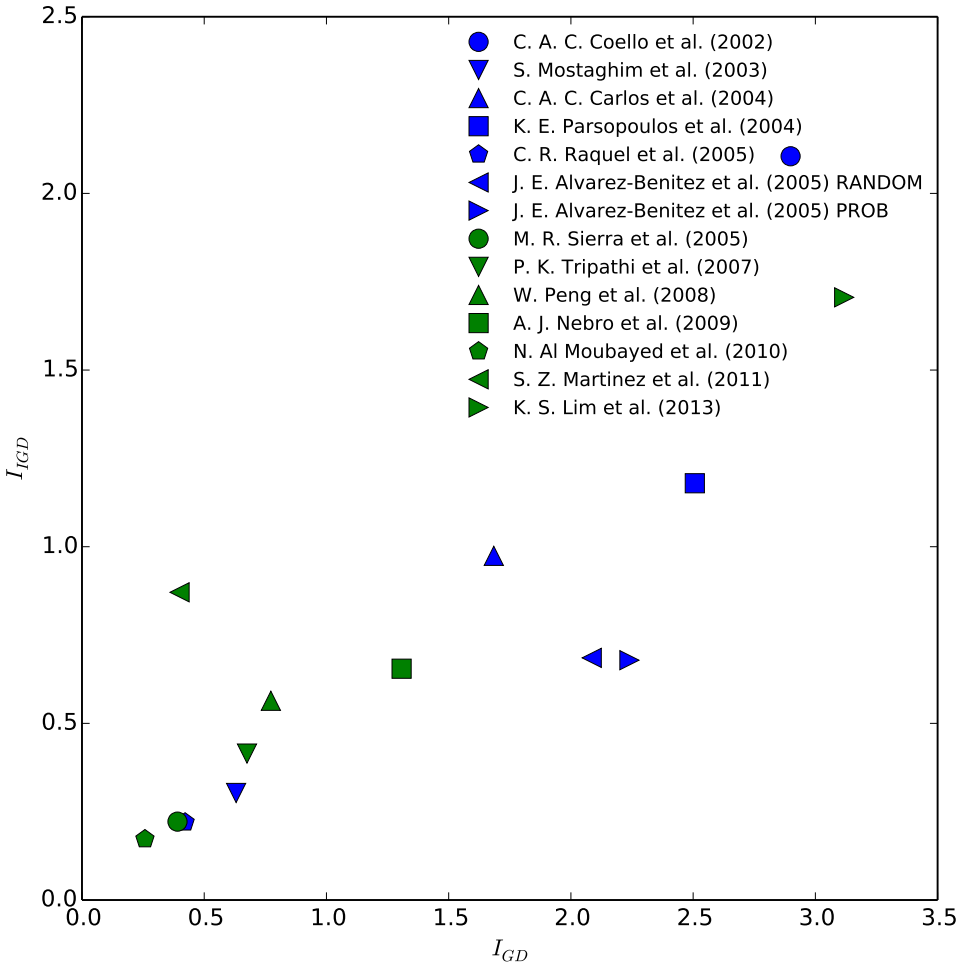


Figure 3.19: Overview of the results for problem UF11. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

For example, it is possible that heterogeneous methods will be able to solve problems in distinctly different problem classes successfully. That is because using different particles means that particles that are more successful with one class of problems will offset the influence of particles that are less successful with that class. And vice versa.

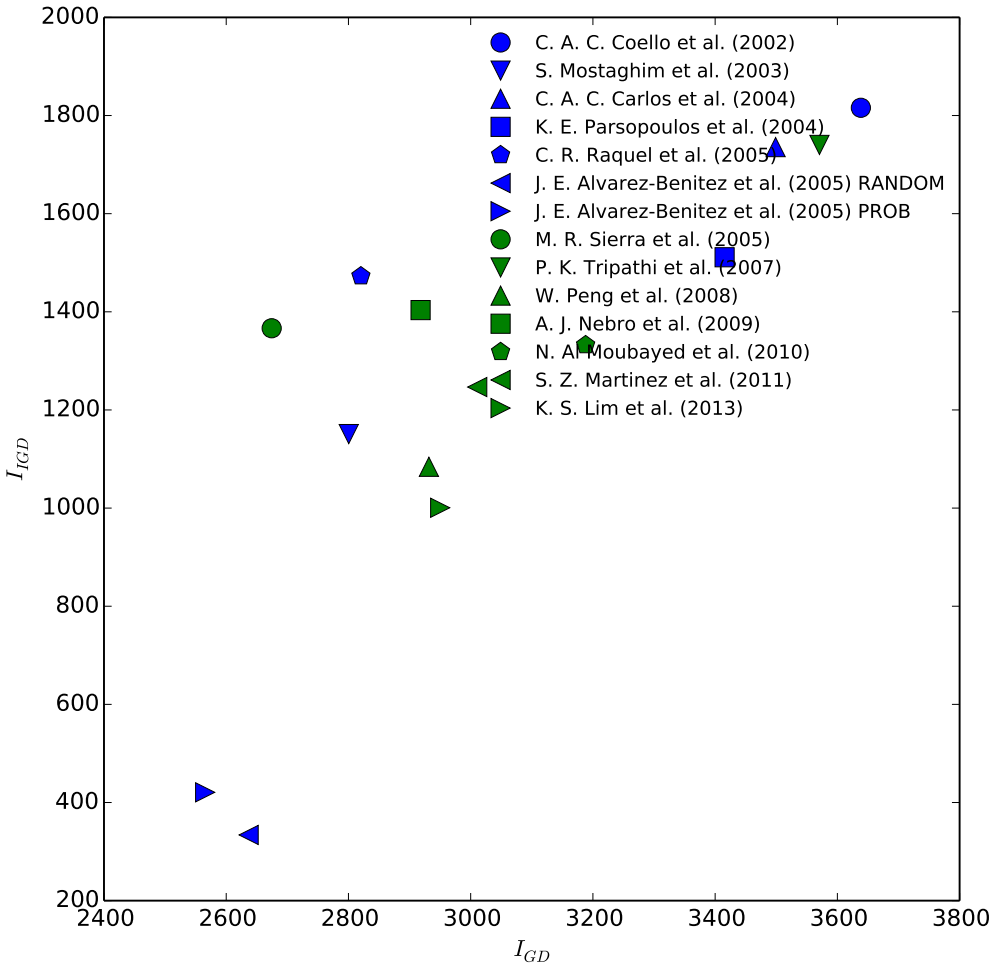


Figure 3.20: Overview of the results for problem UF12. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

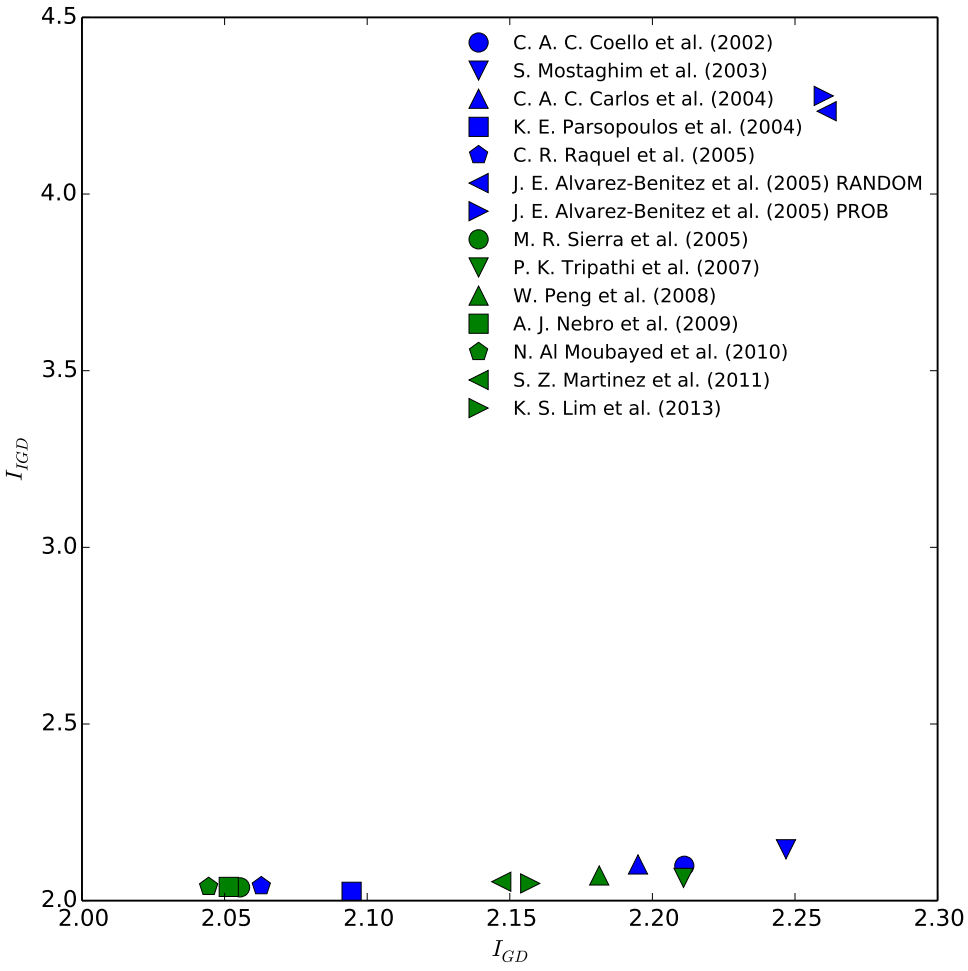


Figure 3.21: Overview of the results for problem UF13. Average values of indicators I_{GD} and I_{IGD} are plotted against each other for every method.

Chapter 4

Proposed Methods

In this chapter, solutions to several perceived problems in the field of multi-objective particle swarm optimization are proposed. Firstly, a classification of existing MOPSO methods is provided. While there exist several different classification schemes, they are usually very broad and do not reflect the implementation details of the underlying methods accurately. In an overview of the existing MOPSO methods it has become apparent that most MOPSO methods differ from the generic MOPSO method that is given in Chapter 2 (Algorithm 4), in three major areas. The first one is the underlying concept (e.g. Vector Evaluated Methods, Decomposition-based methods, etc.), the second one is how the method makes sure that the Pareto points in the approximation uniformly cover the Pareto frontier and the third one is the mutation operator (if any) used. We propose to classify the methods with regards to these three aspects. An example of such classification is given in the following section.

In the second section it is argued that existing performance indicators for the measurement of the solution spread are not adequate for the purpose. Counter examples that demonstrate the cases where these indicators do not accurately measure the intended qualities are given. Two new performance indicators based on this information are proposed. The first one is simple to compute, but it does not fully capture the intuitive notion of the uniform coverage. The second one is more complicated,but also more accurate.

Two new MOPSO methods are introduced further. Both are based on the notion

of heterogeneous PSO – having several different particle types in the same swarm. Particles differ in their velocity and position update rules. Particles share the information via the same non-dominated point archive. The idea behind these methods is that having different particle types, each emphasizing different aspects of MOPSO performance will result in them counter-acting each others drawbacks. So,if we have one particle that emphasizes getting close to the Pareto frontier and another emphasizing covering the Pareto frontier as uniformly as possible,we will end up with a swarm that tries to achieve both of these goals. An experimental comparison between these proposed methods and popular MOPSO methods found in literature is given.

Finally, the software framework that has been developed during the preparation of the thesis is presented from the framework user's point of view. The framework is open-source and publicly available via a Mercurial repository. The possibilities offered by the framework with regards to prototyping, testing and using multi-objective PSO methods are briefly described. Also the technical details of the experimental set-up that has been used for this thesis are given. Because the experiments presented in the thesis are computationally expensive and numerous, they have to be performed on super-computing clusters. The framework is designed with running in distributed computing environments in mind.

4.1 Proposed Classification of Existing MOPSOs

An attempt at classifying existing MOPSO methods is given in the Table 4.1. Each method is described by the primary idea it uses, how it handles population diversity and whether it uses mutation operators.

4.1.0.1 Common MOPSO Types

We classify MOPSO methods by their "type" which means the primary idea behind that method. We use the generic "Pareto dominance" if the method simply updates personal best solution if it dominates the previous one and there

Method	Type	Diversity control	Mut.
C. A. C. Coello et al. (2002)	Pareto dominance	Niching	No
X. Hu et al. (2002)	Dynamic neighbourhood	None	No
K. E. Parsopoulos et al. (2002)	Weighted aggregation	None	No
S. Mostaghim et al. (2003)	Pareto dominance	None	Yes
J. E. Fieldsend et al. (2002)	-	-	-
X. Li et al. (2003)	Non-dominated sorting	Crowding distance	No
X. Hu et al. (2003)	Dynamic neighbourhood	None	No
C. A. C. Coello et al. (2004)	Pareto dominance	Niching	Yes
K. E. Parsopoulos et al. (2004)	Subswarm	None	No
S. Mostaghim et al. (2004)	Subswarm	None	Yes
C. R. Raquel et al. (2005)	Pareto dominance	Crowding distance	Yes
J. E. Alvarez-Benitez et al. (2005)	Pareto dominance	No. of d. solutions	No
M. R. Sierra et al. (2005)	Pareto dominance	Crowding distance	Yes
M. Salazar-Lechuga et al. (2005)	Pareto dominance	Fitness sharing	Yes
P. K. Tripathi et al. (2007)	Pareto dominance	Crowding distance	Yes
W. Peng et al. (2008)	Decomposition	None	Yes
U. Wickramasinghe et al. (2009)	Non-dominated sorting	None	No
A. J. Nebro et al. (2009)	Pareto dominance	Crowding distance	Yes
Y. Wang et al. (2009)	P. O. ranking	None	No
N. Al Moubayed et al. (2010)	Decomposition	None	No
A. Elhossini et al. (2010)	Pareto strength	None	Yes
S. Z. Martinez et al. (2011)	Decomposition	None	No
A. J. Nebro et al. (2013)	Pareto dominance	Hypervolume c.	Yes
K. S. Lim et al. (2013)	Subswarm	None	Yes
I. C. Garcia et al. (2014)	Pareto dominance	Hypervolume c.	Yes

Table 4.1: Classification of MOPSO methods examined here.

is nothing distinct about that method to justify creating a category for it. All approaches that more than one method uses are given in the list below.

Pareto dominance This is a placeholder for the methods that do not do anything special. They usually use Pareto dominance relationships to determine if the personal best solution needs to be updated and collect solutions in a non-dominated point archive. They usually differ by how they handle population diversity. Diversity control methods are discussed below. There are 11 methods in this category.

Dynamic neighbourhood Consists of only two methods and is suitable only for the problems with two objectives. Works by using one of the objectives to find particles neighbours and another to pick the best neighbour.

Non-dominated sorting These methods use the concepts of NSGA-II by K. Deb et al. [12]. Solutions are arranged into ranks, where a rank consists of

non-dominated sets of particles. These ranked particles are then used when updating personal best solutions. There are two methods of this type in our study.

Subswarm In this approach swarm is divided into subswarms. Each subswarm may optimize a single objective of the multi-objective problem. Usually subswarms share information in some way. We analyzed three methods that fall within this category. Most of these methods are inspired by Vector Evaluated Genetic Algorithms (VEGA) by J. D. Schaffer [56].

Decomposition In this approach each particle gets a separate aggregate function with different weights. This is similar to weighted aggregation approaches ,but instead of optimizing a single function, the swarm now optimizes as many functions as there are particles. Weights are usually chosen so, as to uniformly cover the weight space. Neighbourhood relationships between two particles are often determined by their distance from each other in the weight space. These approaches are inspired by the MOEA/D family of algorithms by Q. Zhi et al. [70]. There have recently been a relative abundance of publications in using these techniques in MOPSO methods.

Other methods include "Weighted aggregation" where objectives are aggregated using weights that change during the course of PSO evolution, "P. O. ranking" or Preference Order Ranking, which is described in the work by F. di Pierro et al. [13] when used in genetic algorithms and "Pareto strength" which is identical to the concept used by E. Zitzler et al. [73] in the SPEA2 algorithm.

4.1.0.2 Diversity Control

Multi-objective optimization methods employ diversity control in order that no areas of the objective space remain uncovered and those that are covered are covered uniformly. This is often achieved by directing particles towards those solutions that lie in the areas of objective space sparsely populated by other solutions. This, it is hoped, will help to fill in those parts of the objective space with non-dominated points in the approximation. It is important to note that this

presupposes that Pareto frontier is continuous since with discrete frontiers can often be meaningless. Although, even with discrete frontier, it is important to make sure that no areas of Pareto frontier remain uncovered. However, methods employed here will often not work in the case of discrete frontiers.

Niching When using niching, objective space is divided into a specified number of hyper-cubes and each hyper-cube that contains solutions is assigned a niching count that corresponds to the number of solutions in that hyper-cube. If a solution is in a hyper-cube that contains few solutions, it will be preferred as a leader as opposed to particles that lie in hyper-cubes that already contain many solutions. Two methods we analyzed used niching.

Crowding distance Crowding distance is a method proposed by K. Deb et al. [12] and is very commonly used. It works by sorting solutions by each objective in turn and measuring the distance between the two neighbours - one to the left and one to the right. The sum of these values is then taken as a crowding distance of the solution. Solutions at the start and end of the sorted solution list get infinite crowding distance. This makes sure that the edges and corners of the Pareto frontier are explored.

Hypervolume contribution Hypervolume measures how much of hypervolume a solution alone contributes. That is what is the hypervolume that this solution covers that does not intersect the hypervolume of some other solution. Two methods use this approach. It is described in more detail in the sections on those methods.

Fitness sharing Fitness sharing is similar to niching but a hypersphere is used instead and we count how many solutions fall within a given radius of a solution. Plus fitness sharing penalty is applied, the closer other solutions are to a given solution. One method uses this approach.

Number of dominated solutions Another method of diversity control is to count how many current particles (after position update) each solution in the non-dominated point archive dominates. Particles are preferred if they dominate fewer particles since this indicates that the area around that solution is unexplored. This approach is used by one method.

4.1.0.3 Mutation

In the table we only indicate if mutation is used or not. Common types of mutation are the same as those used in the multi-objective genetic algorithms. Overview of mutation operators is given in the section on the single objective PSO. Polynomial mutation seems to be a popular operator among different MOPSO variants.

4.2 Problems with Existing Performance Indicators for Solution Spread

The goal of measuring solution spread in an approximation is to see how well the approximation covers the real Pareto frontier. The idea and main concern here is that the approximation has to give an accurate picture of the Pareto frontier to the decision maker. First of all, we do not want any large portions of the frontier missing in the approximation. Second of all the spacing of points in the approximation has to be "uniform" in the sense that the distances between the closest points should be similar. All of this, of course, only applies when the frontier is mostly continuous. It can be argued that the existing performance indicators do not capture these intuitive notions very well. Let us look at the indicators that in one way or another can measure solution spread.

First of all, there is I_{IGD} which will take on larger values if large portions of the Pareto frontier are missing. This is because it calculates the mean of the closest distance between each point in the real Pareto frontier (a discretized version of it that is) and a point in the approximation. If the approximation has missing pieces in it, these distances will be increased. The problem with this is that these distances are also influenced by how close to the real Pareto frontier the points are.Therefore,it is not clear how much of the resulting value is contributed by closeness to the real PF and how much by uniformity of coverage. This makes this indicator not suitable for the purpose of measuring the solution spread.

The only other widely used indicator that could be considered for the task is the Schott indicator I_{SP}. It measures the standard deviation of the distance between

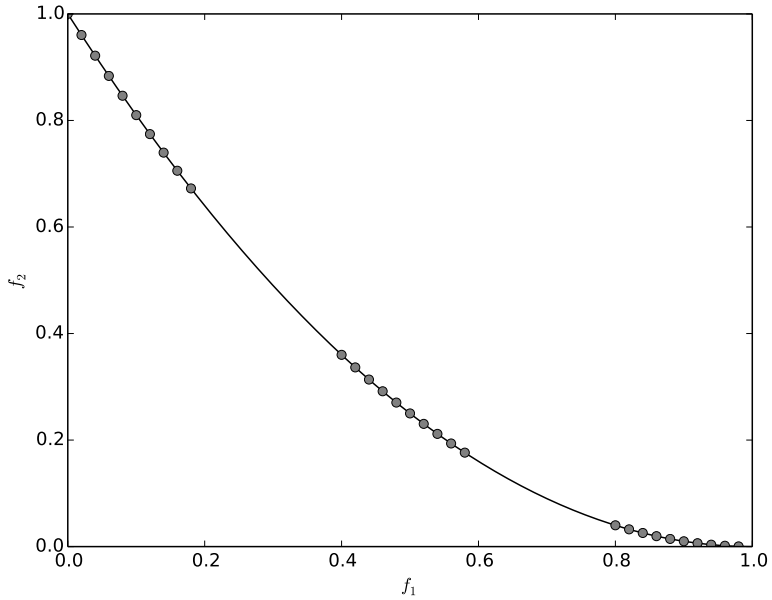


Figure 4.1: An example of a situation where I_{SP} performance indicator will not measure uniformity of coverage accurately.

the point in the approximation and the point closest to it in terms of Euclidean distance. On the surface this seems like a reasonable approach - if the points are at equal distances from one another then the standard deviation will be small. If the points are spaced unevenly,the standard deviation of the distances to the closest neighbours will be larger. However, this only captures a part of what we want from a performance indicator designed to measure the solution spread. It does not work when portions of the Pareto frontier are missing in the approximation. In fact this indicator can give a perfect score even if most of the frontier is missing. It is easy to see why this happens by considering an example in Figure 4.1. If the points are spaced uniformly relative to each other, but large portions of the frontier are missing, the indicator will still give low values even though it does not make sense to say that the frontier is covered uniformly by the approximation.

In our opinion it is apparent that there is a need to develop a new performance indicator that would correspond to the intuitive notion of uniform coverage. We propose two such indicators in the following section.

4.3 Proposed Performance Indicators For Measuring Solution Spread

With the deficiencies of existing metrics in mind, as illustrated by the counter examples given in previous section,there is a need to design new ones. We propose two new metrics. One is designed for simplicity not perfectly capturing the intuitive notions described here. The other one is designed to correspond with those notions more accurately, but it is more difficult to compute. The metrics have been designed with these goals in mind:

1. Quality indicator should accurately capture the intuitive notion of uniform coverage.
2. They should measure one aspect of optimizer performance only – namely the approximation uniformity.
3. They should generalize to any number of objectives.
4. They should be as simple and efficient to compute as possible not violating any of the above goals.

4.3.1 Performance Indicator I_{UNI^1}

When the Pareto set of a problem is known, we propose the following metric which we believe captures uniformity of the aspect of optimizer performance well. Please, note that the Pareto frontier is known in its discretized form. That is the real Pareto frontier is sampled at some fixed number of points. We will suppose that the set that represents the Pareto frontier by definition covers it uniformly and contains enough points. The calculation involves two different stages, which we will discuss separately. During the second stage of the calculation, for each point in approximation, we measure for how many points in the reference set this point is the closest one in terms of Euclidean distance. This stage is illustrated graphically in Figure 4.2. Here the small black dots represent the reference set and the larger gray circles represent the

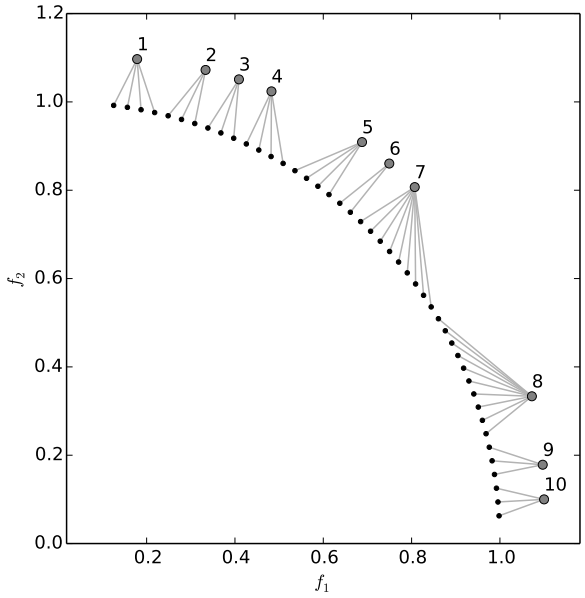


Figure 4.2: Closest points in reference set when approximation is not uniform.

approximation. We connect each black dot to the gray circle that is the closest to it in terms of Euclidean distance.

Here we can see that several large areas of the reference set are missing from the approximation. Therefore, the approximation fails to find certain parts of the Pareto frontier and this must reflect in the value of the metric for that approximation. As any other non-uniform patterns should. In this case, starting with the point labeled 1 the number of points in the reference set for which that point is the closest, one in the approximation are 4, 3, 3, 4, 4, 2, 9, 10, 3 and 3 respectively, the arithmetic mean of those values is 4.5 and the standard deviation is approximately 2.58,which is fairly large. Now, let's look at Figure 4.3, which shows a similar approximation, containing the same number of points, but spaced equally from each other as we may wish in a good approximation.

We can see similar number of lines connecting each point in the reference set to their closest point in the approximation. And if we count them,we see that this time the numbers are 4, 5, 4, 4, 4, 5, 4, 4, 5 and 5 respectively. Mean is 4.4 (similar to what it was before), but standard deviation is only 0.49 approximately. It seems, so far,that our metric measures what we want it to - namely the

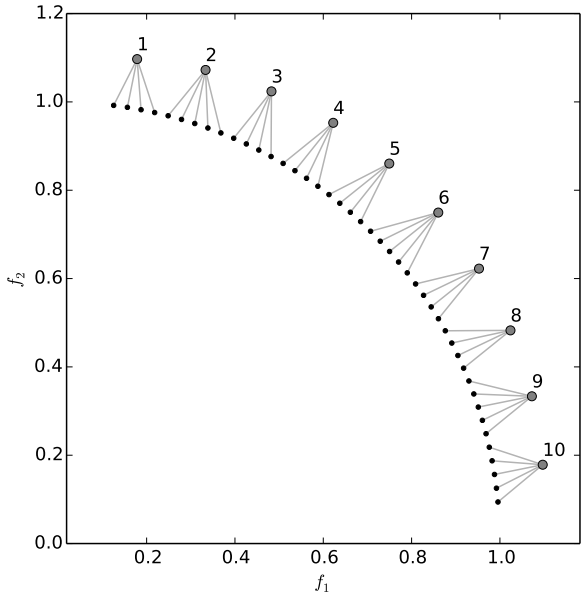


Figure 4.3: Closest points in reference set when approximation is uniform.

uniformity in spacing between the points of the approximation.

It does seem that for the two dimensional case this works fairly well. If parts of the Pareto frontier are missing, this will cause more points in the reference set to associate with the same points in the approximation. Whenever the points are spaced evenly along the Pareto frontier, they will be the closest to a similar number of reference points. This will mean that the standard deviation over the number of the closest reference points will be small. All these concepts also translate trivially to more objectives than two.

There is, however, a problem with using just this simple scheme alone. One of our goals when designing a metric was to make sure it measures uniformity in covering the real Pareto frontier and that alone. This means our metric must be in a sense orthogonal to calculating just the distance to the real Pareto frontier, which can easily be done by calculating generational distance. It can quite simply be shown that it is not the case here and that distance of points to the real Pareto frontier and not just between each other will influence the value of this metric , if we leave it as it is. This can plainly be seen in Figure 4.4, where we see that while the distances between neighbouring points are roughly the same,

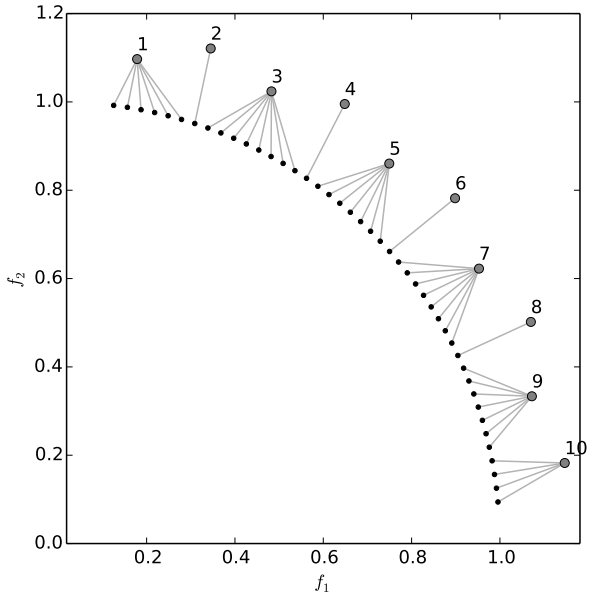


Figure 4.4: Closest points in reference set when approximation points differ in distance from reference set, but are uniform in relation to each other.

the number of reference set points that have these points as the closest ones, differs greatly. Here the number of points in reference set for which each point in the approximation is the closest one are 6, 1, 8, 1, 7, 1, 8, 1, 7 and 4 respectively. Arithmetic mean of those values is 4.4 and standard deviation is 2.97,which is large and would indicate our approximation, is a poor one.

To combat this, we perform an additional step before doing the measurement described above. During this step we project the approximation on to the reference set. This is done simply by replacing each point in the approximation with the point closest to it in the reference set. An illustration of this is given in Figure 4.5, which uses the same data as Figure 4.4 with projection points marked as larger black circles. If we again recalculate the standard deviation as befor, but using this projection instead, we get a value of approximately 0.49, which is the same as in the Figure 4.3 example. After performing the projection we proceed as described above. This avoids the issue of the metric being influenced by different distance to the real Pareto set. This way we get a metric that is in a sense orthogonal to the generational distance metric and is supposed to be used in conjunction with it.

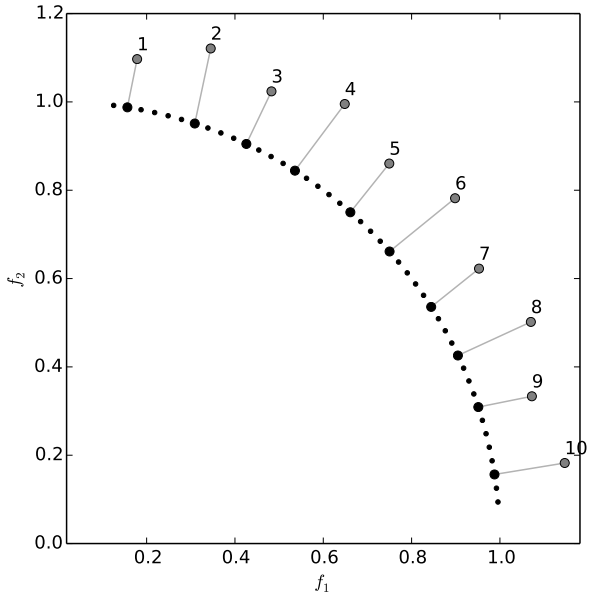


Figure 4.5: Projecting approximation to the the reference set.

We hope this makes it clear the reasoning behind this metric and justify our belief that it captures the aspect of Pareto set approximation we have been discussing. Let us now give a more formal definition of this metric that will allow others to calculate it easily for their data. The method used to replace each element of A with its projection in PF is given in Equation (4.1). Here and elsewhere,unless otherwise stated,we define $\min_{x \in X} f(x)$ to give the value of x for which the value of $f(x)$ is the smallest and not the value $f(x)$ itself. So, in Equation (4.1) projected point $\boldsymbol{p}_i$ is the point in reference set PF, for which Euclidean distance to approximation point $\boldsymbol{a}_i$ is the smallest. Here and elsewhere $\| \ldots \|$ stand for Euclidean norm.

$$\boldsymbol{p}_i = \min_{\boldsymbol{r} \in PF} \|\boldsymbol{r} - \boldsymbol{a}_i\| \tag{4.1}$$

In Equation (4.2) we count the number of points in PF for which point in projection p_i is the closest one.

$$c_i = \left| \left\{ \boldsymbol{r} \mid \boldsymbol{r} \in PF, \boldsymbol{p}_i = \min_{\boldsymbol{a} \in A} \|\boldsymbol{r} - \boldsymbol{a}\| \right\} \right| \tag{4.2}$$

Finally, the result of this metric is given in Equation(4.3) which is simply standard deviation of numbers $c_1, \ldots, c_{|A|}$ that contain the number of points in reference set that are the closest for each point in approximation when projected on to the reference set. Here $\bar{c}$ is the mean value of all $c_1, \ldots, c_{|A}$.

$$I_{UNI1}(A) = \sqrt{\frac{1}{|A|}\sum_{i=1}^{|A|}(c_i - \bar{c})^2} \tag{4.3}$$

We feel that this measure reasonably accurately reflects the difficult to formulate notion of Pareto set approximation uniformity. We hope this measure to be of use for researchers who want to evaluate their algorithms accurately when used on problems with known Pareto sets.

4.3.2 Performance Indicator I_{UNI^2}

While the previously discussed performance indicator is simple and inexpensive to compute, it fails in certain cases. Let us consider a three objective case to see when this happens. An example is given in Figure 4.6. In this case the Pareto frontier is flattened for the purposes of visualization. It will, however, have a shape of some surface in practice corresponding to the constraints set by the concepts of Pareto dominance and our assumption that the surface is mostly continuous. We can see the points in the approximation shown as little crosses and the closest points in the reference set colored differently for each cross. This in fact is similar to a Voronoi diagram. In fact the definition of a Voronoi diagram is just that - a partitioning of space to regions based on distance to a set of points specified in advance. In our case the regions are the sets of points from reference set that are the closest to each point in the approximation.

In the case considered in Figure 4.6 no issues are readily apparent. It would seem on the face of it that this works as intended. If the points are distributed in a uniform pattern, the number of points in the regions will be similar and the standard deviation will be small. It is not difficult to construct cases where this breaks down though. Consider, for example, the case in Figure 4.7. Here we can see that the number of points in each region is very similar. However, it is

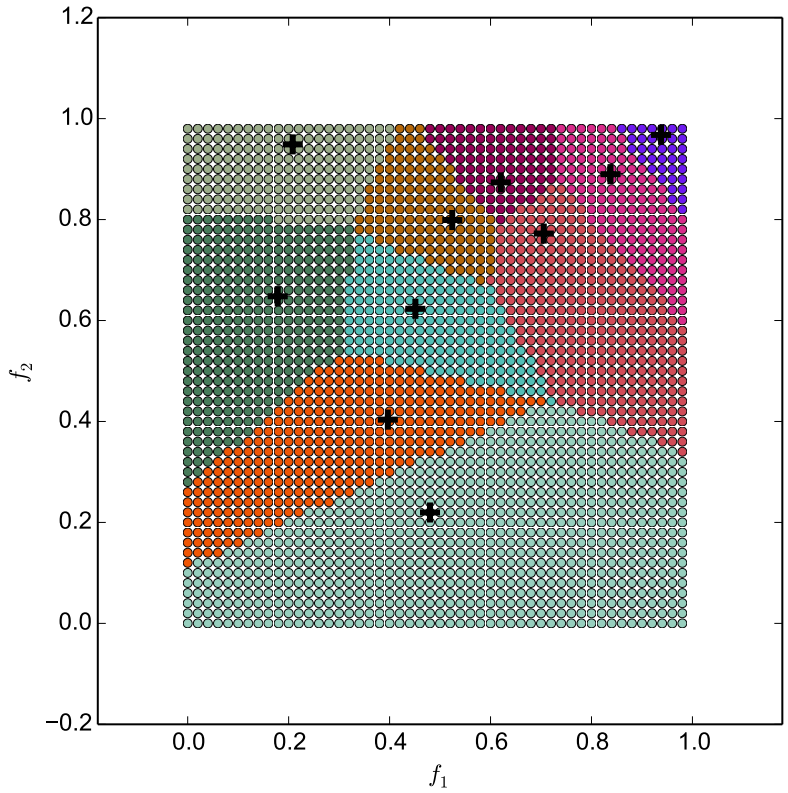


Figure 4.6: Approximation points with the points in reference for which those points are the closest ones. This assumes our reference set is flat.

apparent that performance indicator I_{UNI^1} will give misleading values in this case. It does not make sense to say that the coverage of the Pareto frontier by the approximation is uniform.

It would seem from this that simply counting the number of points in regions and expecting them to be roughly equal does not do it in our case. It seems that what we want to make sure is not only that they contain the same number of points, but also that they extend in every direction by the same amount. This second criterion is violated in the considered case. Here the regions extend in one dimension considerable more than in another. We need some other way of measuring the quality of spread.

Triangulation provides a natural way of covering a surface. For example, it can be argued that covering a surface with equilateral triangles could be the

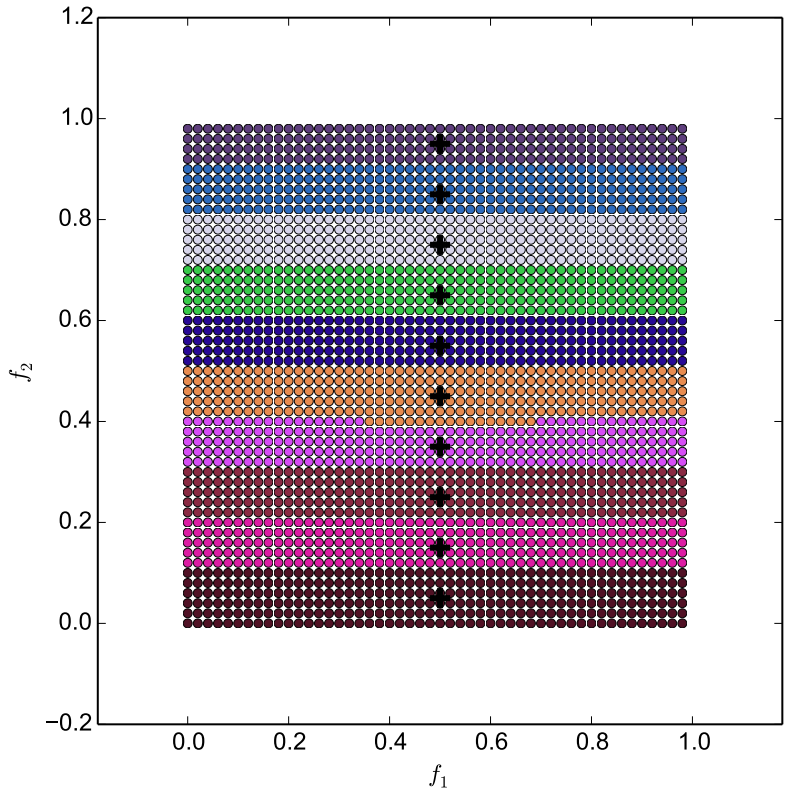


Figure 4.7: We can see that merely using the number of points in each area is not a good measure. Each area has roughly the same number of points, however, we can plainly see that the approximation does not cover the reference set uniformly.

definition (or at least a very good example) of uniform coverage. In our case the points would be at the vertices of the triangles. It makes sense then to triangulate the points in the approximation and measure how much they deviate from equilateral triangles. A simple way of doing that is to simply take the standard deviation of triangle edge lengths. It is a widely known result that a Delaunay triangulation is a dual graph of the Voronoi diagram. We have previously argued that the division of the reference set into regions depending on which point in the approximation they are the closest to is very similar to a Voronoi diagram. One simple way to triangulate the points in the approximation then is to simply connect the points that belong to regions that share a border. To see how, simply look at Figure 4.6 and connect the crosses that are in regions

that touch with edges.The best way to do this is an open question, though. The simplest way is to simply consider that two regions share a border if a point in one region is close enough to any point in the other region. What is close enough it would be a difficult question to answer here. If the points in the reference set were uniformly spaced, it would be easier. However,with the reference sets used in practice this is not always a reasonable assumption to make. Instead some heuristic has to be used. An example of this would be some multiple of the average distance to the closest neighbour. We summarize the procedure used to compute the performance indicator I_{UNI^2} in the list below.

1. Compute the projection of the approximation on to the reference set by replacing each point in the approximation by the closest point to it in the reference set. See Equation (4.1).
2. Group the points in reference set by the distance to the closest point in the projection. Each point in the projection gets as its region the points in the reference set that are the closest to it in terms of Euclidean distance.
3. Connect points that lie in adjacent regions with edges. A discussion of what should be considered an adjacent region can be found above.
4. Compute the standard deviation of said edge lengths. This is the numerical value of the performance indicator I_{UNI^2}.

A few things to consider further. First of all, for this to be perfectly accurate the triangulation would have to be performed on the surface itself using geodesic distance instead of Euclidean distances that we use here. This is possible, however, it would be fairly difficult to do and would make computing the performance indicator even more complicated. We will make the (maybe unfounded) assumption that Euclidean distances provide a reasonable approximation to geodesic distances in this case. The shape of the Pareto frontier surface is constrained by our assumption of continuity as well as the limits placed on it by the requirement that the points on the surface must be mutually non-dominated.

Another important issue to consider is the case when the approximation covers uniformly, but only a small portion of the reference set. If the approximation

covers, for example, a small patch of the surface of the Pareto frontier of a three objective problem and the coverage is uniform, our indicator will report good coverage. This can be remedied by comparing the area covered by the approximation to the area covered by the reference set. If the ratio of one to the other is too small the indicator result should not be relied upon. It is ideal to check this using the performance indicator I_{OS} since it is designed for just this purpose. As such they should be used together.

4.4 Proposed Heterogeneous Swarm HMOPSO-I

The essence of our proposed method is that it has two different particles in the same swarm. These particles are chosen so, as to emphasize two different aspects of multi-objective optimization. The particles use the same non-dominated point archive to exchange the information. The algorithm for updating the non-dominated point archive is given in Algorithm 2. The optimization runs for a fixed number of iterations. The method works as outlined in Algorithm 8. The update of $\boldsymbol{p}_i$ and $\boldsymbol{g}_i$ is different for different particle types. The parameter w is set to 0.4 for both particle types and $\rho_1 = \rho_2 = 1$. It is consistent with the parameters used for other multiobjective PSO algorithms. The swarm will contain different percentage of the two particles described below. Position vectors $\boldsymbol{x}_i$ are initialized to uniformly distributed random values in the allowed value range. The velocity vectors $\boldsymbol{v}_i$ are initialized to zero vectors.

4.4.1 Sigma Particle

This strategy was proposed by S. Mostaghim et al. [40]. It works by assigning certain vectors of values to each solution in the non-dominated point archive. The explanation for why these vectors are calculated this way can be found in the referenced paper. These values are calculated using the formula in Equation (4.4) for the two-objective case and Equation (4.5) for the three-objective case. It

Algorithm 8 Heterogeneous MOPSO Method HMOPSO-I

for $i \leftarrow 1, n$ **do**
    assign particle i to either "Sigma" or "Spread" group
    initialize $\boldsymbol{x}_i$ to random values in feasible range
    initialize $\boldsymbol{v}_i$ to zero
    $\boldsymbol{p}_i \leftarrow \boldsymbol{x}_i$
end for
while stop conditions not satisfied **do**
    update non-dominated point archive $\mathbf{A}$
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i$
    if particle i is of "sigma" type **then**
        update $\boldsymbol{p}_i$ if $\boldsymbol{x}_i \prec \boldsymbol{p}_i$
        update $\boldsymbol{g}_i$ with the solution in $\mathbf{A}$ that has the closest σ value to the σ value of $\boldsymbol{f}(\boldsymbol{x}_i)$ in terms of Euclidean distance
    else if particle i is of "spread" type **then**
        update $\boldsymbol{p}_i$ if not $\boldsymbol{p}_i \prec \boldsymbol{x}_i$
        update $\boldsymbol{g}_i$ with the solution from $\mathbf{A}$ that has the lowest crowding distance
    end if
    $\boldsymbol{v}_i \leftarrow w\boldsymbol{v}_i + \boldsymbol{U}_{(0,\rho_1)}(\boldsymbol{p}_i - \boldsymbol{x}_i) + \boldsymbol{U}_{(0,\rho_2)}(\boldsymbol{g}_i - \boldsymbol{x}_i)$
end while

can be generalized to more objectives.

$$\sigma = \frac{f_1^2 - f_2^2}{f_1^2 + f_2^2} \tag{4.4}$$

$$\boldsymbol{\sigma} = \begin{pmatrix} f_1^2 - f_2^2 \\ f_2^2 - f_3^2 \\ f_3^2 - f_1^2 \end{pmatrix} / \left(f_1^2 + f_2^2 + f_3^2\right) \tag{4.5}$$

When using this method for selecting $\boldsymbol{g}_i$, we first calculate $\boldsymbol{\sigma_i}$ for that particle and then we look for a point in the archive with the value of $\boldsymbol{\sigma}$ that is the closest in terms of the Euclidean distance. This point is then used as the best global position. Authors state that this method allows the particles to fly directly towards the Pareto-optimal front. The problem of updating $\boldsymbol{p}_i$ is solved by updating it only if the new solution dominates the previous $\boldsymbol{p}_i$. The authors also use a mutation operator of the form $x_{i,j} = x_{i,j} + U_{(0,1)}x_{i,j}$, where $U_{(0,1)}$ is a uniformly distributed random number from the value range $(0, 1)$ and $x_{i,j}$ is the

j-th coordinate of the position vector of the i-th particle. This mutation operator is applied with probability 0.05.

4.4.2 Spread Particle

It is a particle that is designed to make sure that approximated PF contains uniformly distributed points. Of course, this presupposes that PF is continuous. To this end two important changes to the velocity update rule are introduced. First of all, when selecting $\boldsymbol{g}_i$ the solution chosen from the archive is the one that has the lowest score of the crowding distance measure. Crowding distance measure is discussed later in this subsection. Furthermore, when updating $\boldsymbol{p}_i$, the new solution replaces the old one when the new one is not dominated by the old one. We have found that this is essential in encouraging the spread of points in approximated PF. All the other parameters are left exactly the same as in the case of the "sigma" particle.

The crowding distance is explained best with the help of Figure 3. Crowding distance is explained in section 2.6.1. Firstly, the elements of A are sorted in increasing order according to the values of the first objective. Then for each point with the index k we measure the perimeter of the hyper-rectangle, formed by the points with indices $k-1$ and $k+1$ as opposing corners. The first and last points (after sorting) always get infinite crowding distance. The archive is then sorted according to each objective in turn and values of the crowding distance are added together. Large values of the crowding distance measure indicate that the solution in question lies in a poorly explored section of PF. The concept of the crowding distance was first proposed by K. Deb et al. [12].

4.5 Comparing HMOPSO-I to Other Methods

To test the performance of the proposed algorithms, we use test problems from the 2009 IEEE Congress on Evolutionary Computation (CEC 2009) special session and competition for multi-objective optimization algorithms. We used only unconstrained problems with 2, 3, and 5 objectives. We did not try our

algorithm with constrained problems since that would add an extra layer of complexity we have yet to tackle. The problems in question can be found in a technical report by Q. Zhang et al. [71]. They are referred to as UF01 to UF13 in the same order as they appear in the aforementioned technical report. We use the PF reference sets provided by the competition organizers, when the performance indicator requires them.

We have done the experiments with the following types of PSO: one containing only the "sigma" particle, one containing 25% of "sigma" and 75% of "spread" particles, one containing 50% of "sigma" and 50% of "spread" particles, one containing 75% of "sigma" and 25% of "spread" particles and one containing only "spread" particles. We also repeated the experiments with OMPSO and SMPSO swarms. Each swarm consisted of 400 particles and ran for 750 iterations. That is consistent with the CEC 2009 competition conditions not to exceed 300000 function evaluations during the run of the algorithm.30 runs were done for each swarm and the resulting PF approximations were recorded. Each approximation was then evaluated, using the performance indicators described below and the average value of those indicators was recorded in the tables that had the indicator values in columns and problems in rows with the mean value of the indicator for that problem at the intersection. One such table was created for each type of swarm. The particle parameters are presented in the section describing them in detail. For two objective problems the size of the Pareto point archive was limited up to 100, for three objective problems it was limited up to 150 and for five objective problems it was limited up to 800. The tables are given in the section of results. Performance indicators used to analyze optimizer performance are described below. They were chosen so, as to cover both closeness to the real Pareto frontier and the uniformity of spread of solutions along it.

4.5.1 Results

The results of the problem UF01 are given in the tables. The best result of this problem in terms of the average value of indicator I_{GD} is given by a swarm consisting of "sigma" particles only. The best result in terms of I_{IGD} is obtained by the swarm consisting of equal parts "sigma" and "spread" particles. The best

Table 4.2: Results of OMPSO swarm.

Problem	$\mu(I_{GD})$	$\sigma(I_{GD})$	$\mu(I_{IGD})$	$\sigma(I_{IGD})$	$\mu(I_{SP})$	$\sigma(I_{SP})$
UF01	0.06905	0.03441	0.10369	0.01662	0.02236	0.01279
UF02	0.01651	0.00200	**0.02781**	0.00204	**0.00695**	0.00312
UF03	0.45414	0.02093	0.35599	0.00935	0.18575	0.04676
UF04	**0.04252**	0.00051	**0.04213**	0.00096	**0.00187**	0.00036
UF05	1.37283	0.46455	1.11808	0.39156	0.09555	0.04602
UF06	0.51504	0.10093	0.48957	0.08744	0.06882	0.07138
UF07	0.06128	0.04389	**0.07590**	0.02897	0.01725	0.00915
UF08	0.05181	0.01815	**0.17104**	0.01606	0.05497	0.02109
UF09	0.40111	0.29435	**0.24362**	0.06212	0.11583	0.06015
UF10	3.37871	0.51717	2.03196	0.19124	0.46891	0.12991
UF11	**0.52629**	0.04409	**0.28262**	0.01778	0.16452	0.01260
UF12	**2681.29**	42.09	1369.57	63.23	307.16	13.68
UF13	2.14740	0.01000	2.07414	0.00875	0.14741	0.02036

Table 4.3: Results of SMPSO swarm.

Problem	$\mu(I_{GD})$	$\sigma(I_{GD})$	$\mu(I_{IGD})$	$\sigma(I_{IGD})$	$\mu(I_{SP})$	$\sigma(I_{SP})$
UF01	0.09486	0.04132	0.12791	0.02004	0.03418	0.01402
UF02	0.05126	0.01087	0.07187	0.00592	0.02068	0.01248
UF03	0.43456	0.02102	0.34047	0.01136	**0.11433**	0.02402
UF04	0.05167	0.00180	0.04965	0.00161	0.00544	0.00112
UF05	1.49339	0.47240	1.28964	0.36340	0.15116	0.08617
UF06	0.61206	0.28469	0.55856	0.15319	0.06948	0.05083
UF07	0.08260	0.03271	0.10302	0.01516	0.03066	0.01287
UF08	0.10189	0.09792	0.23403	0.03382	0.07848	0.04395
UF09	1.23274	0.49392	0.41769	0.05122	0.24267	0.06939
UF10	3.53361	0.41184	2.66109	0.25422	0.60033	0.63174
UF11	0.91032	0.05816	0.48461	0.04117	0.19739	0.01552
UF12	2796.82	33.08	1406.97	75.45	**274.23**	10.98
UF13	**2.04956**	0.01111	**2.04030**	0.00683	**0.05768**	0.00254

result in terms of I_{SP} is obtained by a swarm consisting of 75% "sigma" and 25% "spread" particles.

When solving the problem UF02, the best result in terms of the average value of indicator I_{GD} is achieved by a swarm consisting of "sigma" particles only. The best result in terms of I_{IGD} is obtained by OMPSO. The best result in terms of

Table 4.4: Results of a swarm containing only "spread" particles.

Problem	$\mu(I_{GD})$	$\sigma(I_{GD})$	$\mu(I_{IGD})$	$\sigma(I_{IGD})$	$\mu(I_{SP})$	$\sigma(I_{SP})$
UF01	0.06817	0.02028	0.10462	0.08156	0.00768	0.00272
UF02	0.04098	0.00696	0.04573	0.00666	0.00973	0.00568
UF03	0.21168	0.08493	0.27269	0.04805	0.01706	0.04042
UF04	0.10523	0.01229	0.09536	0.01015	0.00795	0.00172
UF05	1.22954	0.36774	1.21375	0.31413	**0.02170**	0.01168
UF06	0.40376	0.19986	0.60084	0.15296	**0.01059**	0.00705
UF07	0.04529	0.02189	0.23641	0.20164	**0.00688**	0.00448
UF08	0.10195	0.17036	0.48437	0.21205	0.08228	0.20868
UF09	0.32427	0.25321	0.34454	0.07484	0.03362	0.01720
UF10	2.42616	1.33875	1.79859	0.91127	0.19728	0.09507
UF11	0.89914	0.36929	0.80596	0.26764	0.19904	0.03520
UF12	3360.46	354.23	1517.58	222.41	396.64	40.97
UF13	2.23954	0.01918	2.88993	0.71650	0.11573	0.09096

Table 4.5: Results of a swarm containing 25% "sigma" and 75% "spread" particles.

Problem	$\mu(I_{GD})$	$\sigma(I_{GD})$	$\mu(I_{IGD})$	$\sigma(I_{IGD})$	$\mu(I_{SP})$	$\sigma(I_{SP})$
UF01	0.02414	0.01290	0.08441	0.00944	0.00788	0.00452
UF02	0.00602	0.00161	0.03155	0.00711	0.02297	0.00904
UF03	0.19978	0.03287	0.22487	0.02458	0.01194	0.00617
UF04	0.06010	0.00437	0.05950	0.00429	0.01006	0.00406
UF05	0.82542	0.28921	0.79672	0.26289	0.02827	0.01469
UF06	0.24119	0.06698	0.32449	0.08230	0.01585	0.00631
UF07	0.05149	0.04572	0.15369	0.11075	0.01146	0.00659
UF08	**0.00526**	0.00365	0.42786	0.03444	**0.00810**	0.00445
UF09	0.16766	0.18463	0.26644	0.04242	0.03397	0.01703
UF10	0.99130	0.32329	0.75229	0.18676	0.11065	0.02345
UF11	0.56766	0.11193	0.34317	0.05283	0.16390	0.01249
UF12	3057.87	267.43	1271.86	128.81	313.42	33.97
UF13	2.23802	0.03190	2.77777	0.73307	0.10494	0.06007

I_{SP} is found by OMPSO swarm.

When solving the problem UF03, the best result for this problem in terms of average value of indicator I_{GD} is obtained by a swarm consisting of "sigma" particles only. The best result in terms of I_{IGD} is achieved by a swarm consisting of 75% "sigma" and 25% "spread particles. The best result in terms of I_{SP} is

Table 4.6: Results of a swarm containing 50% "sigma" and 50% "spread" particles.

Problem	$\mu(I_{GD})$	$\sigma(I_{GD})$	$\mu(I_{IGD})$	$\sigma(I_{IGD})$	$\mu(I_{SP})$	$\sigma(I_{SP})$
UF01	0.01575	0.00750	**0.08165**	0.01170	0.00755	0.00408
UF02	0.00356	0.00097	0.03322	0.00845	0.02342	0.01094
UF03	0.20803	0.03203	0.22959	0.02333	0.01274	0.00444
UF04	0.05922	0.00431	0.05885	0.00342	0.00889	0.00275
UF05	0.75801	0.24850	0.72940	0.21851	0.03930	0.02672
UF06	0.20716	0.04765	0.29145	0.07421	0.01823	0.00687
UF07	0.02692	0.03321	0.13525	0.13109	0.01402	0.00906
UF08	0.02204	0.09093	0.41220	0.03821	0.01209	0.01598
UF09	0.10947	0.11579	0.25501	0.02833	0.02975	0.01190
UF10	0.68876	0.34230	0.56741	0.12798	0.10353	0.03317
UF11	0.54122	0.08533	0.31286	0.03649	0.15534	0.01067
UF12	2960.47	269.57	**1159.32**	134.86	285.36	21.89
UF13	2.24200	0.02394	2.81058	0.75215	0.10794	0.06930

Table 4.7: Results of a swarm containing 75% "sigma" and 25% "spread" particles.

Problem	$\mu(I_{GD})$	$\sigma(I_{GD})$	$\mu(I_{IGD})$	$\sigma(I_{IGD})$	$\mu(I_{SP})$	$\sigma(I_{SP})$
UF01	0.01006	0.00511	0.08233	0.01088	**0.00578**	0.00282
UF02	0.00287	0.00077	0.03028	0.00773	0.02428	0.01120
UF03	0.20655	0.04960	**0.22134**	0.02318	0.01497	0.00531
UF04	0.05624	0.00355	0.05643	0.00254	0.01017	0.00426
UF05	**0.72383**	0.24078	**0.67690**	0.21466	0.04374	0.02319
UF06	0.18395	0.04477	**0.27422**	0.07109	0.02024	0.00965
UF07	0.02367	0.01932	0.13115	0.11200	0.01745	0.01957
UF08	0.00636	0.00246	0.36882	0.04637	0.01517	0.00600
UF09	0.04703	0.04223	0.25119	0.03753	**0.02222**	0.00928
UF10	0.59045	0.33056	0.49211	0.10640	**0.10199**	0.04317
UF11	0.53675	0.07272	0.28480	0.02629	0.15978	0.01058
UF12	2845.65	228.88	1176.29	107.62	278.17	21.29
UF13	2.25523	0.01301	3.02990	0.78049	0.12820	0.09193

found by SMPSO.

For the problem UF04 the best result in terms of average value of indicator I_{GD} is achieved by the OMPSO swarm. All the other indicators also acquire the best values using this swarm.

Table 4.8: Results of a swarm containing only "sigma" particles.

Problem	$\mu(I_{GD})$	$\sigma(I_{GD})$	$\mu(I_{IGD})$	$\sigma(I_{IGD})$	$\mu(I_{SP})$	$\sigma(I_{SP})$
UF01	**0.00233**	0.00079	0.09494	0.01478	0.00685	0.01190
UF02	**0.00156**	0.00049	0.03820	0.01034	0.02846	0.01168
UF03	**0.18832**	0.03259	0.24132	0.02361	0.02363	0.00983
UF04	0.05190	0.00353	0.05635	0.00334	0.01090	0.00522
UF05	0.73268	0.17750	0.70530	0.15353	0.09038	0.03583
UF06	**0.18186**	0.05563	0.28488	0.06941	0.03168	0.01406
UF07	**0.01474**	0.01900	0.15529	0.10162	0.02987	0.03458
UF08	0.00621	0.00140	0.33729	0.03899	0.01921	0.01192
UF09	**0.01228**	0.00706	0.29227	0.04003	0.02994	0.05062
UF10	**0.36093**	0.19300	**0.45719**	0.10291	0.08614	0.04453
UF11	0.80503	0.06572	0.39390	0.04423	**0.12534**	0.01229
UF12	2852.29	138.45	1214.99	72.94	293.52	21.45
UF13	2.26178	0.00274	3.62414	0.74534	0.12015	0.10284

For the problem UF05, the best result of the indicator I_{GD} is found by a swarm consisting of 75% "sigma" and 25% "spread" particles. The best result in terms of indicator I_{IGD} is also obtained by a swarm consisting of 75% "sigma" and 25% "spread" particles. The best result with regard to I_{SP} is obtained by a swarm consisting only of "spread" particles.

For the problem UF06, the best result of the indicator I_{GD} is achieved by a swarm containing only "sigma" particles. The best result with regard to the indicator I_{IGD} is found by a swarm containing 75% "sigma" and 25% "spread" particles. The best result with regard to indicator I_{SP} is achieved by a swarm containing only "spread" particles.

For the problem UF07, the best result with regard to indicator I_{GD} is found by a swarm consisting of "sigma" particles only. The best result with regards to indicator I_{IGD} is achieved by the OMPSO swarm. The best result with regard to the indicator I_{SP} is found by a swarm containing only "spread" particles.

For the problem UF08, the best result with regard to indicator I_{GD} is achieved by a swarm consisting of 25% "sigma" and 75% "spread" particles. The best result with regard to indicator I_{IGD} is found by OMPSO swarm. The best result with regard to indicator I_{SP} is obtained by the swarm consisting of 25% "sigma" and 75% "spread" particles.

For the problem UF09, the best result with regard to the indicator I_{GD} is found by the swarm consisting of "sigma" particles only. With regard to the indicator I_{IGD} the best result is obtained by OMPSO swarm. The best result with regard to the indicator I_{SP} is achieved by the swarm containing 75% "sigma" and 25% "spread" particles.

For the problem UF10, the best result with regard to the indicator I_{GD} is achieved by the swarm containing only "sigma" particles. The best result with regard to the indicator I_{IGD} is obtained by the swarm containing only "sigma" particles as well. The best result with regard to the indicator I_{SP} is achieved by the swarm consisting of 75% "sigma" and 25% "spread" particles.

For the problem UF11, the best result with regard to the indicator I_{GD} is obtained by the OMPSO swarm. The best result with regard to the indicator I_{IGD} is achieved by the OMPSO swarm. The best result with regard to the indicator I_{SP} is obtained by the swarm consisting only of "sigma" particles.

For the problem UF12, the best result with regard to the indicator I_{GD} is obtained by the OMPSO swarm. The best result with regard to the indicator I_{IGD} is achieved by the swarm consisting of the equal percentage of both types of particle. The best result with regard to indicator I_{SP} is achieved by the SMPSO swarm.

For the problem UF13, the best result with regard to the indicator I_{GD} is obtained by the SMPSO swarm. The best result with regard to the indicator I_{IGD} is achieved by the SMPSO swarm and the same is true for the indicator I_{SP}.

The OMPSO swarm yields the best results with regard to the indicator I_{GD} 3 times out of 13. With regard to the indicator I_{IGD} it yields the best results 6 times out 13. With regard to the indicator I_{SP} it yields the best results 2 times out of 13.

The SMPSO swarm yields the best results with regard to the indicator I_{GD} 1 time out of 13. With regard to the indicator I_{IGD} it yields the best results 1 time out of 13. With regard to the indicator I_{SP} it yields the best results 3 times out of 13.

The swarm consisting only of "spread" particles yields the best results 3 times out of 13 for the indicator I_{SP} and does not yield the best results for any other

indicators.

The heterogeneous swarms yield the best results with regard to the I_{GD} indicator 2 times out of 13. They yield the best result with regard to the indicator I_{IGD} 5 times out of 13. And they yield the best result with regard to the indicator I_{SP} 4 times out of 13.

The swarm consisting only of "sigma" particles yields the best result with regards to the indicator I_{GD} 7 times out of 13. It yields the best result with regard to the indicator I_{IGD} 1 time out of 13. It yields the best result with regard to the indicator I_{SP} 1 time out of 13.

It is important to note that when interpreting the results it is the best to use I_{GD} in combination with I_{SP}, since they are meant to measure the two aspects of multiobjective optimization performance independently. While I_{IGD} has been used as a single figure of merit in some sources it is not clear what is the influence of those aspects to the value of this indicator.

4.5.2 Conclusions

In the experiments we have shown how the advantages of certain types of PSO particle position and velocity update rules can be combined by using different types of particles in the same swarm, sharing information via a common non-dominated point archive. We also propose a new type of particle that is designed to take into account the information on the uniformity of PF approximation. This type of particle flies towards the areas of the search space that correspond to poorly covered areas of PF, assuming continuous PF. Other MOPSO algorithms, in contrast, try to combine these two aspects into a single particle's update rules. We believe the results of the empirical tests show that our approach is promising. Heterogeneous swarm have yielded the best results in terms of the average value of the indicator I_{IGD} almost in a half of all the test problems. In other cases it has come close to the best result. Thus, we feel that this technique is promising and should be explored in the future research. Good results have been obtained using the same percentage of the two different types of particle in the swarm. When optimizing by means of swarms consisting only of one of the two types of particle we see that our initial assumptions on performance of those particles

are justified. Swarms consisting only of "sigma" particles yield good values of the performance indicator I_{GD}. Swarms consisting only of "spread" particles yield good values of the performance indicator I_{SP}. Even when the value of I_{SP} is lower than that yielded by the other methods, it is still similar. In a half of the used test problems our approach gives better results than the commonly used MOPSO algorithms. Trying different types of particles in the swarm and dynamically changing types of particle during the evolution of the swarm could lead us to further performance improvements. A simple change of the type of particle, if there is no improvement after a set number of iterations, is one way of adding dynamic behaviour to the swarm.

4.6 Proposed Heterogeneous Swarm HMOPSO-II

There are two primary properties that we look for in the approximation - how close it is to the actual Pareto frontier and how well do the points in the approximation cover the actual Pareto frontier. There has been quite a bit of research into using PSO for these problems as of late as discussed in a later section. In developing such methods one often has to compromise and decide which attribute of the optimizers performance to prioritize. For example, some methods will try to get as close to the Pareto optimal set as possible, but they are prone to getting stuck in local minima. Others will attempt to ensure uniform spread of solutions along the real Pareto optimal set yet may sacrifice the ability to decrease the distance between approximation and the real Pareto-optimal set. Uniform distribution is important since it lets us see all the different aspects of a continuous Pareto set. A successful algorithm will try to combine those two properties. We propose a method,where we use particles designed separately for each of these aspects of performance together in a swarm sharing information via a non-dominated point archive. Such a swarm (called heterogeneous since it contains particles of different types) works by switching a particles' type to another one if that particle fails to improve over a set number of iterations. This way the swarm can adapt to the problem and use particle types that are most appropriate. We also propose a particle designed to ensure uniform spread of non-dominated points. We perform the analysis of this new PSO variant using a variety of metrics designed to evaluate multi-objective optimization algorithm

performance. Below we will describe the types of particles used in this swarm.

4.6.1 "Sigma" Particle

This particle is the same as the "sigma" particle in the description of the previous method.

4.6.2 "Closest" Particle

This particle works the same way as the "sigma" particle with one exception – $\boldsymbol{g}_i$ is set to the point in the non-dominated point archive that is the closest in terms of Euclidean distance in objective space to our particle. All other parameters and logic is exactly the same as in the "sigma" particle just the procedure by which $\boldsymbol{g}_i$ is chosen is different.

4.6.3 "Spread" Particle

This particle is the same as the "spread" particle in the description of the the previous method.

4.6.4 Heterogeneous Swarm

We propose to use a swarm where particle types are decided dynamically. Initially particle types are chosen from the three discussed above with equal probabilities. During the evolution of the swarm if the particle's $\boldsymbol{p}_i$ does not change for a set number of iterations that particle's type is changed and its $\boldsymbol{x}_i$ and all other properties are reset. All particle types use the same non-dominated point archive, which is the primary medium of information exchange for this swarm. Similar particle swarms for single objective optimization were described by Andries P. Engelbrecht [16] and Marco Antonio Montes de Oca et al. [10]. The algorithm is given in Algorithm 9.

Algorithm 9 Heterogeneous MOPSO Method HMOPSO-II

for $i \leftarrow 1, n$ **do**
    assign particle i to either "sigma", "closest" or "spread" group at random
    initialize $\boldsymbol{x}_i$ to random values in feasible range
    initialize $\boldsymbol{v}_i$ to zero
    $\boldsymbol{p}_i \leftarrow \boldsymbol{x}_i$
end for
while stop conditions not satisfied **do**
    update non-dominated point archive $\mathbf{A}$
    $\boldsymbol{x}_i \leftarrow \boldsymbol{x}_i + \boldsymbol{v}_i$
    if particle i is of "sigma" type **then**
        update $\boldsymbol{p}_i$ if $\boldsymbol{x}_i \prec \boldsymbol{p}_i$
        update $\boldsymbol{g}_i$ with the solution in $\mathbf{A}$ that has the closest σ value to the σ value of $\boldsymbol{f}(\boldsymbol{x}_i)$ in terms of Euclidean distance
    else if particle i is of "closest" type **then**
        update $\boldsymbol{p}_i$ if $\boldsymbol{x}_i \prec \boldsymbol{p}_i$
        update $\boldsymbol{g}_i$ with the solution in $\mathbf{A}$ that is closest to $\boldsymbol{x}_i$ in terms of Euclidean distance
    else if particle i is of "spread" type **then**
        update $\boldsymbol{p}_i$ if not $\boldsymbol{p}_i \prec \boldsymbol{x}_i$
        update $\boldsymbol{g}_i$ with the solution from $\mathbf{A}$ that has the lowest crowding distance
    end if
    if particle's i $\boldsymbol{p}_i$ wasn't updated for a specified number of iterations **then**
        assign particle i to either "sigma", "closest" or "spread" group at random
    end if
    $\boldsymbol{v}_i \leftarrow w\boldsymbol{v}_i + \boldsymbol{U}_{(0,\rho_1)}(\boldsymbol{p}_i - \boldsymbol{x}_i) + \boldsymbol{U}_{(0,\rho_2)}(\boldsymbol{g}_i - \boldsymbol{x}_i)$
end while

4.7 Experimental Evaluation of HMOPSO-II

To test the performance of the proposed algorithms we use test problems from 2009 IEEE Congress on Evolutionary Computation (CEC 2009) special session and competition for multi-objective optimization algorithms. We used only the first 7 unconstrained problems with two objectives. We did not try our algorithm on problems with 3 or 5 objectives and constrained problems since that would add an extra layer of complexity. However, there are no reasons to believe that our results do not translate to more objectives. The problems in question can be found in a technical report by Qingfu Zhang et al. [71]. They are referred to as

UF01 to UF07 in the same order as they appear in the aforementioned technical report. We use PF reference sets provided by the competition organizers when the metric used require them. You can see the PF reference sets plotted in the results section.

We ran the experiments for swarms containing "sigma" particles, "spread" particles and "closest" particles only as well as a heterogeneous swarm containing all three particle types. Each swarm consisted of 250 particles and ran for 1200 iterations. This is consistent with the CEC 2009 competition conditions of not exceeding 300000 function evaluations during the run of the algorithm.30 runs were done for each swarm and the resulting PF approximations were recorded. Each approximation was then measured using the metrics described below and the average value of those metrics was recorded into the tables that have the metrics for columns and problems for rows with the mean value of the metric for that problem at the intersection. One such table was created for each swarm type. Particle parameters are as follows: $c_1 = c_2 = 1.0$, $w = 0.4$. This is contrary to what constitutes "good" values of those parameters for single objective PSO, however, empirical tests suggest these work better. Also mutation Gaussian mutation was used with probability 0.05. Value $N(0, 0.1r)$, where r is the range between lower and upper bounds for that coordinate, was added to each coordinate with aforementioned probability. When using heterogeneous swarm particle type was changed to another one with the position and velocity being re-initialized if particle stagnated for 25 iterations. Stagnation is defined as particle's $\boldsymbol{p}_i$ not being updated.

The size of the Pareto point archive was limited to 100 for two objective problems. The tables are given as a part of the Results section along with the plots of the approximated PF for each problem and swarm that has the lowest value of IGD metric. IGD has been chosen because it is purported to measure both closeness to the true PF and uniformity of the approximation,so it is sometimes used as the only figure of merit for measuring optimizer performance. Metrics used to analyze optimizer performance are described in the subsections below. They have been chosen so, as to cover both closeness to the real Pareto frontier and the uniformity of spread of solutions along it.

Problem	IGD	GD	SP	MG	CM
UF01	0.09608	0.00390	0.00384	0.41662	0.10021
UF02	0.05913	0.00293	0.01857	0.30574	0.03731
UF03	0.22118	0.08248	0.02415	0.62886	0.05426
UF04	0.05633	0.05566	0.00879	0.10533	0.04965
UF05	0.60519	0.63110	0.08325	0.84161	0.15362
UF06	0.31305	0.19439	0.02680	0.53425	0.10645
UF07	0.11991	0.01057	0.02311	0.35233	0.09894

Table 4.9: Results for a swarm containing only sigma particles.

Problem	IGD	GD	SP	MG	CM
UF01	0.06189	0.00791	0.00623	0.24514	0.08910
UF02	0.04491	0.00258	0.01143	0.23481	0.04935
UF03	0.17045	0.17015	0.01771	0.39712	0.04349
UF04	0.05487	0.05489	0.00827	0.09442	0.04307
UF05	0.99020	1.17301	0.06908	1.12065	0.17490
UF06	0.31116	0.33294	0.02348	0.43837	0.12244
UF07	0.05386	0.01030	0.01586	0.18942	0.08037

Table 4.10: Results for a swarm containing only closest particles.

4.7.1 Results and Analysis

We can see the results for different swarm types running on problem UF01 in Figures 4.8a, 4.8b, 4.8c and 4.8d. Also in Tables 4.9, 4.10, 4.11 and 4.12. The inclination of the "spread" particle to get a uniformly covered non-dominated set can plainly be seen in Figure 4.8b. On the other hand, "sigma" particle

Problem	IGD	GD	SP	MG	CM
UF01	0.08357	0.05679	0.01199	0.28950	0.06294
UF02	0.05285	0.02067	0.00674	0.20797	0.01283
UF03	0.24235	0.09586	0.00637	0.67538	0.01457
UF04	0.09224	0.10241	0.00807	0.15422	0.03516
UF05	0.40920	0.23988	0.01869	0.81787	0.04059
UF06	0.41591	0.21311	0.01111	0.74441	0.02480
UF07	0.14598	0.04280	0.01451	0.41531	0.03978

Table 4.11: Results for a swarm containing only spread particles.

Problem	IGD	GD	SP	MG	CM
UF01	0.06110	0.01138	0.00499	0.25806	0.08174
UF02	0.02405	0.01082	0.00709	0.13189	0.02580
UF03	0.17354	0.09969	0.01031	0.52551	0.01965
UF04	0.06313	0.06704	0.00840	0.10469	0.02758
UF05	0.47906	0.51033	0.03454	0.75055	0.07856
UF06	0.28878	0.21472	0.01679	0.48759	0.04913
UF07	0.05653	0.02431	0.01062	0.17192	0.04860

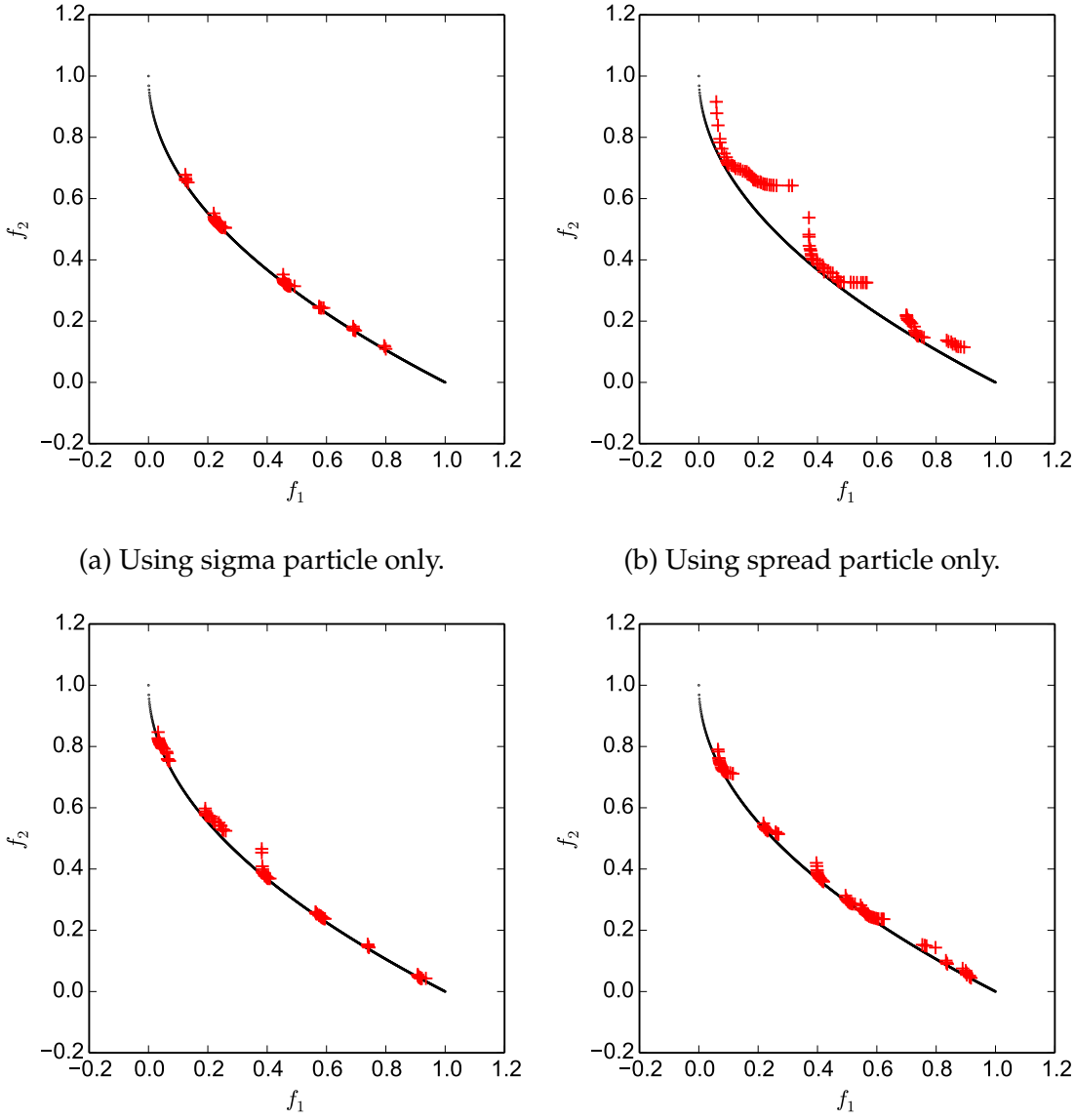
Table 4.12: Results for a heterogeneous swarm.

tries to get as close as possible to the real PF,which results in it uncovering small portions of it that are separated by large gaps.The same can be said of the"closest" particle type. If we look at IGD metric,we see that heterogeneous swarm does the best. In this only case our method does not seem to significantly improve performance compared to the "closest" particle alone,although CM and SP metrics do show a small improvement in the uniformity of the approximated PF. Another interesting thing is that the "closet" particle performs better than the more elaborate "sigma" particle.

The results for problem UF02 can be seen in Figures 4.9a, 4.9b, 4.9c and 4.9d. It can be seen that the swarm using only "sigma" particle has large parts of the PF missing, although, otherwise, it lies almost exactly on the PF. The same can be said of the "closest" particle. "Spread" particle shows very good coverage of the PF, although it is slightly further away. The same conclusion is indicated by the results tables. We can see that for the "spread" particle CM is the lowest indicating best coverage. We can also see that GD metric is an order of magnitude lower for "sigma" and the "closest" particles as opposed to the "spread" particle. This indicates that those two particle types give results much closer to the real PF. If we look at the heterogeneous swarm, we can see that it lowers CM compared to "sigma" and "closest" particles and it lowers GD compared to the "spread" particle. Which is in effect taking the best properties of the two particle types and negating each others disadvantages. This can be most dramatically seen when comparing Figure 4.9d to Figures 4.9a, 4.9b and 4.9c. We also see that IGD is the lowest for heterogeneous swarm.

If we look at the results for problem UF03 in the results tables we see an improve-

Figure 4.8: Results for UF01 using the various swarms.



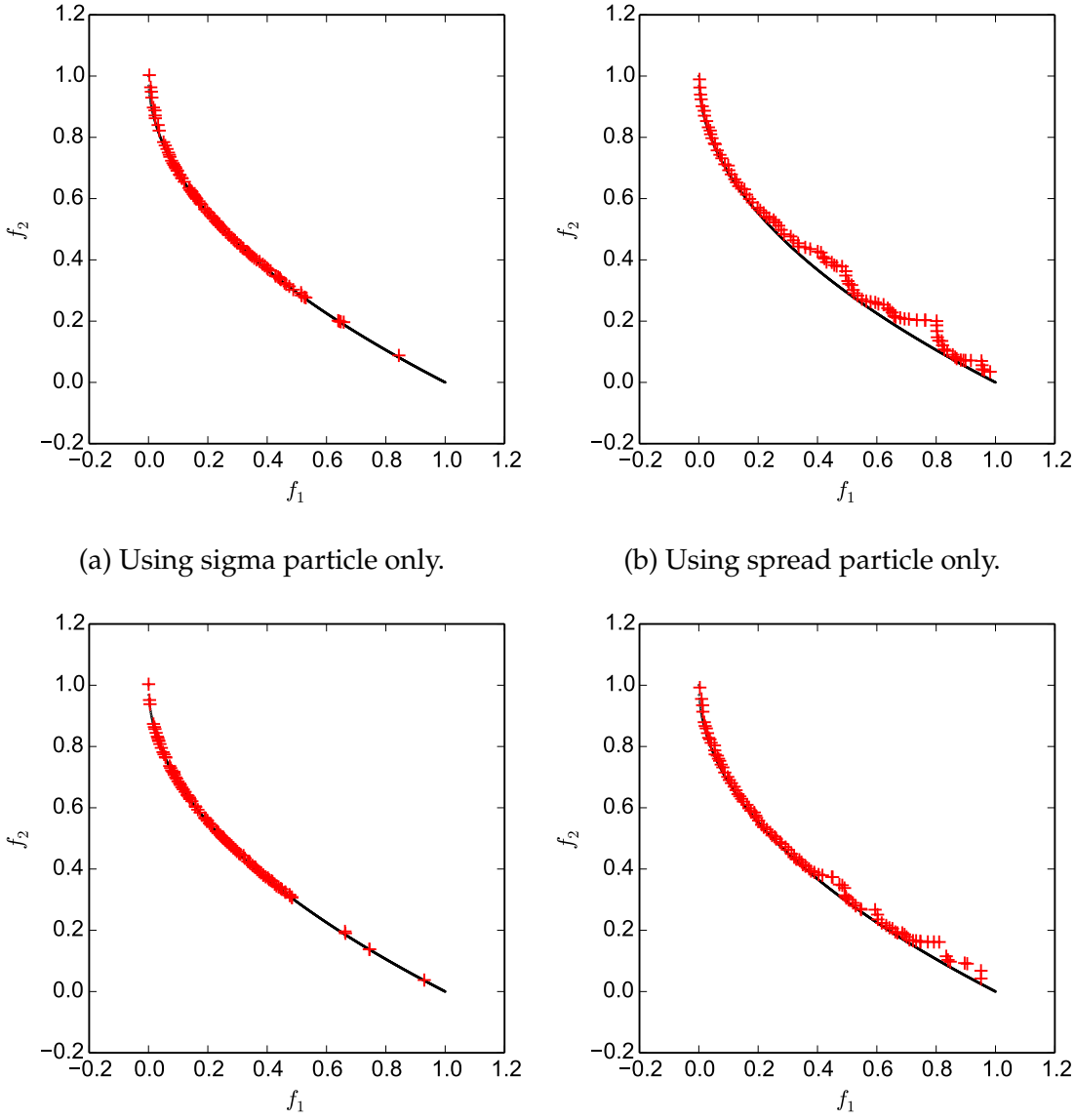
(a) Using sigma particle only.

(b) Using spread particle only.

(c) Using closest particle only.

(d) Using heterogeneous swarm.

Figure 4.9: Results for UF02 using the various swarms.



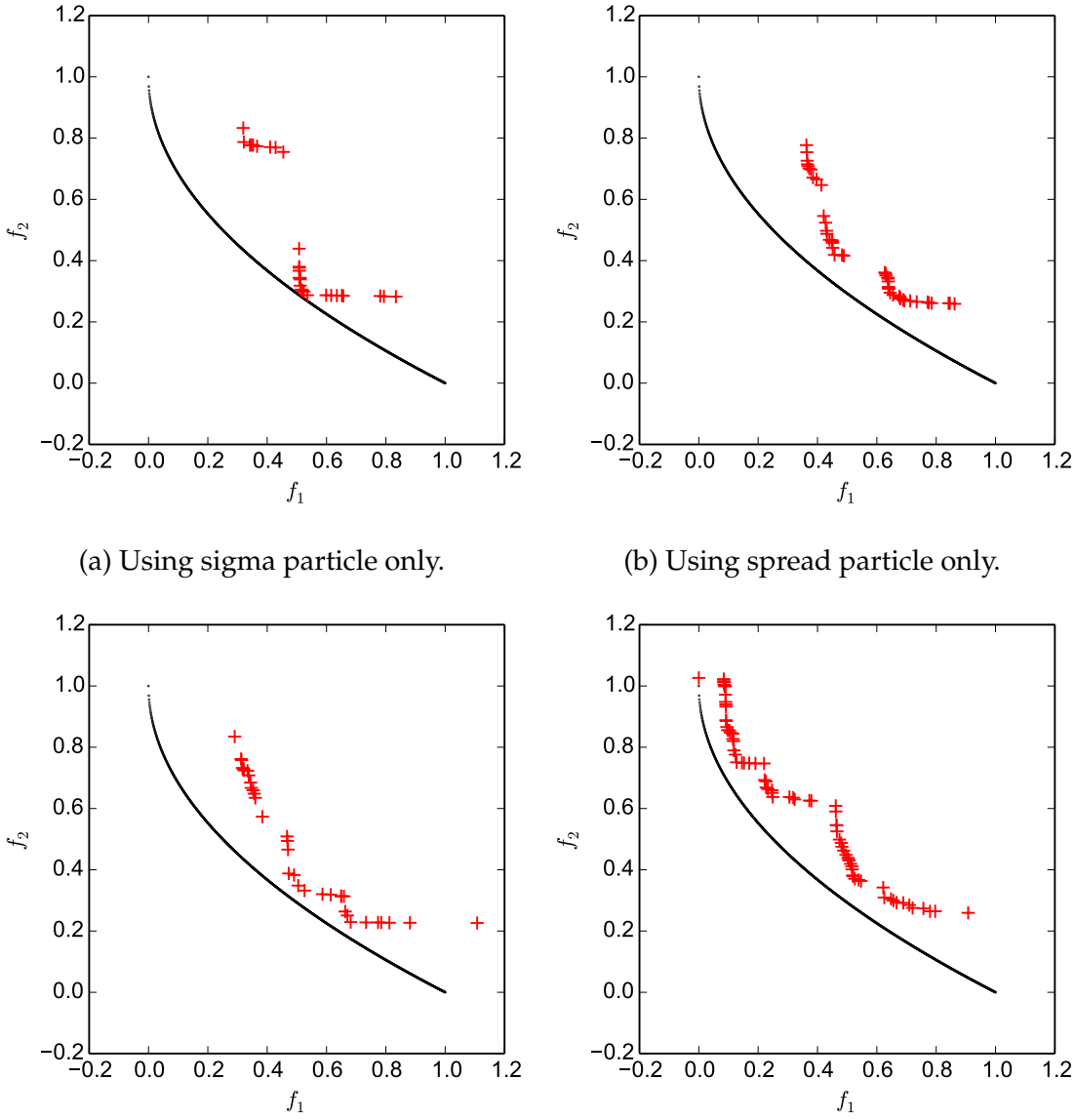
(a) Using sigma particle only.

(b) Using spread particle only.

(c) Using closest particle only.

(d) Using heterogeneous swarm.

Figure 4.10: Results for UF03 using the various swarms.



(a) Using sigma particle only.

(b) Using spread particle only.

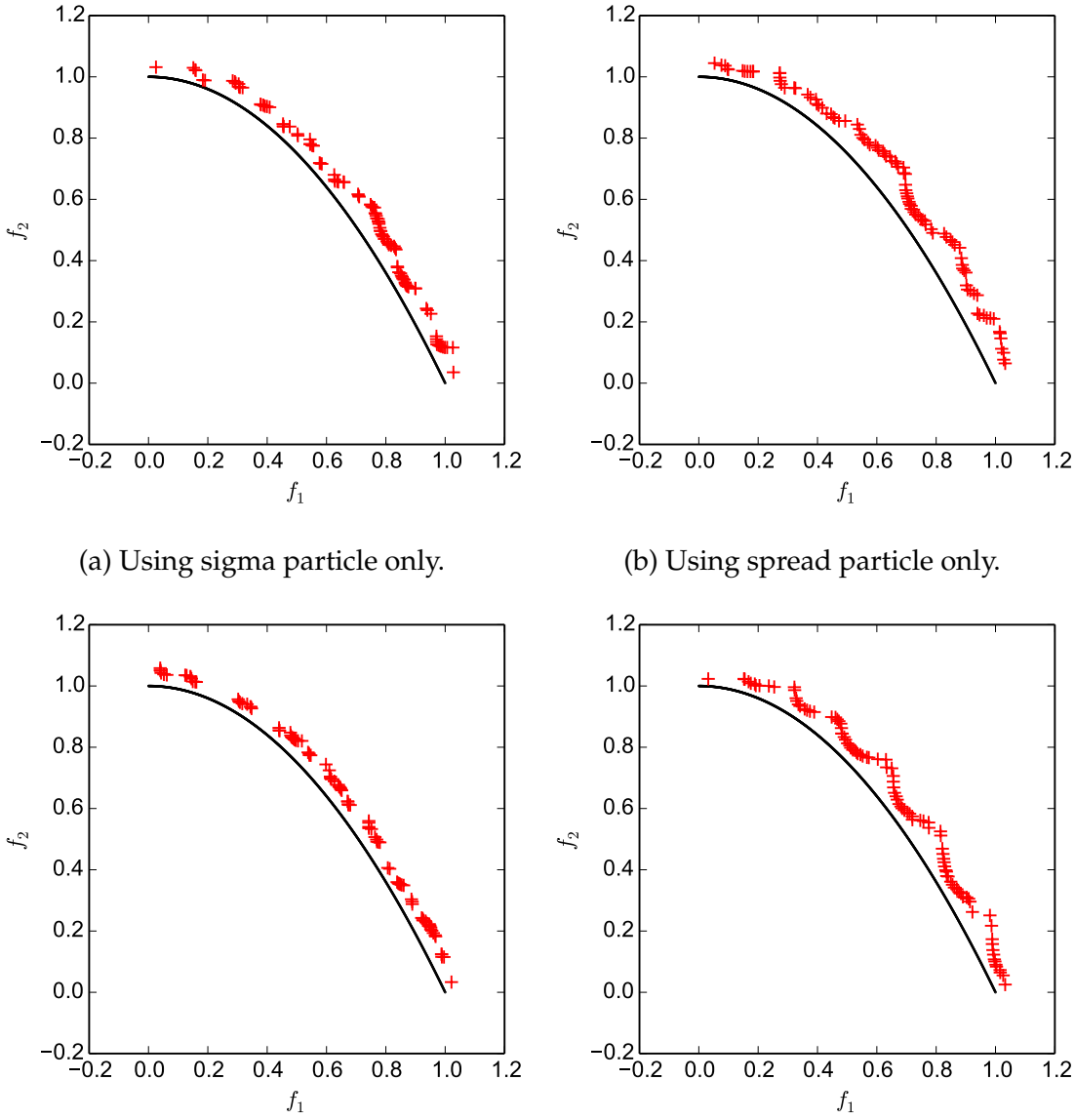
(c) Using closest particle only.

(d) Using heterogeneous swarm.

ment in both GD and CM if using the heterogeneous swarm. The improvement is even more apparent in Figure 4.10d, where we see that the heterogeneous swarm covers try to cover both uniformity and closeness to PF unlike "spread", "closest" or "sigma" particle swarms alone.

For problem UF04 we see the same results. GD metric is reduced for the heterogeneous swarms compared to "spread" particle only and CM metric is reduced

Figure 4.11: Results for UF04 using the various swarms.



(a) Using sigma particle only.

(b) Using spread particle only.

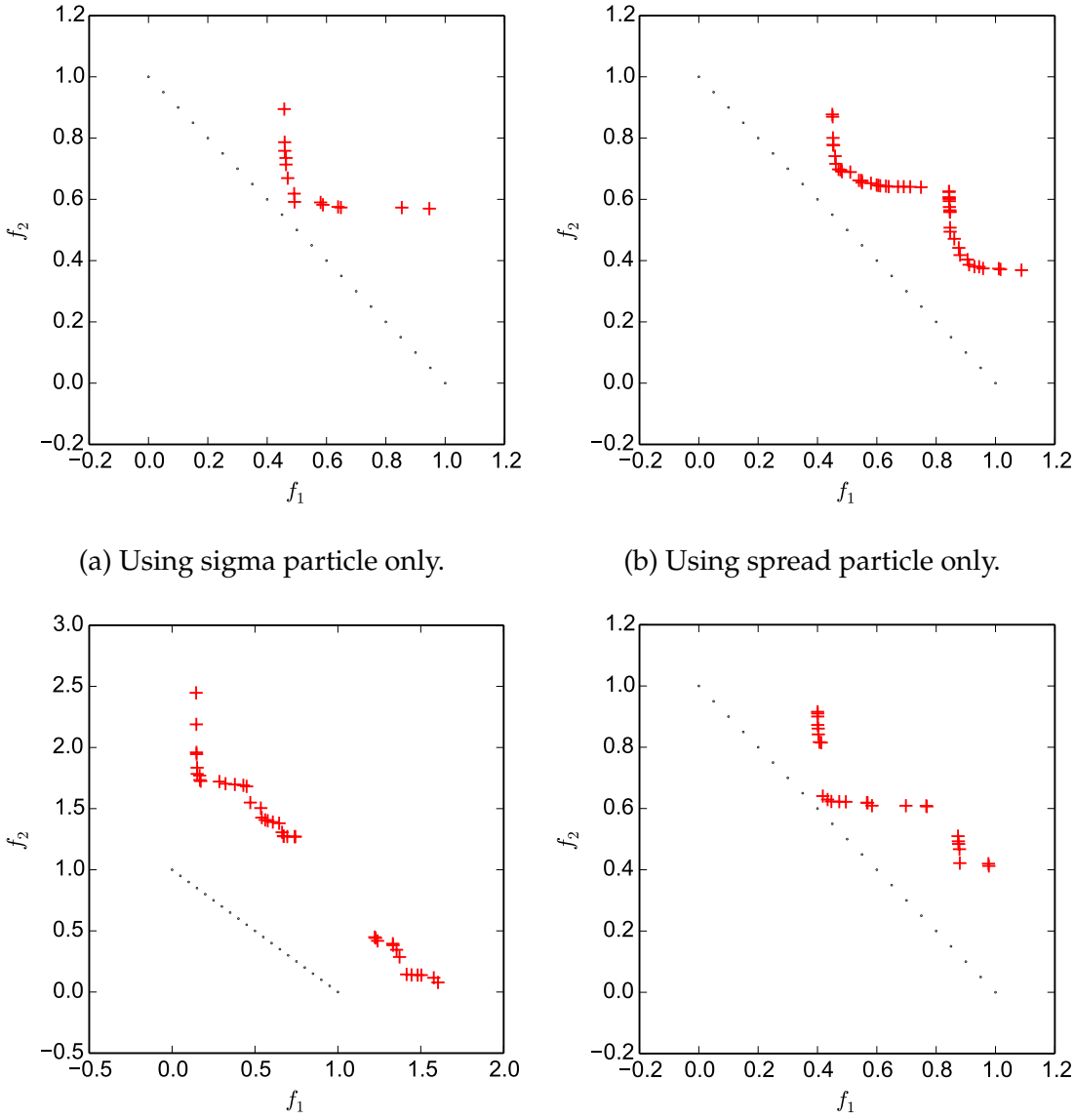
(c) Using closest particle only.

(d) Using heterogeneous swarm.

for heterogeneous swarms compared to using “sigma” or the“closest” particles only. This can also be seen in Figure 4.11d.

Problems UF05 and UF06 do not seem to be handled well by our algorithms. Their shared property is that they both have discrete Pareto sets which means that methods designed expecting continuous Pareto sets do not work well with them.Maybe, it can be solved by designing a separate particle for handling such

Figure 4.12: Results for UF05 using the various swarms.



(a) Using sigma particle only.

(b) Using spread particle only.

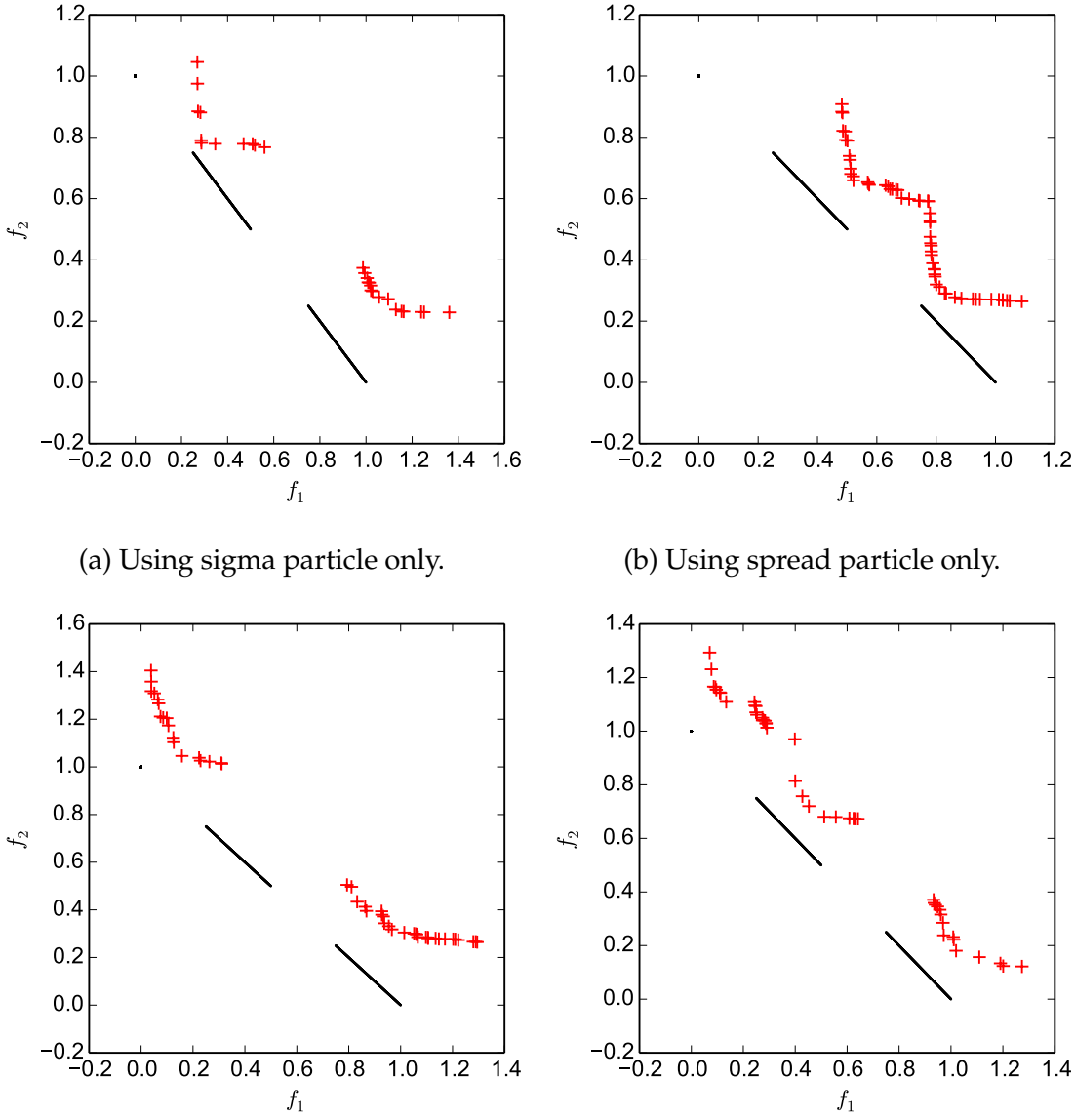
(c) Using closest particle only.

(d) Using heterogeneous swarm.

problems, although, what that would entail, is not entirely clear to us at the moment. Despite this,the trend of the two types of particles complementing each other's advantages is maintained as it can be seen in the results tables.

The advantages of using the proposed heterogeneous swarm can be seen in the case of UF07 as well. When comparing Figure 4.14d to Figures 4.14a, 4.14b and 4.14c we see that heterogeneous swarm gives the best performance of the

Figure 4.13: Results for UF06 using the various swarms.



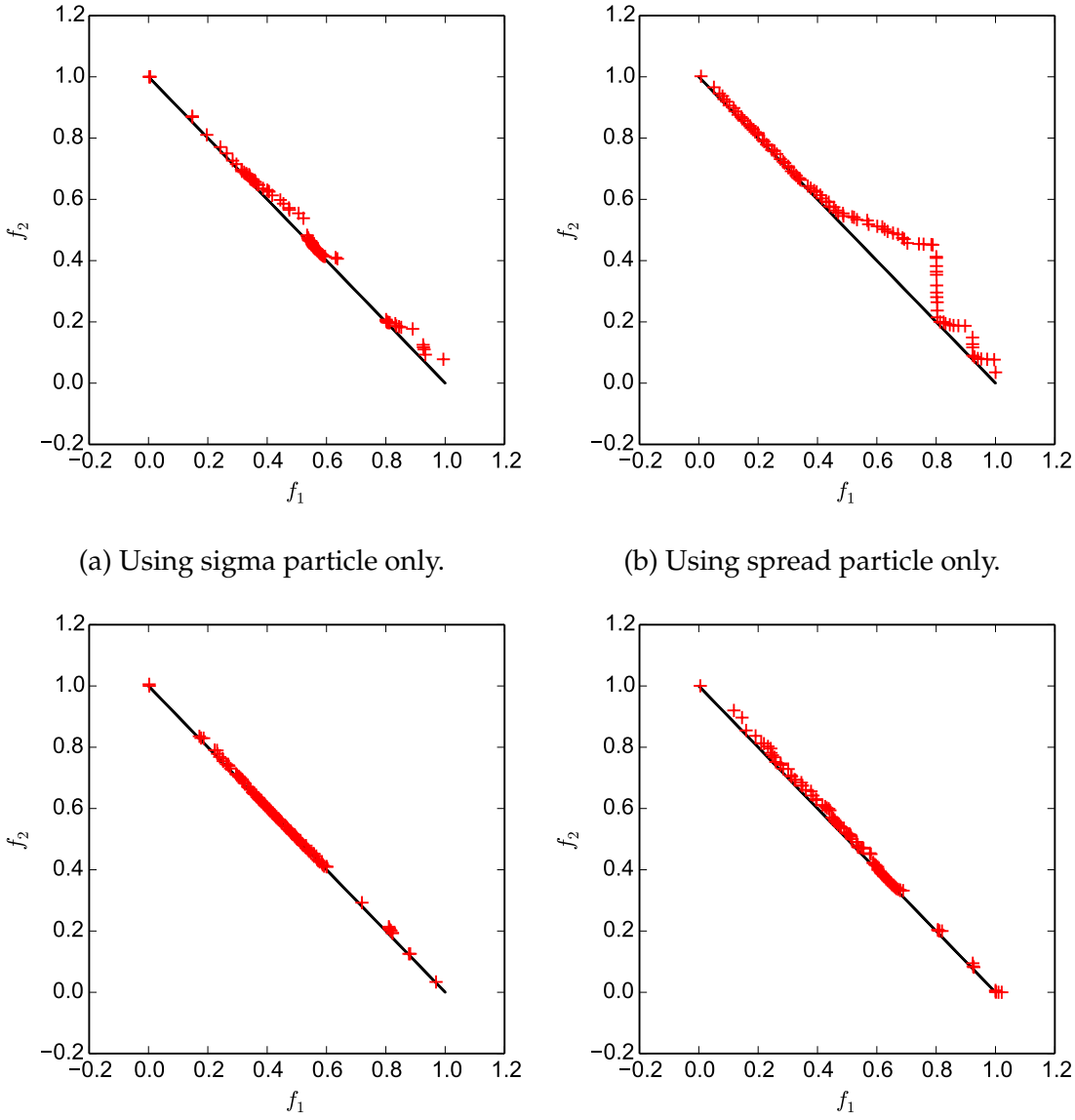
(a) Using sigma particle only.

(b) Using spread particle only.

(c) Using closest particle only.

(d) Using heterogeneous swarm.

Figure 4.14: Results for UF07 using the various swarms.



(a) Using sigma particle only.

(b) Using spread particle only.

(c) Using closest particle only.

(d) Using heterogeneous swarm.

four types visually. The resulting set has fewer missing areas of the PF,but also it is quite close to the real PF. Using only the "spread" particle gives good coverage, but poor distance,and using "sigma" or the"closest" particles gives poor coverage. Using heterogeneous swarm, on the other hand, we see an improvement in both areas. The same can be seen when looking at the values of various metrics in the results tables.

4.7.2 Conclusions

We have shown how disadvantages of certain types of PSO particle's position and velocity update rules can be offset by using different types of particles in the same swarm sharing information via a common non-dominated point archive. We also propose a new particle type that is designed to take into account the information about uniformity of the PF approximation. This particle type flies towards the areas of the search space that correspond to poorly covered areas of the PF assuming continuous PF.There are several key areas of work to develop these ideas that we suppose to be worthwhile:

1. Develop more particle types with different properties using different kinds of information to emphasize different areas of multi-objective algorithm performance. An example could include a particle that relies on the previous $\boldsymbol{p}_i$ values similar to the NSGA-II algorithm and others.
2. Incorporate the concept of PSO topology. It has been shown for the global optimization case that the way in which particles are connected into the neighborhood is important.
3. Improve the spread particle so,that it can be better generalized to more than two objectives.
4. Develop particles designed for problems with discrete Pareto frontiers. This will hopefully allow to get more accurate solutions to problems UF05 and UF06, since these are the ones,where our method does the worst.

Data in support of our thesis is given in terms of values of various metrics designed to measure multi-objective algorithm performance graphically as well.

4.8 Software Framework

A publicly available software library for Particle Swarm Optimization was developed during the course of writing this thesis. The library is unique in it's

modular design, allowing quick prototyping, testing and use of novel Particle Swarm Optimization methods. The library can be obtained from `https://bitbucket.org/bucket_brigade/swarm` using Mercurial version control system. Follow the following instructions to obtain the library and install it on a Unix-like system. This presupposes that you have Mercurial version control system installed.

```
$ hg clone https://bitbucket.org/bucket_brigade/swarm
$ cd swarm
$ sudo python setup.py install
```

From this point onwards you should be able to access the library from the Python language by importing from the package `swarmlab`. Examples of using the library will follow below.

All the software is written in the Python programming language with CPU intensive tasks implemented in C for efficiency. Over 35 different particle types have been implemented as found in literature. These different particle types can then be connected in user specified topologies to implement various swarms. Both multi-objective and single objective PSO methods are supported. Non-dominated point archive classes and methods are implemented for storing data collected during optimization runs.

Over 70 test problems for single objective optimization are included in the software and were collected from optimization literature. These include all kinds of problems including continuous, discontinuous, multi-modal, noisy, etc. All problems can be instantly used to test any PSO method implemented within our framework. There are overall 13 multi-objective test problems included, all taken from the CEC 2009 workshop/competition for multi-objective algorithms. These problems include, convex, non-convex and discrete problems of 2, 3 and 5 objectives. Each problem is 30 dimensional and is designed to challenge various aspects of multi-objective optimization method performance.

There are 5 topologies that can be built automatically with regards to user specified parameters in the software. These are: “gbest”, “lbest”, “grid”, “dsequence” and “random”. Topology “gbest” simply connects every particle to every other particle in the swarm. Topology “lbest” connects particles in a ring and you can specify how many particles to the left and to the right of particle’s

Topology	Parameter	Meaning	Parameter	Meaning
"gbest"	order	Number of particles in the swarm.	-	-
"lbest"	order	Number of particles in the swarm.	size	Number of neighbours for each particle (must be a power of two).
"grid"	order	Number of particles in the swarm (preferable if it has an integer square root).	-	-
"dsequence"	degrees	A list of vertex degrees.	-	-
"random"	order	Number of particles in the swarm.	size	Number of neighbours a particle will have.

Table 4.13: Available topologies and their parameters.

position in a ring to connect to. When using "grid" the particles will be connected in a two dimensional grid where each particle is connected to 4 particles on every side. The "dsequence" option will generate a random topology provided a degree sequence. A degree sequence of a graph is a non-increasing list of the degree of each vertex. Finally the random topology will create a random graph where each particle will be connected to a fixed number of neighbours at random with uniform probabilities. The topologies and their parameters are given in Table 4.13.

```
from swarmlab.swarm import Swarm
from swarmlab.generate import make_topology
from swarmlab.function.rastrigin import Rastrigin
from swarmlab.particle import ConstrictedParticle

if __name__ == '__main__':
    problem = Rastrigin(d=10)
    topology = make_topology('grid', {'order' : 100},
        ConstrictedParticle, {})
    swarm = Swarm(topology, problem)
    result = swarm.run(500)
```

The above code will create a swarm of 100 particle connected in to a "grid" topology. Each particle is of the type described by M. Clerc et al. [4]. The

swarm optimized the well-known Rastrigin test function in 10 dimensions. The swarm is run for 500 iterations, after which the result is returned as a dictionary containing the solution and the fitness value.

The current structure of the code makes it trivial to create new particle types. All the user has to do is inherit from the `Particle` class and override the methods that deal with velocity, movement updates and other particle functionality. An example of how it can be done to implement the Bare-Bones particle described by J. Kennedy [29] is provided in the listing below.

```
class BarebonesParticle(Particle):
    def move(self):
        self.x = self.v.copy()
        self.y = self.function(self.x)
        if self.y < self.py:
            self.py = self.y
            self.px = self.x.copy()

    def update_velocity(self):
        if len(self.neighbours) != 0:
            neighbour = min(self.neighbours)
            self.v = np.random.normal(
                0.5 * (self.px + neighbour.px),
                np.abs(self.px - neighbour.px) + 0.0000001)
        else:
            self.v = np.random.normal(
                self.px, np.abs(self.px) + 0.0000001)
```

After defining the particle this way one can start using it immediately the same way any other `Particle` would be used to create swarms and use them to optimize functions.

4.8.1 Non-dominated Point Archive Management

Implementing multi-objective PSO methods requires a way to collect non–dominated points after every iteration. An archive class should provide methods to append points to the archive and to retrieve points from the archive. Appending works by evaluating if the points currently in the archive dominate the point

in question. If the point is dominated by any of the points it is discarded. After appending the point the archive has still to be checked if all points are mutually non–dominated. The points that are not are to be removed from the archive. The retrieval of points from the archive is important because some algorithms rely on using information about the current approximation of the Pareto frontier to calculate velocity updates. Further conditions may also apply when admitting a point to an archive. For example we may want that the distance between two solutions in the archive not exceed some specified value. This is done so that the archive does not become over-saturated with points that are very close to each other. If the area in the Pareto frontier is small there is no point in having many points represent it. We implement two such options. The first of them is the crowding distance which was described in Section 2.6.1. The second one is the simple distance measurement we described previously. In the first case the user sets the upper limit for the archive size. If the new solution that was appended to the archive makes the archive go over the set limit, the solutions are sorted by their crowding distance and the solutions with lowest crowding distance values are discarded until the archive reaches the required size. In the second case a solution is simply not accepted if it is too close to another solution already in the archive. Below we provide an example of using the archive.

```
from moarchive import Solution, Archive
import random

if __name__ == '__main__':
    archive = Archive(100)
    for _ in range(100)
        archive.add_candidate(
            Solution(random.random(), random.random()))
    archive.visualize()
```

The above example will add one hundred random samples from the unit square, keep the mutually non–dominated solutions and then visualize the result using `matplotlib` library.

4.8.2 Capabilities with Regards to Multi-Objective Optimization

The library also supports multi-objective optimization. Multi-objective optimization is done in a very similar way to single objective optimization. Particles are connected in to a swarm using one of the supported topologies and share information about non-dominated solutions. Usually particles will override the `move` method for position update in order to account for the differences in comparing two solutions. Personal best solution is often updated if a solution dominates the old personal best solution. Likewise global best solutions are often taken from the non–dominated point archive instead of selecting it from the personal bests of the neighbours. Since the procedure for the evolution of the swarm is far less consistent than with single objective optimization methods a static method called `run` is introduced that you will use to run a swarm consisting of particular particle type. In this method the update of the non–dominated point archive and other such things are handled. An example of doing a multi–objective optimization run is given in the listing below.

```
from swarmlab.swarm import Swarm
from swarmlab.generate import make_topology
from swarmlab.particle import MSLechuga2005Particle
from swarmlab.mo_problems.cec2009 import problems
import matplotlib.pyplot as plt

if __name__ == '__main__':
    topology = make_topology('gbest', {'order' : 250},
        MSLechuga2005Particle, {})
    swarm = Swarm(topology, problems['uf01']())
    ax = plt.gca()
    MSLechuga2005Particle.run(100, 1000).visualize(ax)
    plt.show()
```

4.8.3 Computing on Clusters

If a larger empirical test is needed to evaluate algorithm performance usually a large number of tests has to be performed. Since the optimization runs can be done in parallel it is attractive to use super-computing clusters. The software

library provides a simple utility for computing on super-computing clusters using MPI messaging interface. The utility is called `mo_benchmark_run` and it's use is illustrated in the example below.

```
from swarmlab.swarm import Swarm
from swarmlab.generate import make_topology
from swarmlab.utilities import mo_benchmark_run
from swarmlab.particle import XHu2002Particle
import sys

if __name__ == '__main__':
    topology = make_topology('gbest', {'order' : 250},
        XHu2002Particle, {})
    mo_benchmark_run(
        XHu2002Particle.run, topology, sys.argv[1], repeats=30,
        iterations=1200, include_solution=False)
```

This example will run the swarm consisting of particles described by X. Hu et al. [22] for 1200 iterations 30 different times. Each of the 30 runs will be done as a separate MPI task. Now this example can be run on a cluster by running it using `mpiexec` command. For example the following is the batch file to run the example on a cluster using SLURM batch processing system.

```
#!/bin/sh
#SBATCH -p verylong
#SBATCH -J xhu2002
#SBATCH -n 31
#SBATCH --time=168:00:00

mpirun python xhu2002.py $1
```

The command line argument would then be supplied when submitting the job to specify what `.json` file to store the results in. After the run is done the results would be stored in that file in JSON format.

```
squeue xhu2002.sh xhu2002.json
```

Chapter 5

Conclusions

ïż£An experimental analysis of the existing MOPSO approaches was done in this thesis. Two new MOPSO methods were proposed and examined experimentally with regards to the existing MOPSO approaches. MOPSO approaches were clasiffied using a novel scheme relying on the algorithm design choices. Due to the limitations of the existing performance indicators when measuring Pareto solution spread, two new performance indicators were proposed. A large software framework for MOPSO algorithm prototyping and evaluation was written. The research done in this thesis leads to the following conclusions:

- Proposed multi-objective optimisation performance indicators can be used to measure the uniformity of the solution spread in Pareto front approximations. These (or similar) indicators are necessary because existing performance indicators suffer from the problems that are explained in chapter 4. Because of these problems existing indicators that are described in chapter 2, do not accurately capture the intuitive notion of the uniform coverage. The problems are solved by the proposed indicators by taking into careful consideration what it means to cover Pareto frontier uniformly with a discrete approximation of that frontier.
- It is desirable that MOPSO methods (or any other optimization method) could work well over many different problem classes without having to adjust the algorithm manually. It can be seen from the experiments

performed with the existing MOPSO methods that specific personal and global solution selection rules work best with the specific problem types. They work worse with other problem types. By using different types of the particle in the same swarm (so-called heterogeneous swarm) we have tried to improve on this. The advantage of this is that the swarm will use the type of the particle that is the best for the specific problem. The disadvantages of the particles will be cancelled out. It can be seen from the experimental evaluations that proposed heterogeneous multi-objective optimization algorithms perform well over a wide selection of performance indicators and test problems compared to the existing methods.

- It has long been known that the use of mutation operators can improve the performance of the single objective PSOs. However, comparative studies that contrast the methods which use mutation operators and those that do not, have not been performed in the field of multi-objective PSO. Because of the large number of MOPSO methods that have been surveyed in this research both with and without mutation, conclusions about the use can be made. It can be seen from the data that mutation can significantly improve the performance of MOPSO methods. With all test problem, the methods that don't use mutation end up with highest values of I_{IGD} and I_{GD} indicators by orders of magnitude. The only exception are the decomposition based methods, that do comparatively well.

- Decomposition based approaches work well on problems with discontinuous Pareto frontiers. This may be due to the fact that they do not make assumptions about the Pareto frontier being continuous the way the most MOPSO approaches do.

- Vector evaluated MOPSO methods have been earlier proposed for the use with the multi-objective problems. In vector evaluated approaches there are several populations of individuals each optimizing a single objective. Information is exchanged between populations. In the simplest case one population will take the best solution found by another population as it's best global solution. Since they optimize different objectives this will allow the swarms to explore solutions that optimize both of them. However, from the experimental results that have been gathered in chapter 3 it can be

seen that vector evaluated MOPSO methods do not work well compared to other MOPSO approaches. Several vector evaluated approaches have been tested and none of them come close in terms of the results to other approaches.

Other results accomplished during the writing of the thesis are as following:

- A large software library has been developed with the multiple PSO methods and test problems available. The library is designed for empirical experiments that can be run on supercomputing clusters. The library is publicly available and open-source. It's modular design allows the user to create new particle types and swarm topologies quickly. Numerical experiments evaluating the performance of these methods can then be performed using the included suite of the test problems and performance indicators.
- A systematized overview and classification of most MOPSO methods described in literature is presented. Previous classification schemes rely entirely on a single underlying principle to classify the methods. Here, several important factors are identified: underlying idea, method of solution diversity control and mutation. These three choices in the method design are then used as the basis for the MOPSO classification.

Appendix A

Result Plots

In this appendix we present results optained by the different PSO multi-objective optimization in a visual form. We only consider two objective problems. Problems with more objectives are not presented here because they are too difficult to visualize and interpret. The results for problems with more than two objectives are given in table form where the tables contain mean values of various performance indicators. Each page will contain figures for 7 problems in our test problem suite. In each figure all Pareto Frontier approximations are plotted. That is they represent results from all optimization runs we performed on those problems. While visualizations cannot replace numerical estimates of performance they are useful in quickly spotting problems with methods or quickly picking out the ones most useful for the given situation.

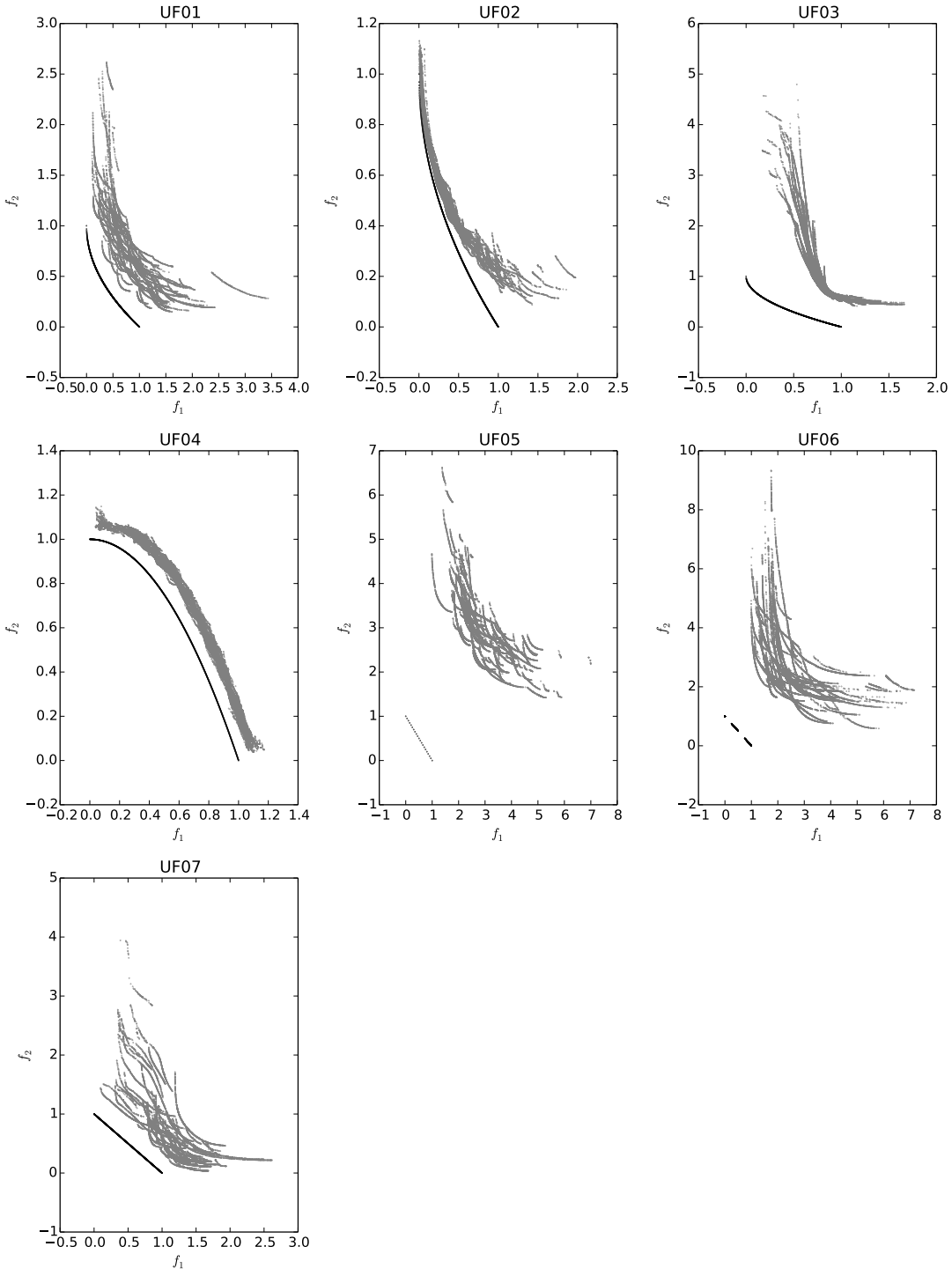


Figure A.1: PF approximations obtained by PSO variant described by C. A. C. Coello et al. (2002)

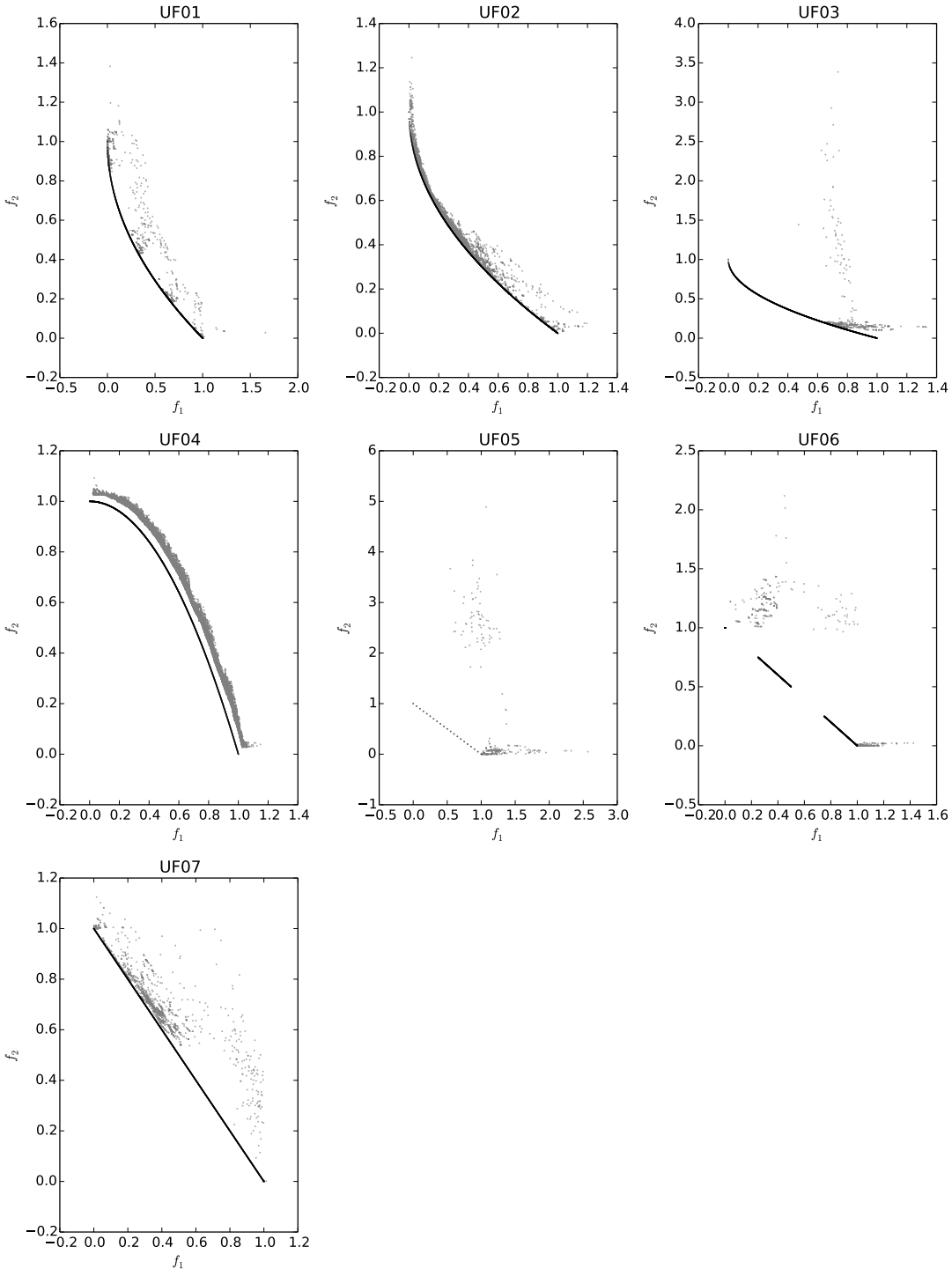


Figure A.2: PF approximations obtained by PSO variant described by X. Hu et al. (2002)

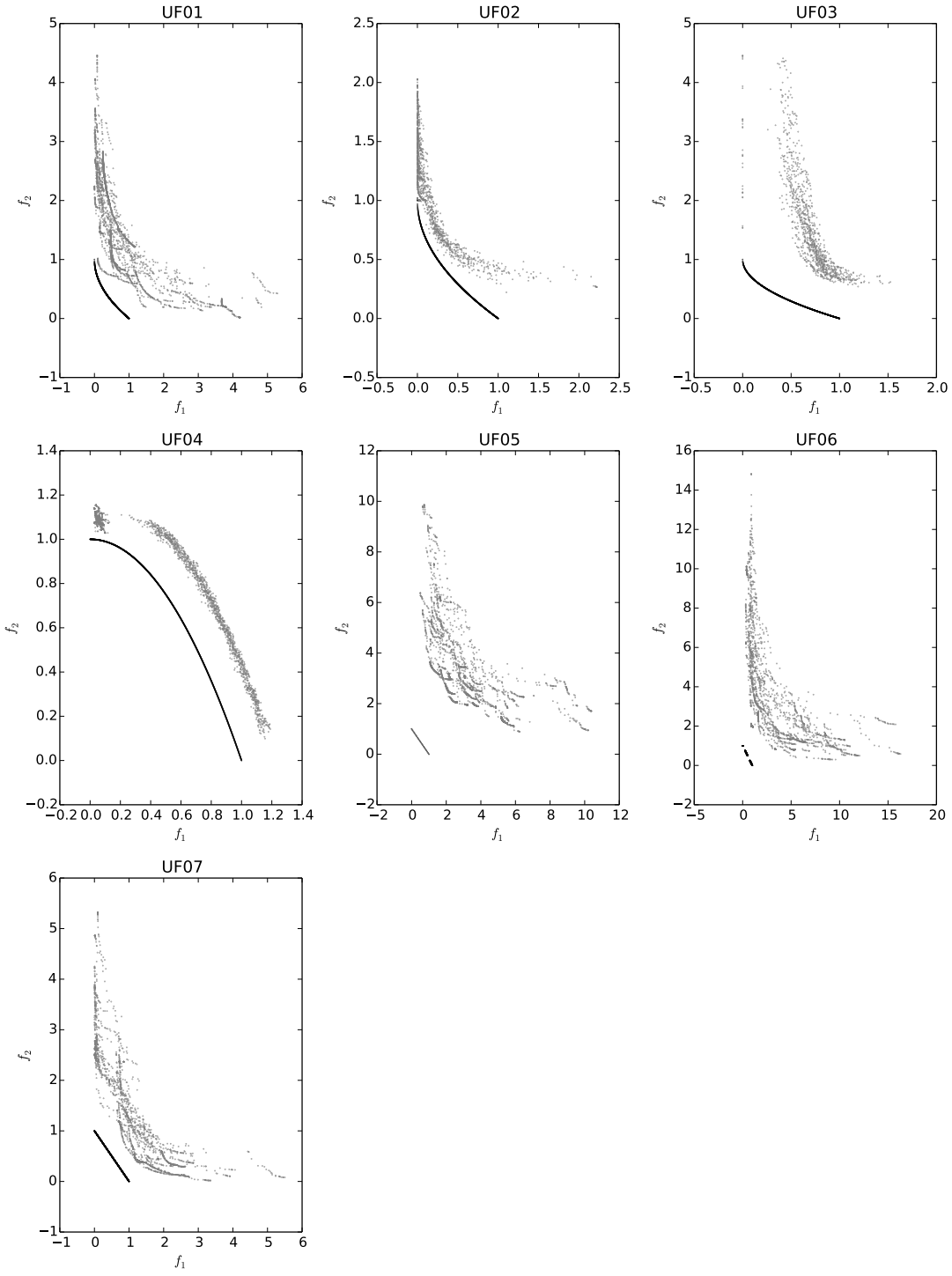


Figure A.3: PF approximations obtained by PSO variant using Bang-Bang Weighted Aggregation described by K. E. Parsopoulos et al. (2002)

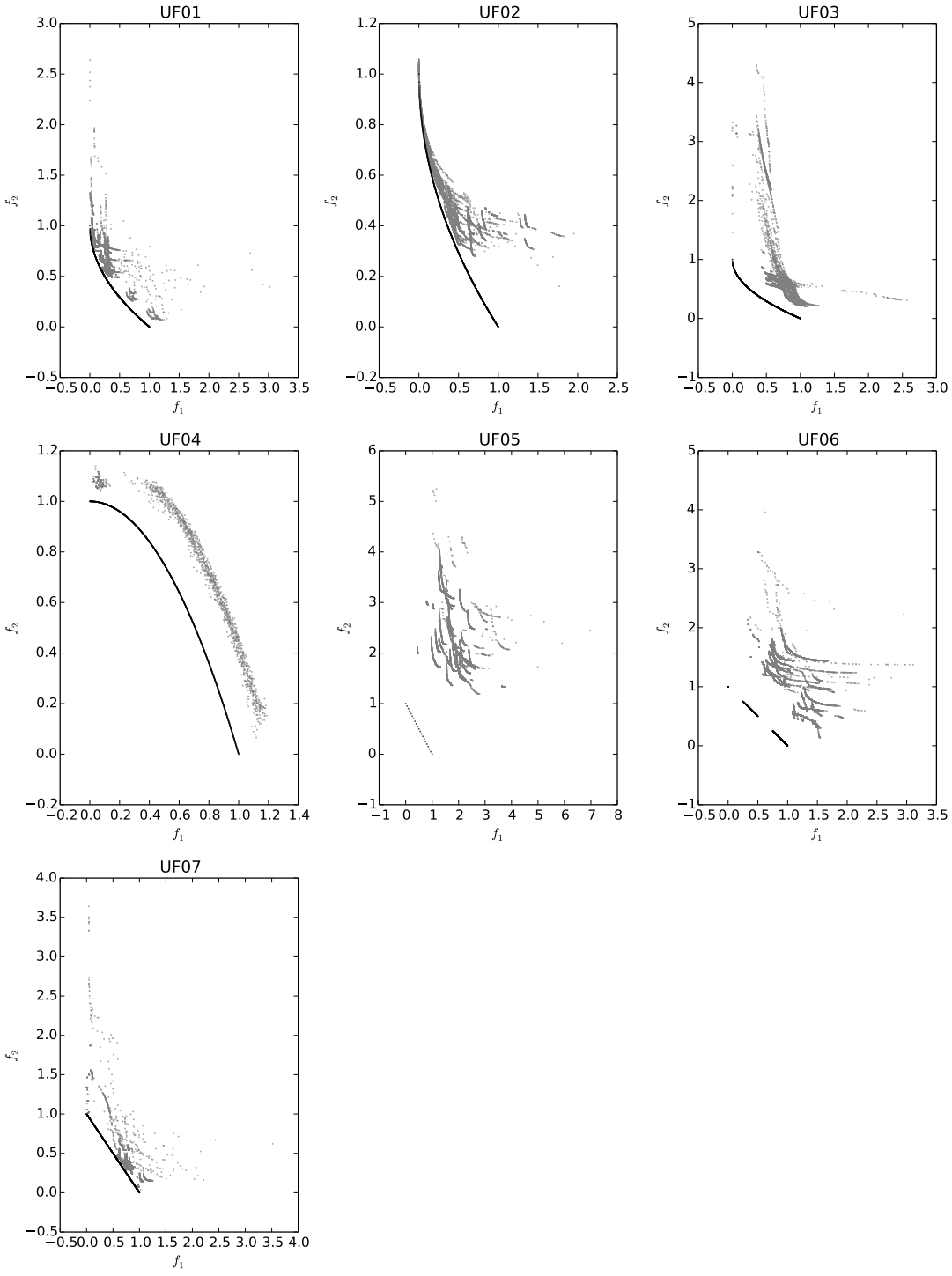


Figure A.4: PF approximations obtained by PSO variant using Dynamic Weighted Aggregation described by K. E. Parsopoulos et al. (2002)

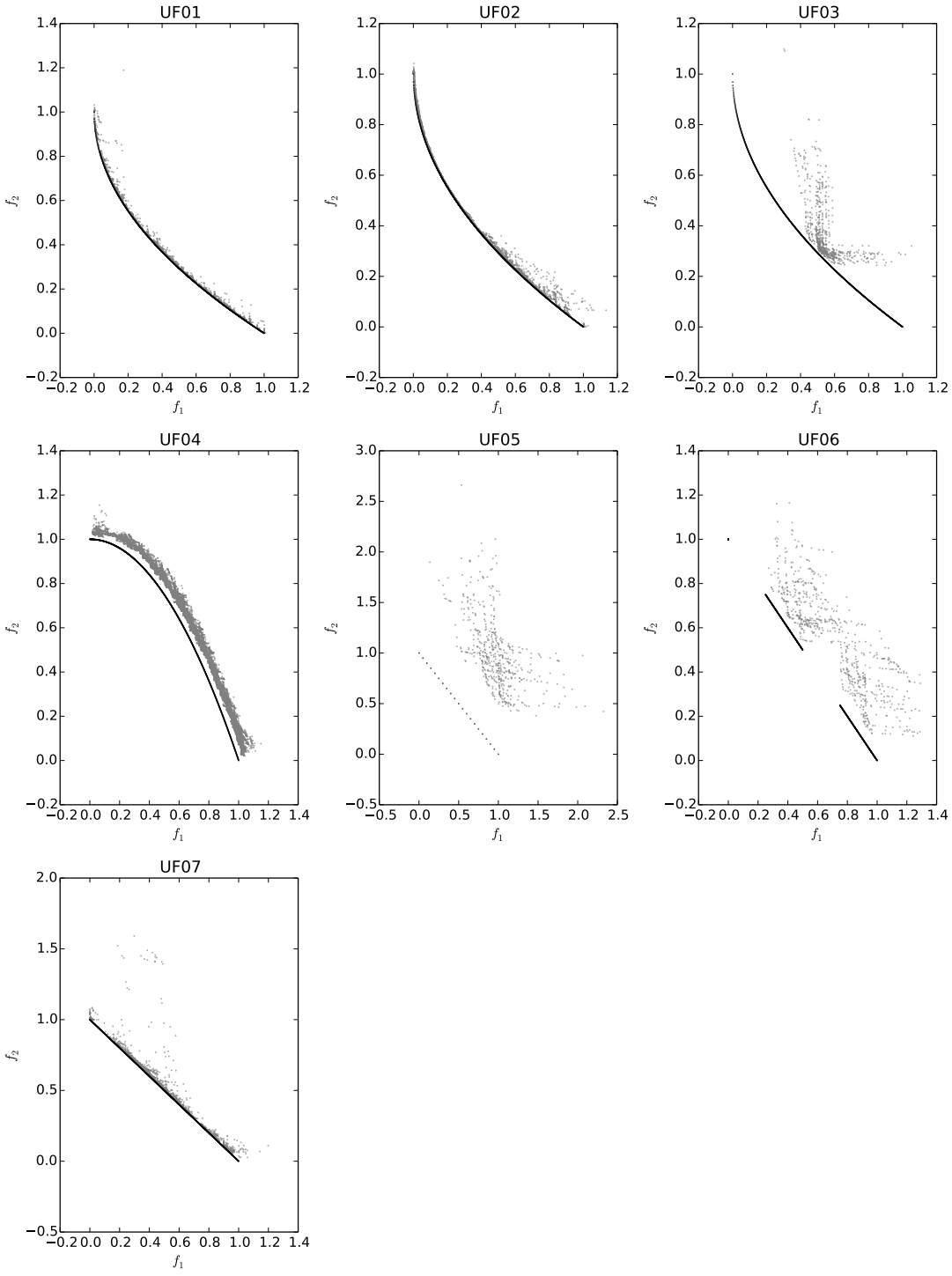


Figure A.5: PF approximations obtained by PSO variant described by S. Mostaghim et al. (2003)

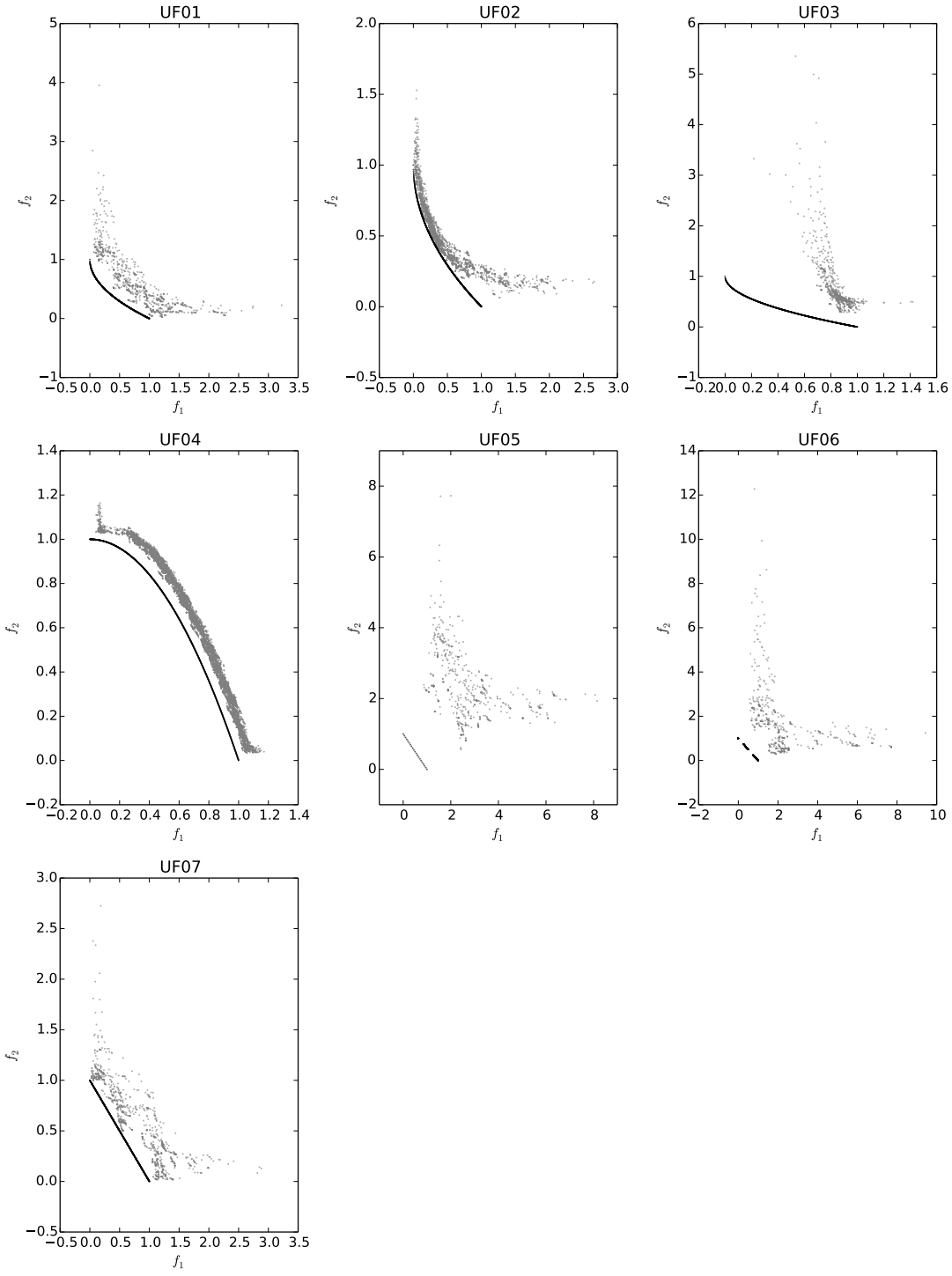


Figure A.6: PF approximations obtained by PSO variant described by X. Hu et al. (2003)

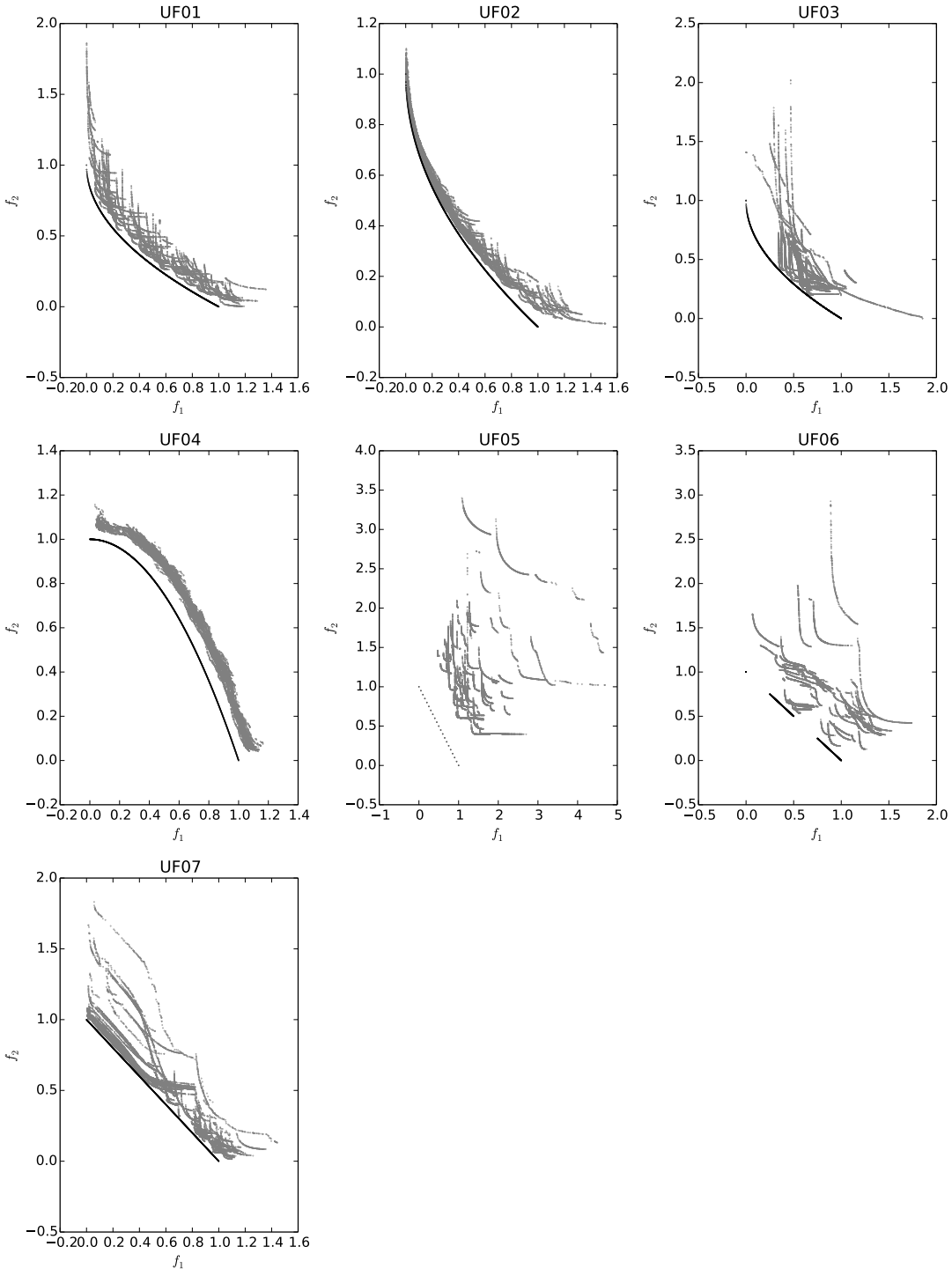


Figure A.7: PF approximations obtained by PSO variant described by C. A. C. Coello et al. (2004)

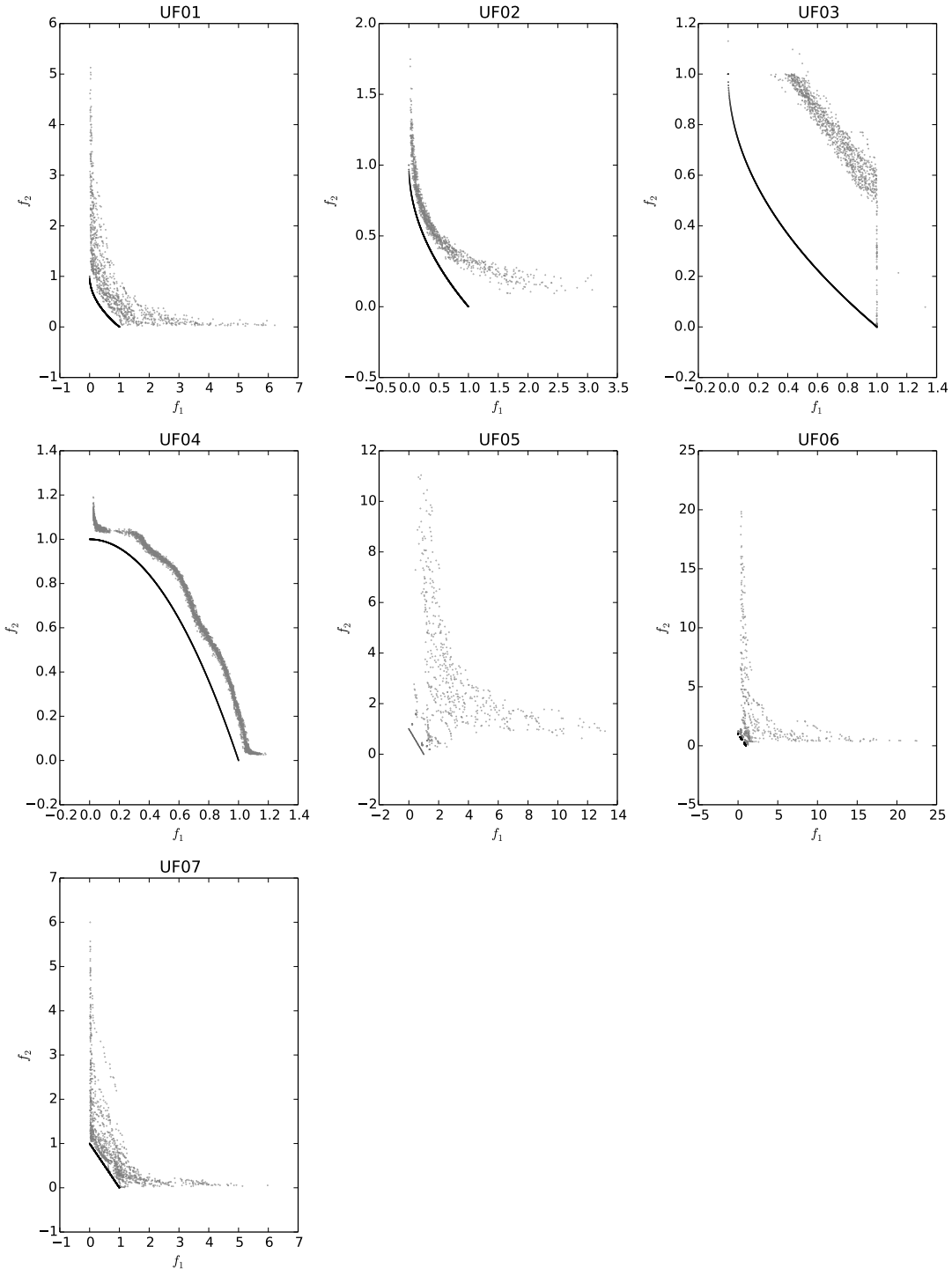


Figure A.8: PF approximations obtained by PSO variant described by K. E. Parsopoulos et al. (2004)

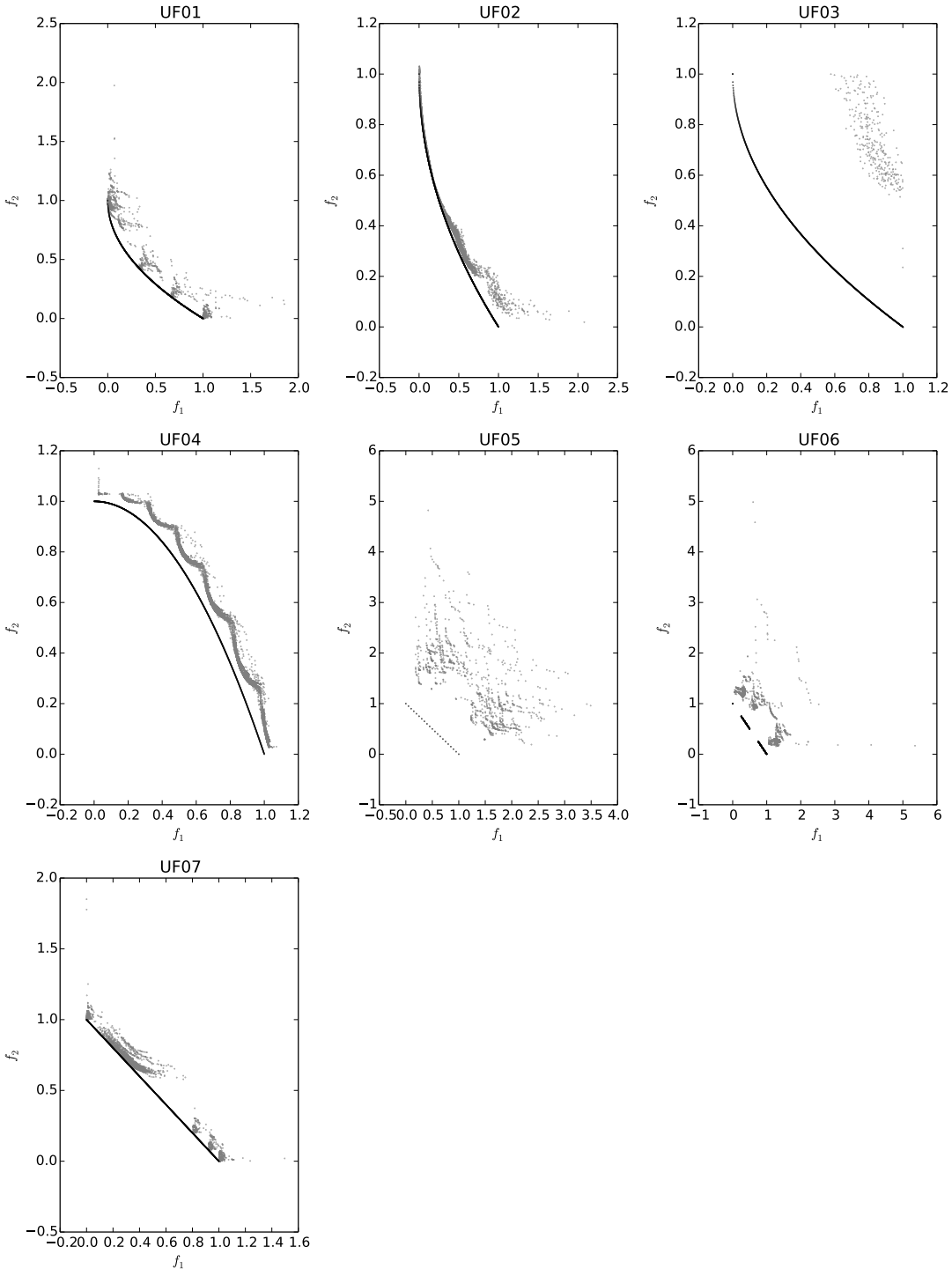


Figure A.9: PF approximations obtained by PSO variant described by C. R. Raquel et al. (2005)

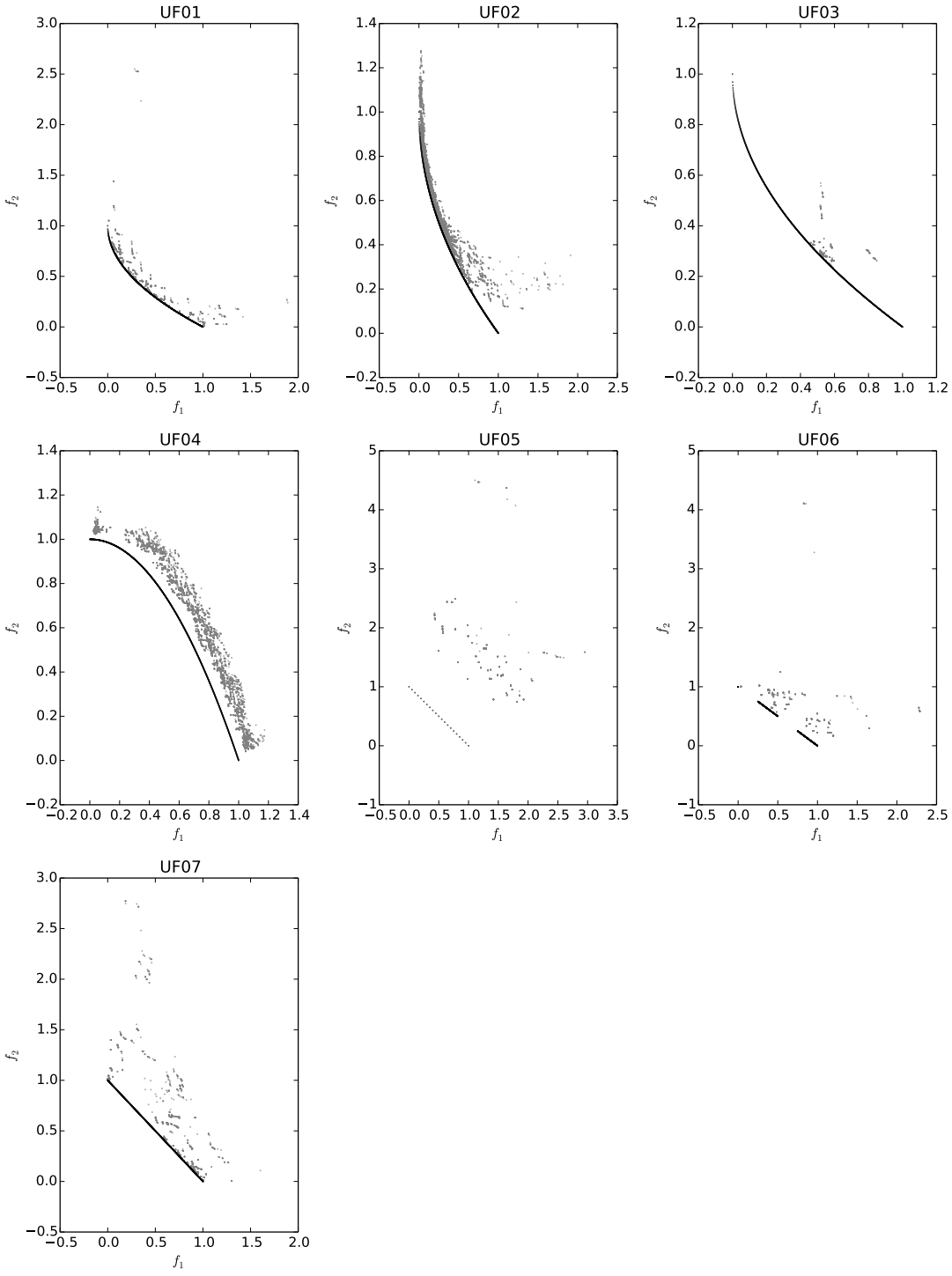


Figure A.10: PF approximations obtained by PSO variant using RANDOM leader selection scheme described by J. E. Alvarez-Benitez et al. (2005)

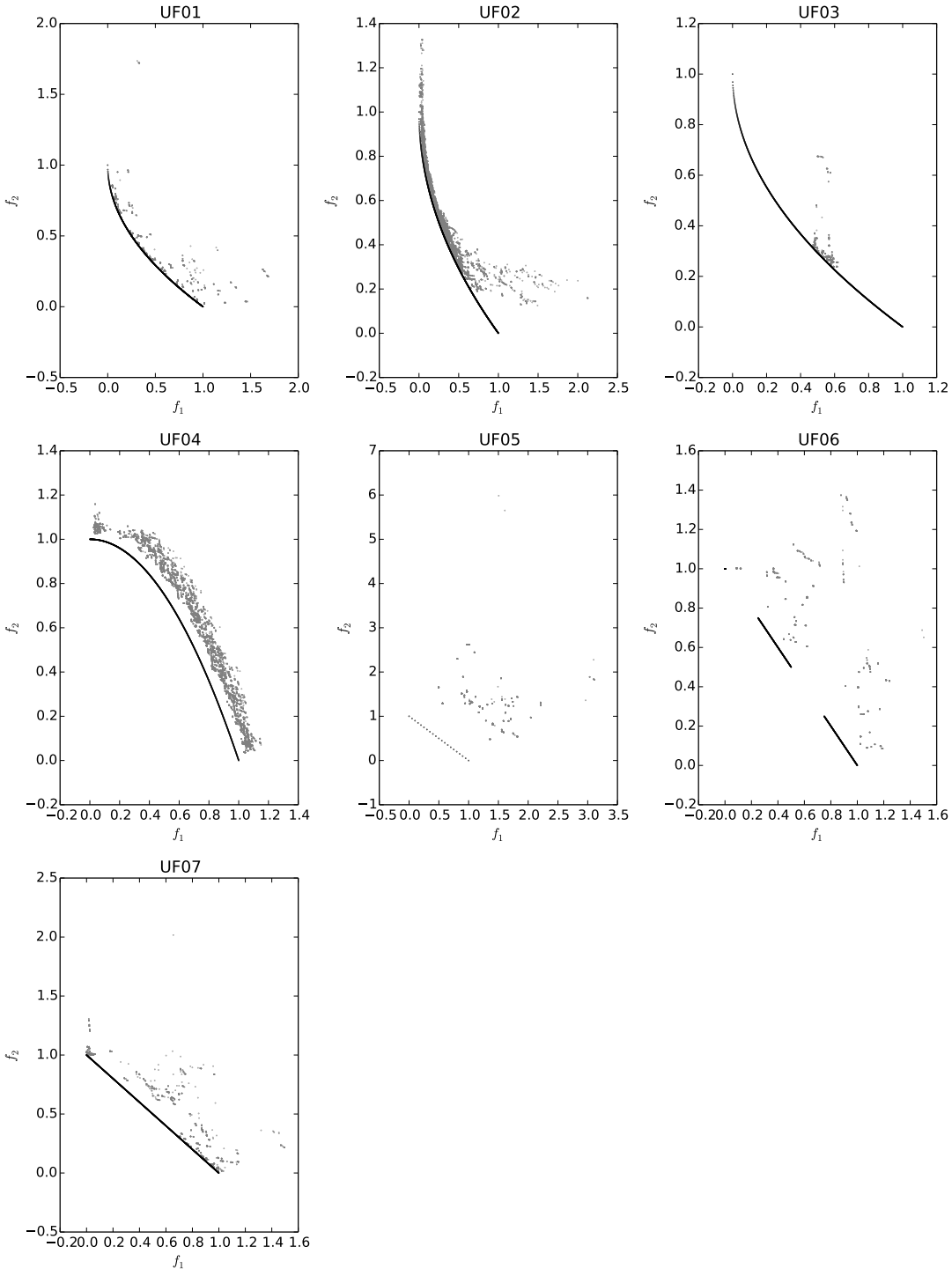


Figure A.11: PF approximations obtained by PSO variant using PROB leader selection scheme described by J. E. Alvarez-Benitez et al. (2005)

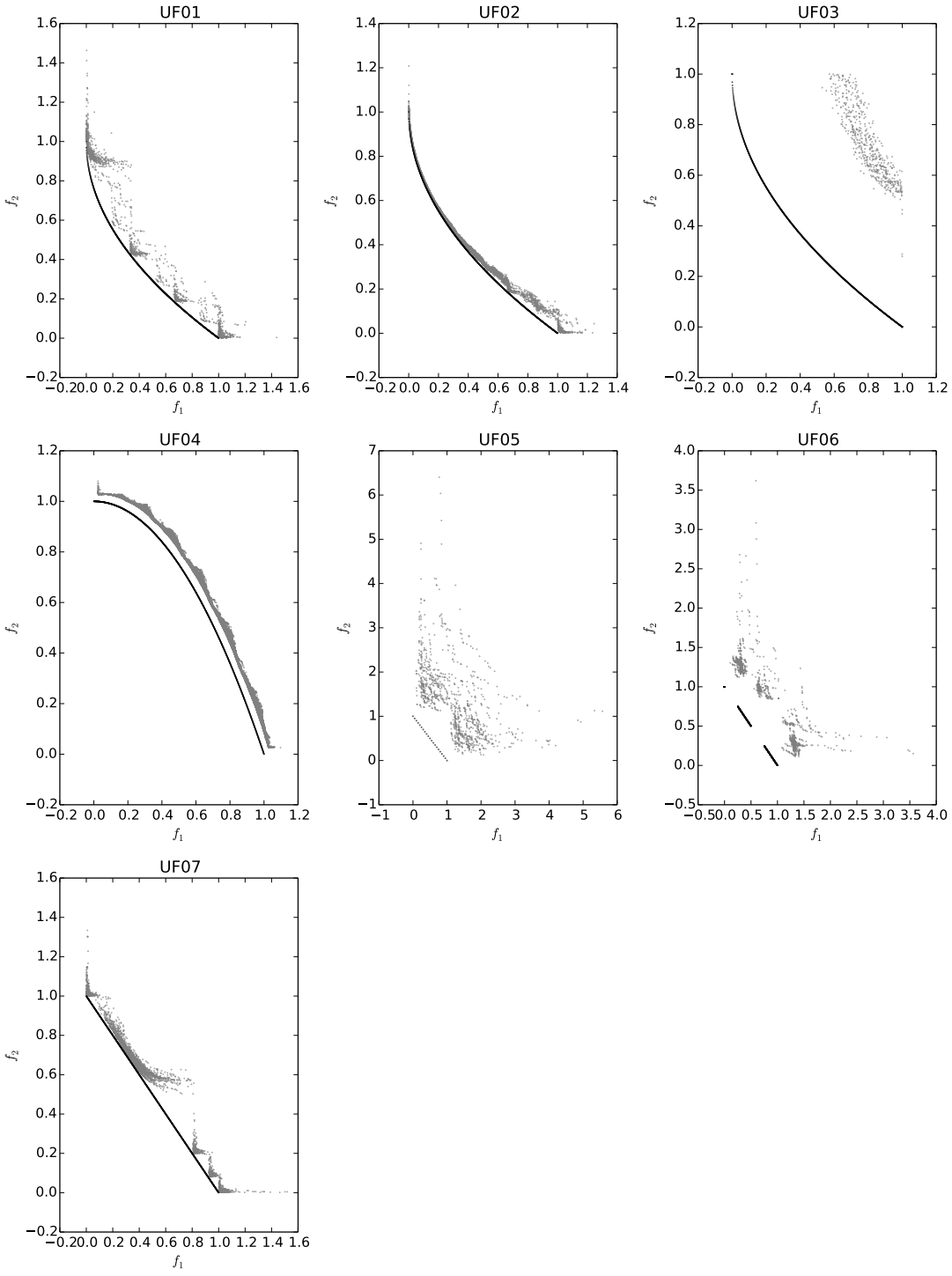


Figure A.12: PF approximations obtained by PSO variant described by M. R. Sierra et al. (2005)

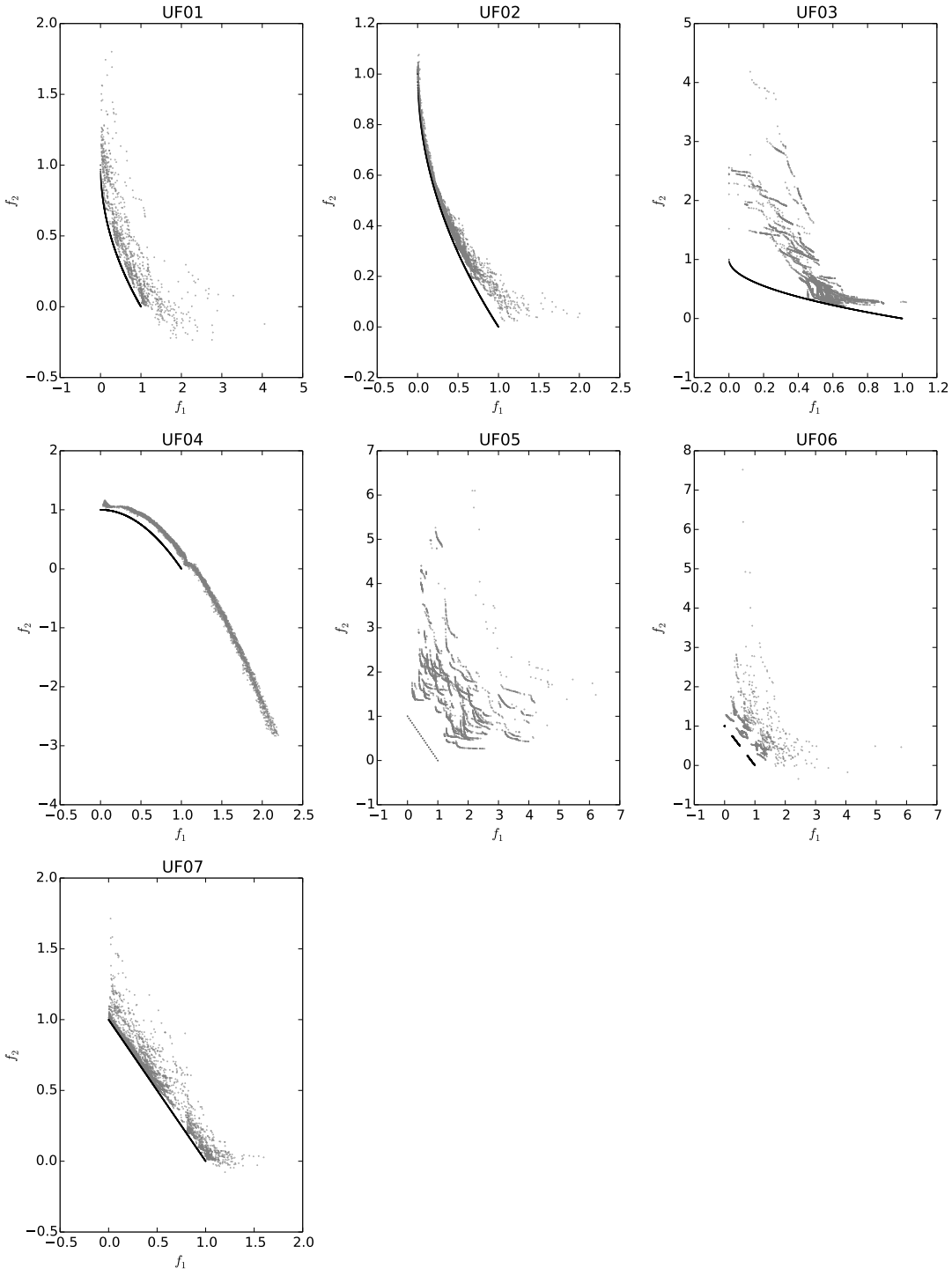


Figure A.13: PF approximations obtained by PSO variant described by P. K. Tripathi et al. (2007)

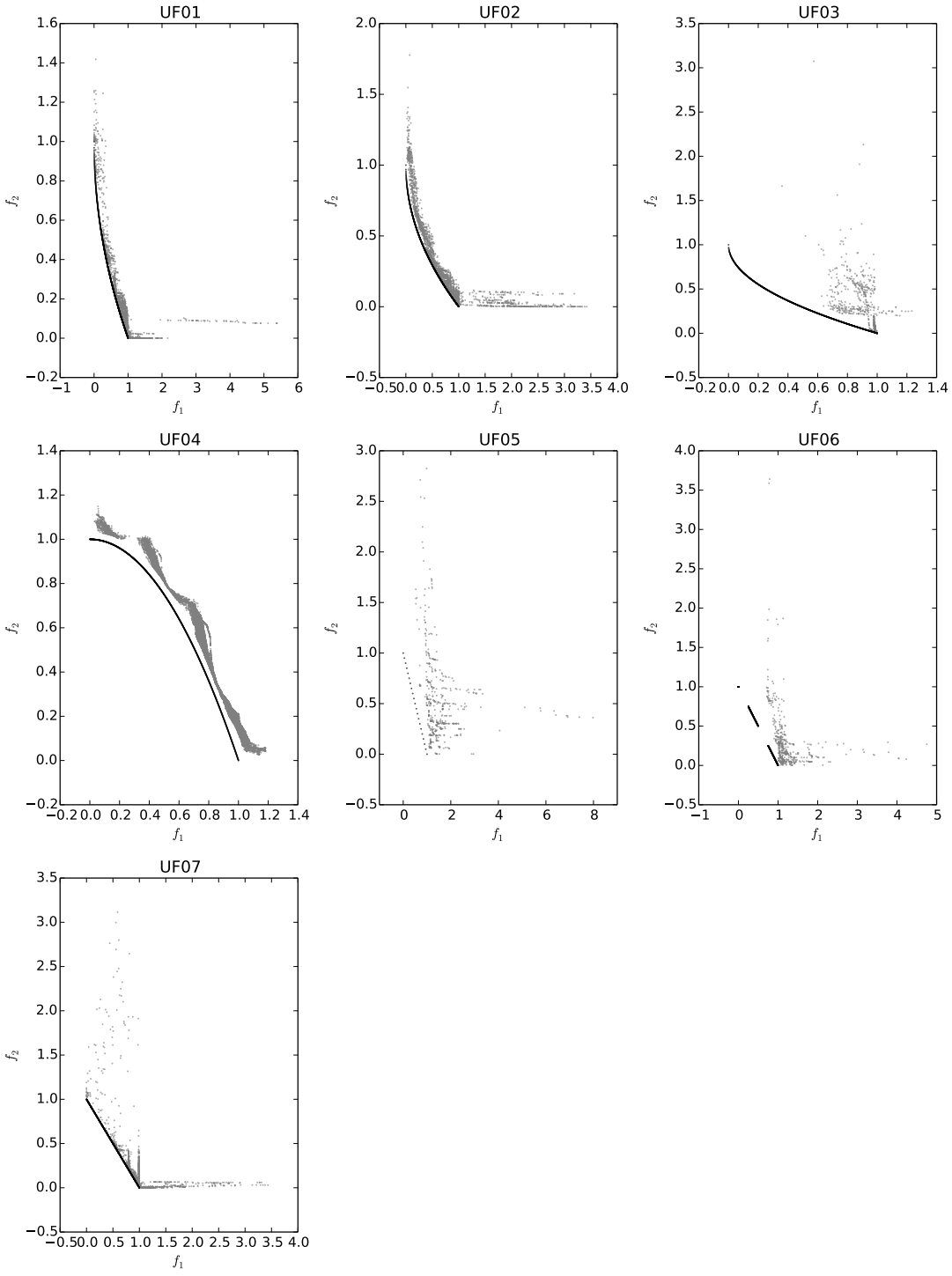


Figure A.14: PF approximations obtained by PSO variant described by W. Peng et al. (2008)

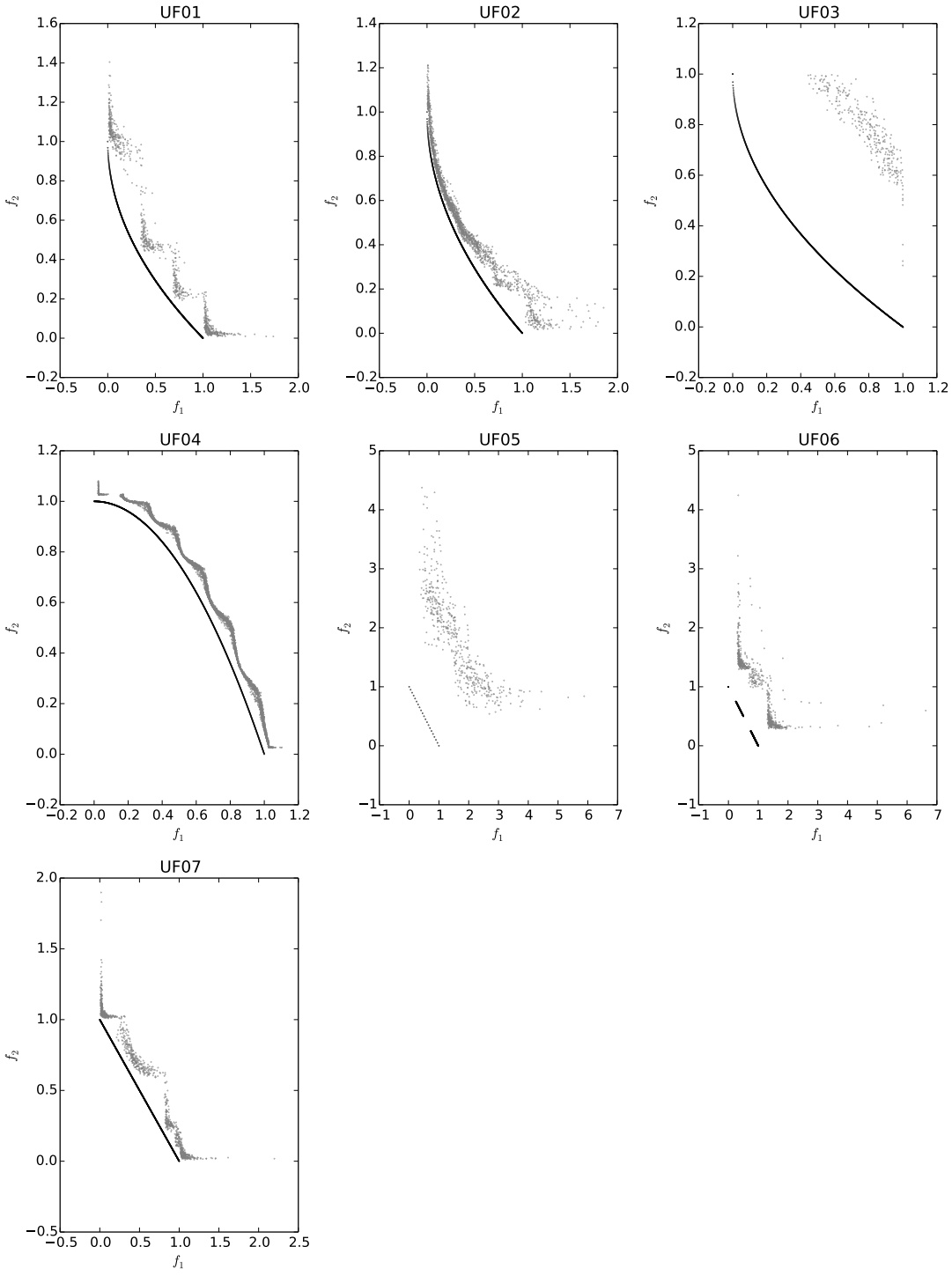


Figure A.15: PF approximations obtained by PSO variant described by A. J. Nebro et al. (2009)

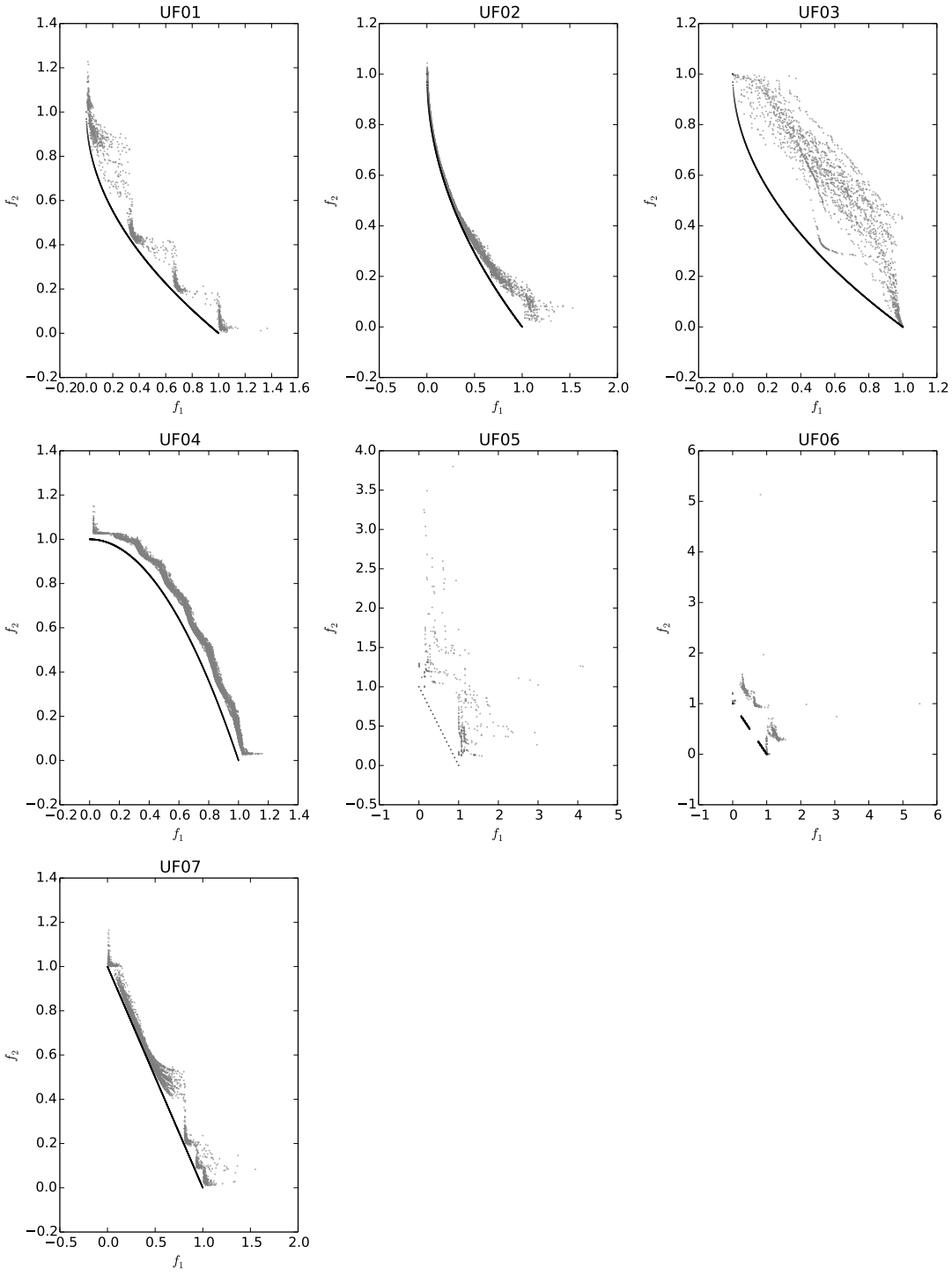


Figure A.16: PF approximations obtained by PSO variant described by N. Al Moubayed et al. (2010)

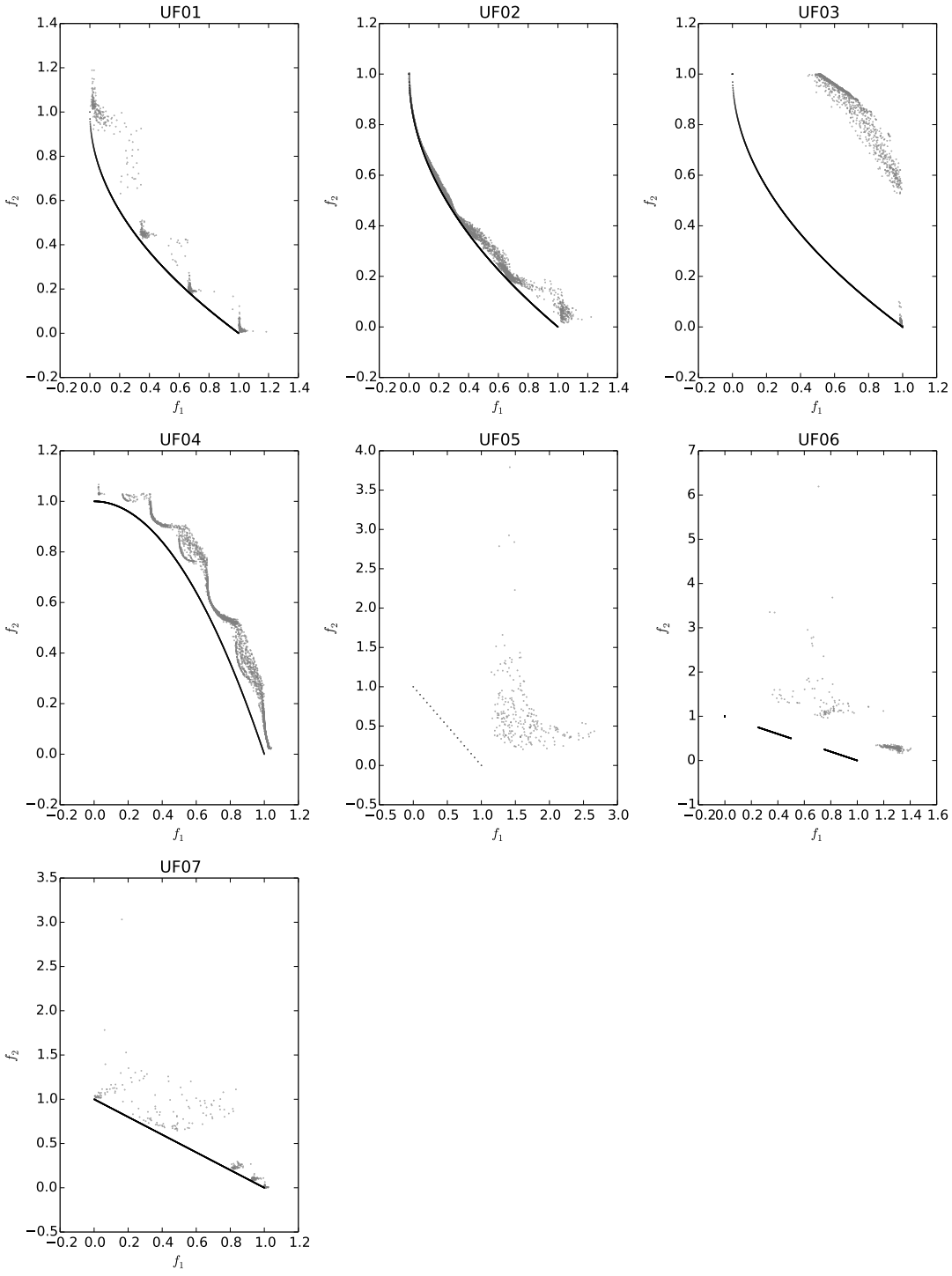


Figure A.17: PF approximations obtained by PSO variant described by S. Z. Martinez et al. (2011)

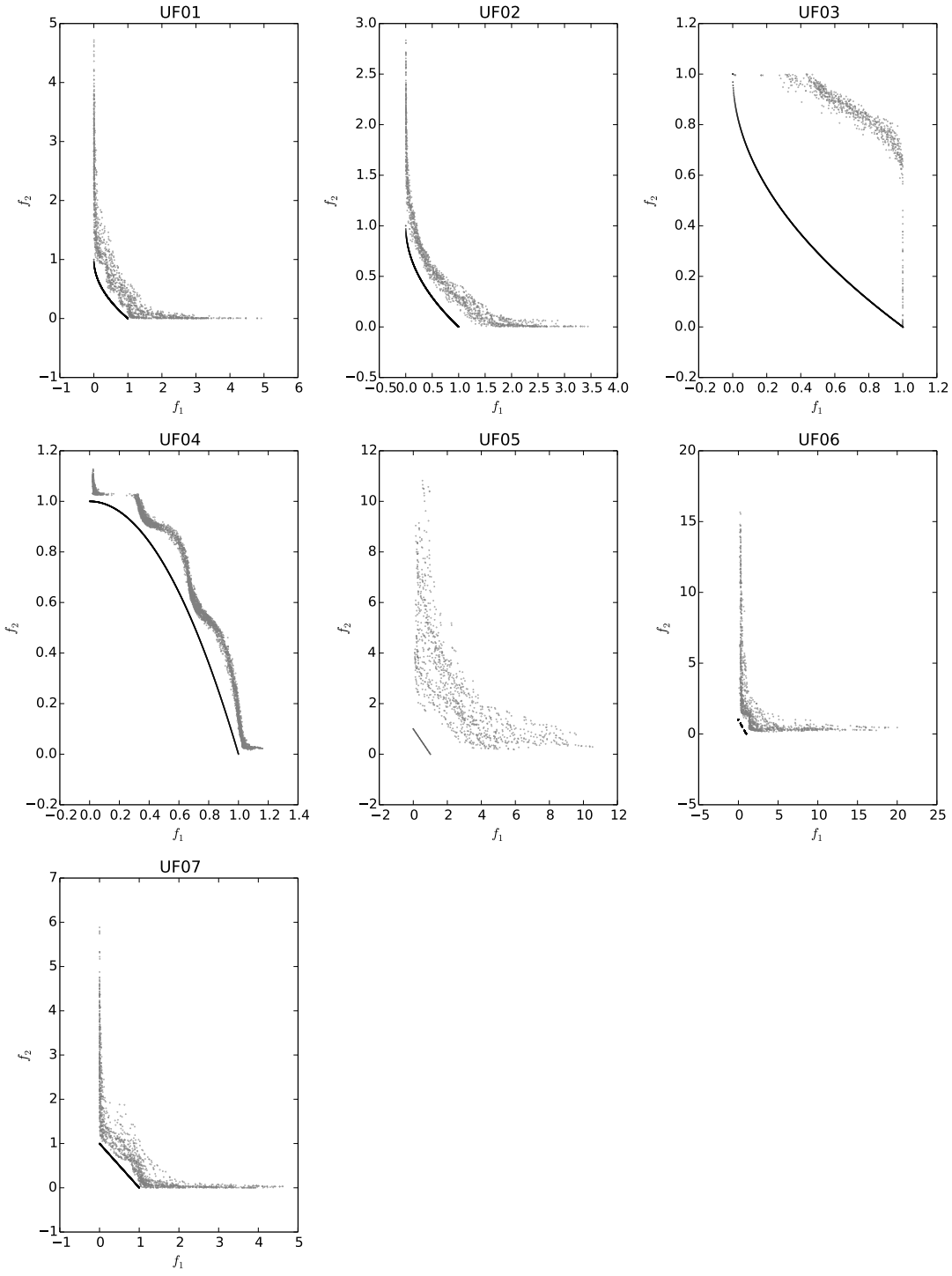


Figure A.18: PF approximations obtained by PSO variant described by K. S. Lim et al. (2013)

Appendix B

Result Tables

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.54265	0.47095	523.43333	0.00347
UF02	0.08409	0.11031	998.63333	0.00137
UF03	0.75389	0.56439	1002.10000	0.00228
UF04	0.10240	0.09460	974.26667	0.00215
UF05	3.51676	3.20375	793.73333	0.00383
UF06	3.05551	2.53389	770.00000	0.01270
UF07	0.63042	0.56749	555.33333	0.00328
UF08	0.57268	0.51948	4457.23333	0.01107
UF09	0.61015	0.57310	4753.63333	0.00842
UF10	4.00434	3.74844	3565.53333	0.05063
UF11	2.89858	2.10509	4996.46667	0.17478
UF12	3638.42544	1815.77537	3544.73333	265.98619
UF13	2.21119	2.09877	5000.00000	0.05546

Table B.1: Results for the C. A. C. Coello et al. (2002) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.03032	0.11958	72.26667	0.03919
UF02	0.00820	0.04157	498.56667	0.00672
UF03	0.17101	0.35715	17.60000	0.17773
UF04	0.04868	0.04792	1435.86667	0.00206
UF05	0.54354	0.77018	13.20000	0.35785
UF06	0.21924	0.37175	19.63333	0.08158
UF07	0.06631	0.14703	65.86667	0.04640

Table B.2: Results for the X. Hu et al. (2002) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	1.27435	0.82493	133.46667	0.08226
UF02	0.33057	0.24215	51.06667	0.06355
UF03	1.03112	0.65763	35.76667	0.09652
UF04	0.17425	0.17205	63.60000	0.01829
UF05	4.66808	3.61357	223.56667	0.13406
UF06	5.67040	3.27762	160.13333	0.18085
UF07	1.50044	0.86421	113.56667	0.07970

Table B.3: Results for the K. E. Parsopoulos et al. (2002) multi-objective particle swarm optimizer using Bang-Bang Weighted Aggregation.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.37434	0.32519	34.23333	0.12233
UF02	0.12079	0.11550	105.53333	0.03249
UF03	0.56132	0.48638	29.23333	0.21525
UF04	0.08047	0.07624	340.10000	0.00600
UF05	2.82009	2.29250	20.90000	0.32918
UF06	2.21840	1.45629	19.83333	0.55231
UF07	0.28109	0.27671	37.86667	0.10464

Table B.4: Results for the X. Hu et al. (2003) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.10770	0.11509	425.86667	0.00187
UF02	0.03667	0.05707	994.03333	0.00129
UF03	0.20510	0.24646	520.36667	0.00328
UF04	0.09494	0.08881	723.53333	0.00226
UF05	1.04895	1.00956	394.16667	0.00246
UF06	0.39866	0.49690	424.60000	0.00141
UF07	0.11199	0.17042	728.80000	0.00191
UF08	0.08903	0.24950	4872.53333	0.00771
UF09	0.39467	0.32150	4573.40000	0.01243
UF10	1.49229	1.36298	2105.30000	0.03957
UF11	1.68457	0.97422	5000.00000	0.14000
UF12	3498.94397	1735.62627	3610.46667	250.26862
UF13	2.19495	2.10249	5000.00000	0.05864

Table B.5: Results for the C. A. C. Coello et al. (2004) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.92593	0.38110	36.50000	0.17747
UF02	0.21998	0.17182	39.40000	0.09452
UF03	0.40522	0.34060	43.90000	0.06593
UF04	0.10182	0.09193	124.73333	0.00894
UF05	3.06011	2.10905	21.80000	0.28321
UF06	2.48982	1.12768	25.00000	0.33471
UF07	0.79351	0.33550	42.23333	0.12676
UF08	0.96576	0.50872	83.40000	0.38049
UF09	1.05419	0.48133	73.33333	0.47012
UF10	4.95058	3.52530	47.56667	1.16979
UF11	2.50684	1.17960	549.50000	0.42250
UF12	3415.47140	1511.15797	1602.60000	326.52693
UF13	2.09453	2.02508	5000.00000	0.05832

Table B.6: Results for the K. E. Parsopoulos et al. (2004) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.05722	0.12441	82.06667	0.02136
UF02	0.00636	0.04417	907.73333	0.00581
UF03	0.44608	0.36406	12.46667	0.24672
UF04	0.05725	0.05866	1579.40000	0.00945
UF05	1.29442	1.16855	39.76667	0.07703
UF06	0.50781	0.49462	71.60000	0.03672
UF07	0.03488	0.08136	134.20000	0.01255
UF08	0.00480	0.42761	4969.73333	0.00108
UF09	0.06632	0.25327	3356.13333	0.05422
UF10	0.72134	0.79535	1403.53333	0.07884
UF11	0.42172	0.22005	4987.53333	0.10142
UF12	2820.35112	1472.71590	1046.06667	329.28449
UF13	2.06287	2.04144	5000.00000	0.05754

Table B.7: Results for the Carlo R. Raquel et al. (2005) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.05257	0.16329	56.06667	0.01082
UF02	0.03149	0.11230	421.30000	0.01071
UF03	0.02214	0.31144	27.03333	0.00033
UF04	0.09722	0.09871	219.20000	0.00515
UF05	1.27642	1.33712	19.83333	0.03695
UF06	0.27121	0.55955	26.23333	0.00735
UF07	0.18447	0.31504	49.80000	0.02100
UF08	0.07777	0.36712	436.70000	0.02083
UF09	0.08742	0.45085	506.80000	0.05267
UF10	0.39628	0.74818	430.76667	0.03944
UF11	2.23943	0.67881	385.10000	0.20007
UF12	2565.55780	420.78130	900.30000	181.34900
UF13	2.26008	4.27774	250.00000	0.03033

Table B.8: Results for the Julio E. Alvarez-Benitez et al. (2005) multi-objective particle swarm optimizer using the RANDOM leader selection scheme.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.05907	0.17340	49.46667	0.00883
UF02	0.03498	0.10915	431.10000	0.00900
UF03	0.02355	0.31195	28.96667	0.00101
UF04	0.09562	0.09738	239.63333	0.00470
UF05	1.20888	1.29193	26.46667	0.04118
UF06	0.25690	0.59528	30.83333	0.01396
UF07	0.09127	0.31757	46.96667	0.03245
UF08	0.06508	0.36055	499.86667	0.02381
UF09	0.09380	0.44839	418.53333	0.05930
UF10	0.33794	0.69500	457.86667	0.02824
UF11	2.08516	0.68540	339.90000	0.21291
UF12	2636.68717	333.91581	867.76667	195.48762
UF13	2.26108	4.23445	250.00000	0.03401

Table B.9: Results for the Julio E. Alvarez-Benitez et al. (2005) multi-objective particle swarm optimizer using the PROB leader selection scheme.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.06905	0.10369	62.20000	0.02236
UF02	0.01651	0.02781	253.23333	0.00695
UF03	0.45414	0.35599	18.30000	0.18575
UF04	0.04252	0.04213	994.23333	0.00187
UF05	1.37283	1.11808	30.03333	0.09555
UF06	0.51504	0.48957	39.16667	0.06882
UF07	0.06128	0.07590	93.20000	0.01725
UF08	0.05181	0.17104	933.76667	0.05497
UF09	0.40111	0.24362	382.76667	0.11583
UF10	3.37871	2.03196	112.60000	0.46891
UF11	0.52629	0.28262	1000.00000	0.16452
UF12	2681.29049	1369.56726	999.40000	307.16064
UF13	2.14740	2.07414	1000.00000	0.14741

Table B.10: Results for the Margarita R. Sierra et al. (2005) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.09486	0.12791	29.73333	0.03418
UF02	0.05126	0.07187	103.40000	0.02068
UF03	0.43456	0.34047	28.10000	0.11433
UF04	0.05167	0.04965	215.96667	0.00544
UF05	1.49339	1.28964	18.16667	0.15116
UF06	0.61206	0.55856	23.00000	0.06948
UF07	0.08260	0.10302	39.40000	0.03066
UF08	0.10189	0.23403	264.16667	0.07848
UF09	1.23274	0.41769	159.06667	0.24267
UF10	3.53361	2.66109	51.13333	0.60033
UF11	0.91032	0.48461	882.83333	0.19739
UF12	2796.82429	1406.97118	1936.46667	274.22997
UF13	2.04956	2.04030	5000.00000	0.05768

Table B.11: Results for the Praveen Kumar Tripath et al. (2007) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.08110	0.08006	161.83333	0.02332
UF02	0.13009	0.06133	155.33333	0.02306
UF03	0.12393	0.39260	53.23333	0.06134
UF04	0.05960	0.06247	1528.86667	0.00211
UF05	0.67116	0.79115	18.53333	0.15540
UF06	0.39718	0.64558	17.73333	0.18418
UF07	0.18239	0.36804	70.56667	0.13459
UF08	0.94433	0.21901	432.33333	0.15521
UF09	0.74370	0.35457	760.90000	0.12309
UF10	4.50931	1.89385	91.30000	0.85921
UF11	0.77264	0.56398	2663.80000	0.19917
UF12	2931.64161	1084.20675	1960.56667	298.40098
UF13	2.18136	2.07135	5000.00000	0.06416

Table B.12: Results for the W. Peng et al. (2008) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.12366	0.13942	28.53333	0.04042
UF02	0.08122	0.09043	63.03333	0.02855
UF03	0.44927	0.36511	12.46667	0.18939
UF04	0.05196	0.05074	212.96667	0.00553
UF05	1.77617	1.48813	16.23333	0.18964
UF06	0.76306	0.63015	21.76667	0.16551
UF07	0.11835	0.12402	33.60000	0.04958
UF08	0.35906	0.27913	105.63333	0.21601
UF09	1.13376	0.48351	87.90000	0.33498
UF10	5.00787	3.36219	47.80000	0.97501
UF11	1.30740	0.65470	517.86667	0.26796
UF12	2918.02658	1403.51672	2129.46667	273.86523
UF13	2.05146	2.03887	5000.00000	0.05808

Table B.13: Results for the A. J. Nebro et al. (2009) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.05751	0.08950	56.70000	0.02118
UF02	0.02130	0.04242	300.26667	0.00808
UF03	0.19017	0.20413	78.06667	0.03031
UF04	0.06093	0.05636	271.33333	0.00546
UF05	0.43615	0.68731	20.80000	0.09149
UF06	0.22401	0.58481	39.66667	0.06967
UF07	0.03933	0.05251	153.63333	0.01300
UF08	0.02279	0.24901	1917.63333	0.01350
UF09	0.05366	0.29724	2486.53333	0.03126
UF10	2.24683	1.74295	69.03333	0.27392
UF11	0.25754	0.17251	4999.90000	0.07739
UF12	3188.04526	1332.33054	2275.66667	287.80740
UF13	2.04441	2.03943	5000.00000	0.05893

Table B.14: Results for the N. Al Moubayed et al. (2010) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.04508	0.10965	27.70000	0.03882
UF02	0.00852	0.03725	534.83333	0.00675
UF03	0.23912	0.35778	90.10000	0.05544
UF04	0.05719	0.07289	215.06667	0.01517
UF05	0.89289	0.94541	8.36667	0.16973
UF06	0.52364	0.60466	13.43333	0.24930
UF07	0.07885	0.19819	19.03333	0.14440
UF08	0.08824	0.36573	85.13333	0.17507
UF09	0.00951	0.29282	2869.63333	0.01374
UF10	0.14417	0.93789	10.36667	0.56399
UF11	0.39910	0.87097	3862.00000	0.18241
UF12	3010.71854	1246.67380	2121.36667	292.57181
UF13	2.14701	2.05360	5000.00000	0.05950

Table B.15: Results for the S. Z. Martinez et al. (2011) multi-objective particle swarm optimizer.

Problem	$\mu(I_{GD})$	$\mu(I_{IGD})$	$\mu(I_N)$	$\mu(I_{SP})$
UF01	0.90021	0.30685	53.26667	0.05998
UF02	0.54484	0.20606	53.20000	0.04596
UF03	0.38168	0.35229	35.30000	0.06430
UF04	0.07972	0.08039	145.16667	0.00858
UF05	3.82114	2.30341	35.60000	0.18686
UF06	4.18194	1.13630	41.50000	0.26138
UF07	1.00934	0.33026	52.96667	0.06678
UF08	4.11143	1.17021	241.60000	0.34851
UF09	5.76503	1.22421	261.40000	0.39591
UF10	19.64855	7.26586	176.70000	1.67115
UF11	3.11788	1.70583	876.93333	0.47266
UF12	2950.12020	1000.66883	1105.96667	309.88862
UF13	2.15717	2.04876	5000.00000	0.05892

Table B.16: Results for the K. S. Lim et al. (2013) multi-objective particle swarm optimizer.

Bibliography

[1] Noura Al Moubayed, Andrei Petrovski, and John McCall. A novel smart multi-objective particle swarm optimisation using decomposition. In *Parallel Problem Solving from Nature, PPSN XI*, pages 1–10. Springer, 2010.

[2] Julio E Alvarez-Benitez, Richard M Everson, and Jonathan E Fieldsend. A mopso algorithm based exclusively on pareto dominance concepts. In *Evolutionary Multi-Criterion Optimization*, pages 459–473. Springer, 2005.

[3] Paul S Andrews. An investigation into mutation operators for particle swarm optimization. In *Evolutionary Computation, 2006. CEC 2006. IEEE Congress on*, pages 1044–1051. IEEE, 2006.

[4] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *Evolutionary Computation, IEEE Transactions on*, 6(1):58–73, 2002. ISSN 1089-778X. doi: 10.1109/4235.985692.

[5] Carlos A Coello Coello, Gregorio Toscano Pulido, and M Salazar Lechuga. Handling multiple objectives with particle swarm optimization. *Evolutionary Computation, IEEE Transactions on*, 8(3):256–279, 2004.

[6] Carlos A Coello Coello and Maximino Salazar Lechuga. Mopso: A proposal for multiple objective particle swarm optimization. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, volume 2, pages 1051–1056. IEEE, 2002.

[7] Jared L Cohon. Multicriteria programming: Brief review and application. *Design optimization*, pages 163–191, 1985.

[8] Jared L Cohon. *Multiobjective programming and planning*. Courier Corporation, 2013.

[9] Indraneel Das. A preference ordering among various pareto optimal alternatives. *Structural optimization*, 18(1):30–35, 1999.

[10] Marco Antonio Montes de Oca, Jorge Peña, Thomas Stutzle, Carlo Pinciroli,

and Marco Dorigo. Heterogeneous particle swarm optimizers. In *Evolutionary Computation, 2009. CEC'09. IEEE Congress on*, pages 698–705. IEEE, 2009.

[11] Kalyanmoy Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[12] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, 2002.

[13] Francesco di Pierro, Soon-Thiam Khu, and Dragan A Savic. An investigation on preference order ranking scheme for multiobjective evolutionary optimization. *Evolutionary Computation, IEEE Transactions on*, 11(1):17–45, 2007.

[14] Russ C Eberhart and Yuhui Shi. Comparing inertia weights and constriction factors in particle swarm optimization. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 84–88. IEEE, 2000.

[15] Ahmed Elhossini, Shawki Areibi, and Robert Dony. Strength pareto particle swarm optimization and hybrid ea-pso for multi-objective optimization. *Evolutionary Computation*, 18(1):127–156, 2010.

[16] Andries P Engelbrecht. Heterogeneous particle swarm optimization. In *Swarm Intelligence*, pages 191–202. Springer, 2010.

[17] Susana C Esquivel and CA Coello Coello. On the use of particle swarm optimization with multimodal functions. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 1130–1136. IEEE, 2003.

[18] Jonathan E Fieldsend and Sameer Singh. A multi-objective algorithm based upon particle swarm optimisation, an efficient data structure and turbulence. 2002.

[19] Ivan Chaman Garcia, Carlos A Coello Coello, and Alfredo Arias-Montano. Mopsohv: A new hypervolume-based multi-objective particle swarm optimizer. In *Evolutionary Computation (CEC), 2014 IEEE Congress on*, pages 266–273. IEEE, 2014.

[20] F. Heppner and U. Grenander. A stochastic nonlinear model for coordinated bird flocks. In E. Krasner, editor, *The ubiquity of chaos*, pages 233–238. AAAS Publications, 1990.

[21] Natsuki Higashi and Hitoshi Iba. Particle swarm optimization with gaussian mutation. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of*

the 2003 IEEE, pages 72–79. IEEE, 2003.

[22] Xiaohui Hu and Russell Eberhart. Multiobjective optimization using dynamic neighborhood particle swarm optimization. In *Proceedings of the Evolutionary Computation on 2002. CEC '02. Proceedings of the 2002 Congress - Volume 02*, CEC '02, pages 1677–1681, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7803-7282-4. URL `http://dl.acm.org/citation.cfm?id=1251972.1252305`.

[23] Xiaohui Hu, Russell C Eberhart, and Yuhui Shi. Particle swarm with extended memory for multiobjective optimization. In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 193–197. IEEE, 2003.

[24] VL Huang, AK Qin, K Deb, E Zitzler, PN Suganthan, JJ Liang, M Preuss, and S Huband. Problem definitions for performance assessment of multiobjective optimization algorithms. *Nanyang Technological University, Singapore, Tech. Rep*, 2007.

[25] Simon Huband, Philip Hingston, Luigi Barone, and Lyndon While. A review of multiobjective test problems and a scalable test problem toolkit. *Evolutionary Computation, IEEE Transactions on*, 10(5):477–506, 2006.

[26] Rui Mendes James Kennedy. Population structure and particle swarm performance. *Proceedings of the 2002 Congress on Evolutionary Computation*, 2:1671 – 1676, 2002.

[27] Russell C. Eberhart James Kennedy. Particle swarm optimization. *IEEE International Conference on Neural Networks, Proceedings*, 4:1942 – 1948, 1995.

[28] J. Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3, pages –1938 Vol. 3, 1999. doi: 10.1109/CEC.1999.785509.

[29] J. Kennedy. Bare bones particle swarms. In *Swarm Intelligence Symposium, 2003. SIS '03. Proceedings of the 2003 IEEE*, pages 80–87, 2003. doi: 10.1109/SIS.2003.1202251.

[30] James Kennedy. Small worlds and mega-minds: effects of neighborhood topology on particle swarm performance. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.

[31] James Kennedy and Rui Mendes. Population structure and particle swarm performance. In *Evolutionary Computation, 2002. CEC'02. Proceedings of the*

2002 Congress on, volume 2, pages 1671–1676. IEEE, 2002.

[32] Juhani Koski. Multicriterion optimization in structural design. Technical report, DTIC Document, 1981.

[33] Marco Laumanns, Lothar Thiele, Kalyanmoy Deb, and Eckart Zitzler. Combining convergence and diversity in evolutionary multiobjective optimization. *Evolutionary computation*, 10(3):263–282, 2002.

[34] Hui Li and Qingfu Zhang. Comparison between nsga-ii and moea/d on a set of multiobjective problems with complicated pareto sets. *IEEE Trans. on Evolutionary Computation*, 2008.

[35] Hui Li and Qingfu Zhang. Multiobjective optimization problems with complicated pareto sets, moea/d and nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 13(2):284–302, 2009.

[36] Xiaodong Li. A non-dominated sorting particle swarm optimizer for multiobjective optimization. In *Genetic and Evolutionary ComputationâĂŤGECCO 2003*, pages 37–48. Springer, 2003.

[37] Kian Sheng Lim, Zuwairie Ibrahim, Salinda Buyamin, Anita Ahmad, Faradila Naim, Kamarul Hawari Ghazali, and Norrima Mokhtar. Improving vector evaluated particle swarm optimisation by incorporating nondominated solutions. *The Scientific World Journal*, 2013, 2013.

[38] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: simpler, maybe better. *Evolutionary Computation, IEEE Transactions on*, 8(3): 204–210, 2004. ISSN 1089-778X. doi: 10.1109/TEVC.2004.826074.

[39] Zbigniew Michalewicz. *Genetic algorithms+ data structures= evolution programs*. springer, 1996.

[40] Sanaz Mostaghim and Jürgen Teich. Strategies for finding good local guides in multi-objective particle swarm optimization (mopso). In *Swarm Intelligence Symposium, 2003. SIS'03. Proceedings of the 2003 IEEE*, pages 26–33. IEEE, 2003.

[41] Sanaz Mostaghim and Jürgen Teich. Covering pareto-optimal fronts by subswarms in multi-objective particle swarm optimization. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1404–1411. IEEE, 2004.

[42] Antonio J Nebro, JJ Durillo, Jose Garcia-Nieto, CA Coello Coello, Francisco Luna, and Enrique Alba. 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, 2009.

[43] Antonio J Nebro, Juan J Durillo, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. Mocell: A cellular genetic algorithm for multiobjective optimization. *International Journal of Intelligent Systems*, 24(7):726–746, 2009.

[44] Antonio J Nebro, Juan José Durillo, and Carlos Artemio Coello Coello. Analysis of leader selection strategies in a multi-objective particle swarm optimizer. In *Evolutionary Computation (CEC), 2013 IEEE Congress on*, pages 3153–3160. IEEE, 2013.

[45] Ender Ozcan and Chilukuri K Mohan. Analysis of a simple particle swarm optimization system. *Intelligent engineering systems through artificial neural networks*, 8:253–258, 1998.

[46] Ender Ozcan and Chilukuri K Mohan. Particle swarm optimization: surfing the waves. In *Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on*, volume 3. IEEE, 1999.

[47] K. E. Parsopoulos and M. N. Vrahatis. Particle swarm optimization method in multiobjective problems. In *Proceedings of the 2002 ACM symposium on Applied computing*, SAC '02, pages 603–607, New York, NY, USA, 2002. ACM. ISBN 1-58113-445-2. doi: 10.1145/508791.508907. URL `http://doi.acm.org/10.1145/508791.508907`.

[48] Konstantinos E Parsopoulos, Dimitris K Tasoulis, Michael N Vrahatis, et al. Multiobjective optimization using parallel vector evaluated particle swarm optimization. In *Proceedings of the IASTED international conference on artificial intelligence and applications (AIA 2004)*, volume 2, pages 823–828, 2004.

[49] Wei Peng and Qingfu Zhang. A decomposition-based multi-objective particle swarm optimization algorithm for continuous optimization problems. In *Granular Computing, 2008. GrC 2008. IEEE International Conference on*, pages 534–537. IEEE, 2008.

[50] Riccardo Poli and William B Langdon. Markov chain models of bare-bones particle swarm optimizers. In *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 142–149. ACM, 2007.

[51] Riccardo Poli, James Kennedy, and Tim Blackwell. Particle swarm optimization. *Swarm Intelligence*, 1(1):33–57, 2007. URL `http://www.springerlink.com/index/10.1007/s11721-007-0002-0`.

[52] Carlo R Raquel and Prospero C Naval Jr. An effective use of crowding distance in multiobjective particle swarm optimization. In *Proceedings of*

the 2005 conference on Genetic and evolutionary computation, pages 257–264. ACM, 2005.

[53] Margarita Reyes-Sierra and CA Coello Coello. Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International journal of computational intelligence research*, 2(3):287–308, 2006.

[54] Craig W Reynolds. Flocks, herds and schools: A distributed behavioral model. In *ACM Siggraph Computer Graphics*, volume 21, pages 25–34. ACM, 1987.

[55] M. Salazar-Lechuga and J.E. Rowe. Particle swarm optimization and fitness sharing to solve multi-objective optimization problems. In *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, volume 2, pages 1204–1211 Vol. 2, Sept 2005. doi: 10.1109/CEC.2005.1554827.

[56] J. David Schaffer. Multiple objective optimization with vector evaluated genetic algorithms. In *Proceedings of the 1st International Conference on Genetic Algorithms*, pages 93–100, Hillsdale, NJ, USA, 1985. L. Erlbaum Associates Inc. ISBN 0-8058-0426-9. URL `http://dl.acm.org/citation.cfm?id=645511.657079`.

[57] Jason R Schott. Fault tolerant design using single and multicriteria genetic algorithm optimization. Technical report, DTIC Document, 1995.

[58] O Schutze, Xavier Esquivel, Adriana Lara, and Carlos A Coello Coello. Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization. *Evolutionary Computation, IEEE Transactions on*, 16(4):504–522, 2012.

[59] Yuhui Shi and Russell C Eberhart. Parameter selection in particle swarm optimization. In *Evolutionary Programming VII*, pages 591–600. Springer, 1998.

[60] Margarita Reyes Sierra and Carlos A Coello Coello. Improving pso-based multi-objective optimization using crowding, mutation and epsilon-dominance. In *Evolutionary multi-criterion optimization*, pages 505–519. Springer, 2005.

[61] Andrew Stacey, Mirjana Jancic, and Ian Grundy. Particle swarm optimization with mutation. In *Evolutionary Computation, 2003. CEC'03. The 2003 Congress on*, volume 2, pages 1425–1430. IEEE, 2003.

[62] Ralph E Steuer. *Multiple criteria optimization: theory, computation, and applications*. Wiley, 1986.

[63] Praveen Kumar Tripathi, Sanghamitra Bandyopadhyay, and Sankar Kumar Pal. Multi-objective particle swarm optimization with time variant inertia and acceleration coefficients. *Information Sciences*, 177(22):5033–5049, 2007.

[64] David A Van Veldhuizen and Gary B Lamont. On measuring multiobjective evolutionary algorithm performance. In *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 204–211. IEEE, 2000.

[65] Jakob Vesterstrom and Rene Thomsen. A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems. In *Evolutionary Computation, 2004. CEC2004. Congress on*, volume 2, pages 1980–1987. IEEE, 2004.

[66] Yujia Wang and Yupu Yang. Particle swarm optimization with preference order ranking for multi-objective optimization. *Information Sciences*, 179 (12):1944–1959, 2009.

[67] Upali Wickramasinghe and Xiaodong Li. Choosing leaders for multiobjective pso algorithms using differential evolution. In *Simulated Evolution and Learning*, pages 249–258. Springer, 2008.

[68] Jin Wu and Shapour Azarm. Metrics for quality assessment of a multiobjective design optimization solution set. *Journal of Mechanical Design*, 123 (1):18–25, 2001.

[69] Saúl Zapotecas Martínez and Carlos A Coello Coello. A multi-objective particle swarm optimizer based on decomposition. In *Proceedings of the 13th annual conference on Genetic and evolutionary computation*, pages 69–76. ACM, 2011.

[70] Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *Evolutionary Computation, IEEE Transactions on*, 11(6):712–731, 2007.

[71] Qingfu Zhang, Aimin Zhou, Shizheng Zhao, Ponnuthurai Nagaratnam Suganthan, Wudong Liu, and Santosh Tiwari. 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*, 2008.

[72] Eckart Zitzler and Lothar Thiele. Multiobjective optimization using evolutionary algorithmsâĂŤa comparative case study. In *Parallel problem solving from natureâĂŤPPSN V*, pages 292–301. Springer, 1998.

[73] Eckart Zitzler, Marco Laumanns, Lothar Thiele, Eckart Zitzler, Eckart Zitzler, Lothar Thiele, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm, 2001.

[74] Eckart Zitzler, Lothar Thiele, Marco Laumanns, Carlos M Fonseca, and Viviane Grunert Da Fonseca. Performance assessment of multiobjective optimizers: An analysis and review. *Evolutionary Computation, IEEE Transactions on*, 7(2):117–132, 2003.

Vytautas Jančauskas

EVALUATING THE PERFORMANCE OF MULTI-OBJECTIVE PARTICLE SWARM OPTIMIZATION ALGORITHMS

Doctoral Dissertation

Physical Sciences (P 000)
Informatics (09 P)

Editor Vilija Ambrasienė