BRNO UNIVERSITY OF TECHNOLOGY
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF RADIOENGINEERING
ÚSTAV RADIOELEKTRONIKY

MULTI-OBJECTIVE OPTIMIZATION OF EM STRUCTURES WITH VARIABLE NUMBER OF DIMENSIONS
VÍCE-KRITERIÁLNÍ OPTIMALIZACE EM STRUKTUR S PROMĚNNÝM POČTEM DIMENZÍ

DOCTORAL THESIS
DIZERTAČNÍ PRÁCE

AUTHOR — **Ing. Martin Marek**
AUTOR PRÁCE

SUPERVISOR — **Ing. Petr Kadlec, Ph.D.**
ŠKOLITEL

BRNO 2020

ABSTRACT

This dissertation thesis deals with multi-objective evolutionary optimization algorithms with a variable number of dimensions. Such an algorithm enables us to solve optimization tasks that are otherwise solved only by assuming unnatural simplifications. The research of the optimization algorithms with a variable number of dimensions required the development of a new optimization framework. This framework contains, apart from various optimization methods including two novel multi-objective algorithms for a variable number of dimensions – VND-GDE3 and VND-MOPSO, a library of various benchmark problems. A set of multi-objective benchmark problems with a variable number of dimensions is a part of the library designed to assess and verify the novel methods with a variable number of dimensions. Novel methods are exploited on several miscellaneous real-life optimization problems in the final chapter of this thesis.

KEYWORDS

Evolutionary algorithms, EM structures design, FOPS, multi-objective optimization, variable number of dimensions, VND-GDE3, VND-MOPSO.

ABSTRAKT

Tato dizertační práce pojednává o více-kriteriálních optimalizačních algoritmech s proměnným počtem dimenzí. Takový algoritmus umožňuje řešit optimalizační úlohy, které jsou jinak řešitelné jen s použitím nepřirozených zjednodušení. Výzkum optimalizačních method s proměnnou dimenzí si vyžádal vytvoření nového optimalizačního frameworku, který obsahuje vedle zmíněných vícekriteriálních metod s proměnnou dimenzí – VND-GDE3 a VND-MOPSO – i další optimalizační metody různých tříd. Optimalizační framework obsahuje také knihovnu rozličných testovacích problémů. Mezi nimi je také sada více-kriteriálních testovacích problémů s proměnnou dimenzí, které byly navrženy pro nastavení a ověření nových metod s proměnnou dimenzí. Nové metody jsou dále použity k optimalizaci několika různorodých optimalizačních úloh z reálného světa.

KLÍČOVÁ SLOVA

Evoluční algoritmus, FOPS, návrh EM struktur, proměnný počet dimenzí, více-kriteriální optimalizace, VND-GE3, VND-MOPSO.

MAREK, Martin *Multi-objective Optimization of EM Structures With Variable Number of Dimensions*: doctoral thesis. Brno: Brno University of Technology, Faculty of Electrical Engineering and Communication, Ústav radioelektroniky, 2020. 153 p. Supervised by Ing. Petr Kadlec, Ph.D.

DECLARATION

I declare that I have written my doctoral thesis on the theme of "Multi-objective Optimization of EM Structures With Variable Number of Dimensions" independently, under the guidance of the doctoral thesis supervisor and using the technical literature and other sources of information which are all quoted in the thesis and detailed in the list of literature at the end of the thesis.

As the author of the doctoral thesis I furthermore declare that, as regards the creation of this doctoral thesis, I have not infringed any copyright. In particular, I have not unlawfully encroached on anyone's personal and/or ownership rights and I am fully aware of the consequences in the case of breaking Regulation § 11 and the following of the Copyright Act No 121/2000 Sb., and of the rights related to intellectual property right and changes in some Acts (Intellectual Property Act) and formulated in later regulations, inclusive of the possible consequences resulting from the provisions of Criminal Act No 40/2009 Sb., Section 2, Head VI, Part 4.

Brno ....

(author's signature)

ACKNOWLEDGEMENT

I would like to thank my supervisor Ing. Petr Kadlec Ph.D. for his support, motivation, and knowledge in the field every time it was needed.

Brno . . . .

(author's signature)

SIX research centre
sensor, information and communication systems

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czechia

http://www.six.feec.vutbr.cz

Výzkum popsaný v této diplomové práci byl realizovaný v laboratořích podpořených projektem Centrum senzorických, informačních a komunikačních systémů (SIX); registrační číslo CZ.1.05/2.1.00/03.0072, operačního programu Výzkum a vývoj pro inovace.

MINISTERSTVO ŠKOLSTVÍ, MLÁDEŽE A TĚLOVÝCHOVY

EVROPSKÁ UNIE
EVROPSKÝ FOND PRO REGIONÁLNÍ ROZVOJ
INVESTICE DO VAŠÍ BUDOUCNOSTI

2007–13
OP Výzkum a vývoj pro inovace

CONTENTS

1 Introduction **7**

2 Theoretical background **9**

2.1 Single-objective Optimization 9

2.2 Multi-objective Optimization 11

2.2.1 Principle of Dominance 11

2.2.2 Non-dominated Sorting 12

2.2.3 Maintenance of Diversity 12

2.3 Performance Metrics 14

2.3.1 Generational Distance 14

2.3.2 Spread 15

2.3.3 Hypervolume 15

2.3.4 Number of Variables Deviation 16

2.4 Non-parametric Statistical Tests 16

2.4.1 Wilcoxon's Signed Ranks Test 16

2.4.2 Friedman's Test 17

3 Survey of the previous work **19**

3.1 As the Time Went By 19

3.2 Particle Swarm Optimization for Variable Number of Dimensions 22

3.2.1 Position Update 23

3.2.2 Dimension Check 23

3.3 Motivation 24

3.4 Dissertation Objectives 25

4 FOPS **27**

4.1 Simulations in FOPS 28

4.1.1 Simulation Types 30

4.1.2 Optimization Execution 31

4.2 Visualisation of Results 31

4.2.1 Results of Tasks and Chains 32

4.2.2 Results of Comparative Study 32

4.2.3 Statistics 33

4.3 Discretization Method 33

4.3.1 Discretization Method 35

4.3.2 Test Problems 35

4.3.3 Results 36

4.3.4 Conclusion 37

4.4 Surrogate Optimization 37

4.4.1 Performance Assessment 40

4.4.2 Results . . . 41
4.4.3 Conclusion . . . 44
4.5 Conclusion . . . 45

5 Multi-objective Testing Problems with Variable Number of Dimensions 47
5.1 Sample Problem – VND-MOZDT1 . . . 48
5.2 Sample Problem – VND-DTLZ2 . . . 48

6 VND-GDE3 51
6.1 GDE3 . . . 51
6.2 VND-GDE3 . . . 52
6.3 VLGDE3 . . . 53

7 VND-MOPSO 54
7.1 MOPSO . . . 54
7.2 VND-MOPSO . . . 55
7.3 VLMOPSO . . . 55

8 Performance Assessment 57
8.1 Influence of Probability of Dimension Transition Parameter of VND-GDE3 57
8.2 Influence of Probabilities to Follow Parameter of VND-MOPSO . . . 60
8.3 Comparison of Clustered, Pure-VND, and Impure-VND Approaches . . . 63

9 Applications 71
9.1 Design of an Anisotropic Band-Stop Filter . . . 73
9.1.1 Optimization Parameters . . . 73
9.2 Hybrid Optimization . . . 77
9.3 Optimal Placement of Transmitters . . . 79
9.3.1 Linear Antenna Array Design . . . 83
9.4 Synthesis of the Digital Circuits . . . 85
9.4.1 Experiments . . . 86
9.5 Image Thresholding Problem . . . 90
9.5.1 Exhaustive vs. Evolutionary Approach . . . 93
9.5.2 Thresholding with Variable Number of Thresholds . . . 96
9.5.3 License Plate Recognition by Using Thresholding . . . 106
9.6 Clustering Problem . . . 110
9.6.1 Evolutionary Clustering . . . 111
9.6.2 Verification of the Method . . . 112

10 Conclusion 128

Bibliography 130

Author's Bibliography 141

1 INTRODUCTION

These days, optimization is used in almost every discipline of engineering. Optimization is a process of finding the best solution from a set of available solutions. The quality of a solution is defined by fitness values calculated from fitness (objective, cost) functions. The fitness functions describe the behavior of an optimized system with properties called decision variables e.g. dimensions, reliability, or a price of a product. Therefore, fitness values depend on the decision variables of the optimized system. The optimization process is a process of finding minima (or maxima) of fitness functions.

If the system is described by one fitness function, the problem is called the Single-Objective Optimization Problem (SOOP). If there are more fitness functions, the problem is called multi-objective (MOOP). If the objectives are conflicting, the result is a set of optimal trade-off solutions called Pareto-front.

The Pareto-Front, a set of trade-off solutions, is named after an Italian economist Vilfred Pareto [1]. He defined that a member of the Pareto-front has to satisfy the Pareto efficiency: improvement of the solution in one objective has to deteriorate the quality of all other objectives.

Most of the real-world optimization problems are by its nature multi-objective and the objectives are also conflicting. This aimed the research to develop various multi-objective optimization methods.

A common optimization problem has a fixed number of decision variables. Therefore, the optimization algorithm knows the dimensionality of the decision space and tries to find the optimal position. Its fitness function depends only on decision variables. However, there are some optimization problems where the fitness function depends on the number of components of the decision vector as well. Such problems are called problems with a Variable Number of Dimensions (VND). When optimizing VND problems, an algorithm has to find not only the proper position vector but its dimension as well.

A real-life example of a VND problem is the placement of transmitters to broadcast a radio signal. An optimizer tries to find the optimal placement of the transmitters, the output power of the transmitters, but also the number of transmitters so that they cover the whole area with minimal overlaps. Each transmitter is defined by three decision variables – coordinates x, y, and $power$. The first fitness function describes the area covered by the signal, and the second fitness function is the overlapped area. Other fitness functions may reflect the cost of the connection between the backbone network and the proposed locations of the transmitters. This problem is elaborated in Section 9.3. The third fitness function there minimizes the number of transmitters.

The thesis is organized as follows: the next chapter deals with the theoretical background of the thesis. Afterward, a survey of the previous work is given, followed by the motivation and objectives of this thesis. Chapter 4 presents the Fast Optimization ProcedureS (FOPS) optimization framework. The framework was developed as a part of this thesis. Chapter 5 shows the methodology for the creation of benchmark problems with a

variable number of dimensions. The following two chapters propose the novel VND-GDE3 and VND-MOPSO optimization algorithms, respectively. Chapter 8 contains the verification of the proposed VND algorithms on the benchmark problems, and Chapter 9 shows several examples of exploiting the optimization with a variable number of dimensions in real-life problems. Finally, Chapter 10 concludes the thesis.

2 THEORETICAL BACKGROUND

This chapter treats the general concepts of optimization. In the beginning, the difference between single- and multi-objective optimization methods is shown. Afterward, the performance metrics for multi-objective optimization are described. The rest of the chapter presents the non-parametric statistic tests. Both the performance metrics and the statistic tests are used for the performance assessment of optimization methods.

The optimization methods are divided into two groups – local and global methods. Local methods are commonly based on a derivation of a fitness function, which is an intuitive approach with the benefit of rapid convergence into extremes of the fitness function. The problem is that although they converge to an extreme, it is not guaranteed whether it is the global extreme or a local one. The performance of a local method is affected by the initial guess, which the user has to define.

Note that extremes (minima and maxima) are discussed here generally, but for the rest of this thesis, only the minimization optimization problems are taken into account. If the fitness function is to be maximized, it is simply converted to a minimization function by multiplying the fitness value by -1.

Global methods overcome the risk of being caught in a local optimum of a fitness function. Usually, the global optimization method works with multiple sets of decision variables that are modified in each iteration of an algorithm. These population-based and stochastic methods, inspired by the theory of evolution, are generally called Evolutionary Algorithms (EAs), which covers Genetic Algorithms (GA) [2], Particle Swarm Optimization (PSO) [3], Differential Evolution (DE) [4], etc.

Although the researchers initially worked with single-objective optimization problems, it was only by assuming huge simplifications. As was mentioned before, most nature-inspired optimization tasks are multi-objective ones. Therefore, the need for multi-objective optimization algorithms soon emerged.

An optimization problem can be described by:

$$\begin{aligned} \text{Minimize:} \quad & f_m(\mathbf{x}), && m = 1, 2, \ldots, M, && (2.1)\\ \text{subject to:} \quad & g_k(\mathbf{x}) \geq 0, && k = 1, 2, \ldots, K, && (2.2)\\ & x_j^{(\min)} \leq x_j \leq x_j^{(\max)}, && j = 1, 2, \ldots, D, && (2.3) \end{aligned}$$

where M denotes the number of objectives, D denotes the number of decision variables, $\mathbf{x}$ is the vector of decision variables, $\mathbf{x}^{(\min)}$ and $\mathbf{x}^{(\max)}$ are the lower and upper bounds of decision variables, and K is the number of constraints g.

2.1 Single-objective Optimization

Single-objective optimization methods were used for a long time, even on problems with multiple objective functions [5, 6]. The problem is that a single-objective optimization produces single solution as a result. When the multiple objectives are conflicting, it is

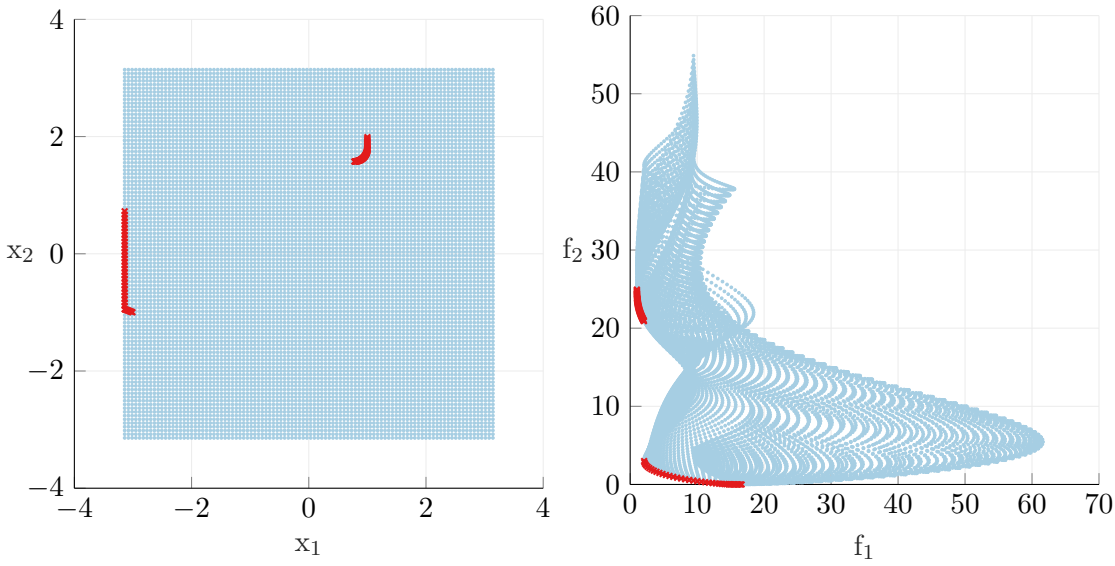
advantageous to have a set of trade-off solutions, i.e. Pareto-front. A Pareto-front with a single-objective algorithm can only be achieved by optimizing the same problem several times with different settings. There are various methods to handle multiple objective functions by a single-objective method.

The simplest and most commonly used method for transforming MOOP to SOOP is the Weighted Sum Method (WSM). This method converts M objective functions into one fitness function F:

$$F = \sum_{m=1}^{M} w_m f_m, \tag{2.4}$$

where w_m denotes the weight of the m-th objective function. A difficulty with the WSM method is that the user has to determine the weights for all the objectives and the optimization process produces only one solution that strongly depends on the user-specified weights. Therefore, the user has to have some knowledge about the shape of the Pareto-front, but it is unknown in most cases.

There are other methods such as the ϵ-Constrained Method [7] or the Rotated Weighted Metric Method [5, Chapter 3], but all these methods need additional knowledge about the optimized problem and produce only one solution from the Pareto-front per one simulation run. The need for a "pure" multi-objective optimizer has arisen with more complex optimization problems and shortcomings of single-objective methods.



(a) Decision space

(b) Objective space

Figure 2.1: Both spaces of the Poloni's study. Dark red "×" - true Pareto-front, light blue "·" - dominated solutions.

2.2 Multi-objective Optimization

The "pure" multi-objective optimization techniques originated at the turn of the 21st century. In most cases, the popular and well-known algorithms were modified to handle several objective functions – Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II) [8], the successor of GA; Multi-objective Particle Swarm Optimization (MOPSO) [9], the successor of PSO; Generalized Differential Evolution (GDE3) [10], the successor of DE; and other EAs and their numerous modifications [11].

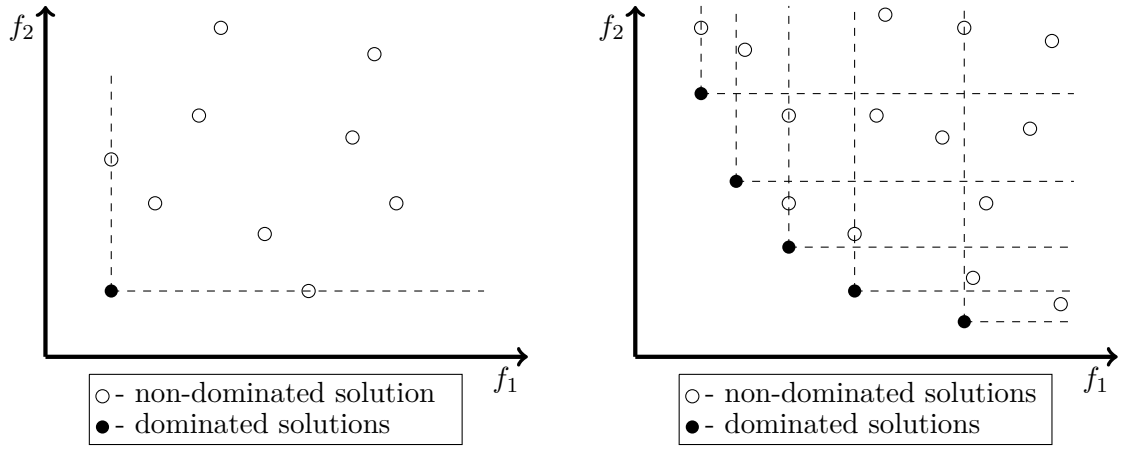
The multi-objective optimization algorithm confronts two fundamental requirements:

- Minimize the distance between found solutions produced by the optimization algorithm and the true Pareto-front.
- Maximize the spread of found trade-off solutions, so the solutions are distributed as uniformly as possible over the whole Pareto-front.

The multi-objective optimization process deals with a finite number of fitness functions. There are two spaces in multi-objective optimization – the decision space and the objective space. Both spaces are connected by fitness functions. Figure 2.1 shows both spaces of Poloni's study [5] (dark red "×" solutions represent the true Pareto-front).

2.2.1 Principle of Dominance

In the case of a single-objective problem, the best solution in a set of solutions is simply the one with the lowest fitness value. However, when the problem is multi-objective and the solutions are described by multiple fitness values, it is necessary to take into account all the objectives at once. Most multi-objective algorithms use the concept of dominance [5]. The principle of dominance is a comparison of two solutions with respect to all fitness values. There are three scenarios for two solutions $\mathbf{x}_1$ and $\mathbf{x}_2$:



(a) One solution dominates all the other solutions.

(b) Set of mutually non-dominated solutions (filled circles).

Figure 2.2: Principle of dominance [5].

- The solution $\mathbf{x}_1$ dominates the solution $\mathbf{x}_2$.
- The solution $\mathbf{x}_1$ is dominated by the solution $\mathbf{x}_2$.
- The solutions $\mathbf{x}_1$ and $\mathbf{x}_2$ are non-dominated.

The solution $\mathbf{x}_1$ is said to dominate the solution $\mathbf{x}_2$ if both following conditions are true:

- The solution $\mathbf{x}_1$ is no worse than the solution $\mathbf{x}_2$ in all objectives.
- The solution $\mathbf{x}_1$ is strictly better than the solution $\mathbf{x}_2$ in at least one objective.

Figure 2.2 further clarifies the principle of dominance. In Figure 2.2a, the non-dominated solution (full circle) is not worse in any objective compared to dominated solutions (empty circles) and strictly better in one or both objectives. Figure 2.2b shows several non-dominated solutions. One of any two non-dominated solutions (full circles) has the first fitness value lower than the other non-dominated solution but is worse in the second objective and vice versa. Therefore, it can not be said which non-dominated solution is better. In other words, trade-off solutions were found.

2.2.2 Non-dominated Sorting

The non-dominated sorting procedure determines which solutions from a set of solutions Q are non-dominated, based on the dominance principle. The simplest but the time-demanding approach is to compare all solutions from set Q with all the other solutions of set Q.

The most computationally efficient but difficult to understand is Kung et al.'s method [12]. The first step is to sort the set of solutions according to the first objective function value in descending order. Then a recursive *Front* function is called. In the *Front* function, the set of solutions is halved to the top (T) and bottom (B) sub-populations repeatedly until one of the sub-populations has only one member. When any of the sub-populations has exactly one member, the merging phase begins. Solutions of B are checked for domination by the members of T. All non-dominated solutions of B are combined with members of T. All such merged solutions are returned as a result of the *Front* function.

2.2.3 Maintenance of Diversity

The goal of multi-objective optimization is to find as many non-dominated solutions as possible, but also as diverse as possible. The maximal number of trade-off solutions in the process of optimization is usually fixed due to the growing computational complexity. A common assumption is that a convenient number of non-dominated solutions is identical to the number of agents in the population. The overall number of trade-off solutions can rapidly grow during the optimization process, which leads to a computationally demanding non-dominated sorting routine in every iteration of an algorithm. Therefore, some non-dominated solutions have to be discarded from time to time. However, it is not a simple task to determine which non-dominated solution is worse than the other. When the

diversity of the non-dominated set is to be maintained, only non-dominated solutions that are less crowded in the objective space should be preserved rather than the one with multiple neighbors close to it.

The closeness of neighboring solutions is estimated by crowding distance [8]. The crowding distance of a given solution is the combination of Euclidean distances from two neighboring (closest) solutions in objective space, as shown in Figure 2.3.

Two Euclidean distances from the nearest neighboring solutions are multiplied, and a solution with the smallest crowding distance in the set of non-dominated solutions should be eliminated. If another solution has to be discarded, only the crowding distances of neighbors of the eliminated solution have to be updated while the crowding distances of

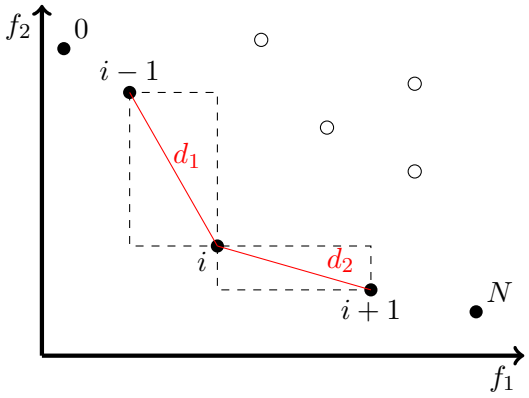
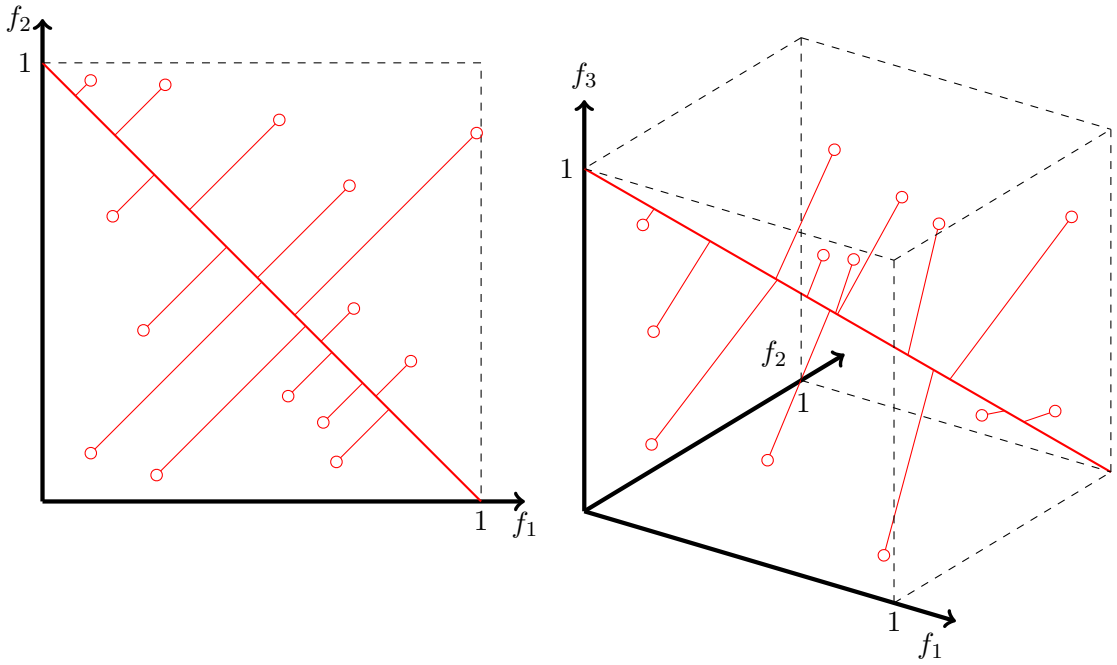


Figure 2.3: Crowding distance estimation [8].



(a) Two-dimensional objective space. **(b)** Three-dimensional objective space.

Figure 2.4: Principle of Equal-average Nearest Neighbor Search method [13].

all the other solutions remain the same.

The crowding distance technique is best suited for two-objective problems. Many-objective (3 objectives or more) spaces require modification of the crowding distance calculation. The difference resides in the conversion of many-objective space to a two-objective space just for the crowding distance calculation. Equal-average Nearest Neighbour Search (ENNS) method is described in [13]. This method projects M-dimensional objective space to a line that traverses it. If the objective space is normalized, then the best projection vector for minimization of all the objectives goes through point $\{0, 0, \ldots, 0, 1\}$ and point $\{1, 1, \ldots, 1, 0\}$, as shown in Figure 2.4.

2.3 Performance Metrics

The result of multi-objective optimization is a set of solutions, which makes it hard to decide whether one set of solutions is better than the other [5]. In order to simplify the decision, a performance metric represents an entire Pareto-front with a single number. The value of a metric depends on the quality of the solution. However, there are several metrics known in the open literature. Each metric classifies the Pareto-front from a different point of view. For example, a solution with the best value of generational distance (see Subsection 2.3.1) can have poor diversity. This suggests the basic sorting of performance metrics – either they describe the closeness of non-dominated solutions to the true Pareto-front, the spread of the solutions over the Pareto-front, or both.

2.3.1 Generational Distance

The generational distance (GD [14]) finds the average Euclidean distance of a member of set Q (found Pareto-front) from the closest member of set P^* (true Pareto-front) according to:

$$\mathrm{GD} = \frac{\sum_{i=1}^{|Q|} d_i}{|Q|}, \tag{2.5}$$

where the d_i is the Euclidean distance between solution $i \in Q$ and the nearest member of P^*.

$$d_i = \min_{k=1}^{|P^*|} \sqrt{\sum_{m=1}^{M} \left(f_m^{(i)} - f_m^{*(k)} \right)^2}, \tag{2.6}$$

where $f_m^{*(k)}$ is the m-th objective function value of the k-th member of P^* and $f_m^{(i)}$ is the m-th objective function value of the i-th member of Q. The metric expresses the closeness of non-dominated solutions to the true Pareto-front. Therefore, true Pareto-front must be known to calculate generational distance.

2.3.2 Spread

The spread metric (Δ [8]) measures the quality of distribution of non-dominated solutions, but it also takes into account the distance from true Pareto-front extremes. The metric is described by:

$$\Delta = \frac{d^e + \sum_{i=1}^{|Q|} \left| d_i - \bar{d} \right|}{d^e + |Q|\,\bar{d}}, \tag{2.7}$$

where d_i is the average of two Euclidean distances from two neighboring solutions from the non-dominated set of i-th solution and $\bar{d}$ is a mean value of d_i values. The parameter d^e is the sum of Euclidean distances between corresponding extremes of sets Q and P^*.

Value of spread can be zero only if extremes of the found non-dominated set are identical to the extremes of true Pareto-front and simultaneously all the distances between neighboring solutions are equal to their mean value (solutions are uniformly distributed).

2.3.3 Hypervolume

The hypervolume metric (HV [15]) calculates the volume in the objective space covered by the members of set Q. Hypervolume is a sum of volumes of hypercubes v_i. Each hypercube v_i is constructed with a reference point W and solution $\mathbf{x}_i \in Q$ as diagonal corners of the hypercube:

$$HV = \text{volume}\left(\cup_{i=1}^{|Q|} v_i\right). \tag{2.8}$$

The reference point W has a significant influence on the HV scale. The reference point for the hypervolume calculation is usually defined as the nadir point of objective space (the upper bound of each objective in the entire true Pareto-front [5, Chapter 2]). The scale of hypervolume value depends on the problem. But on the other hand, it expresses both closeness of the non-dominated set from the true Pareto-front and the distribution of solutions. The bigger the value of hypervolume, the better solution.

The different range of hypervolume values for each problem may influence the clarity of results, mostly in cases where multiple optimization problems are part of a single comparative study. In such cases, the hypervolume value can be normalized by the hypervolume of true Pareto-front such as:

$$\text{HVR} = \frac{\text{HV}\,(Q)}{\text{HV}\,(P^*)}. \tag{2.9}$$

However, hypervolume ratio (HVR) becomes insensitive to small changes of HV value if large values of HV are considered. Therefore, distance hypervolume metric is used in this work, which seems to reflect small changes more clearly. The lower the value of distance hypervolume is, the better the Pareto-front. Distance hypervolume is defined as:

$$\text{dHV} = \text{HV}\,(P^*) - \text{HV}\,(Q). \tag{2.10}$$

2.3.4 Number of Variables Deviation

The number of variables deviation (nVD) is an in-house metric. It expresses the average difference between the number of decision variables of a solution $D(\mathbf{x})$ and the optimal number of decision variables for the corresponding solution $D^{(\text{opt})}(\mathbf{x})$:

$$\text{nVD} = \frac{\sum_i^{|P|} \left| D(\mathbf{x}_i) - D^{(\text{opt})}(\mathbf{x}_i) \right|}{|P|}. \tag{2.11}$$

2.4 Non-parametric Statistical Tests

Performance metrics were described in the previous section. A performance metric can express the quality of the result of the single run of optimization. However, when there are multiple optimization algorithms to be compared, and the stochastic processes are repeated many times, it becomes difficult to estimate the superior algorithm. A lot can be derived from the mean or median values of a given metric [16]. Nonetheless, it cannot be said that the difference between the mean (median) values has a statistical significance without the use of statistical tests.

It can not be determined definitively whether one optimizer is better than the other because only a finite number of samples can be known. Instead, a probability (called p-value) is calculated by the statistical test. Based on a chosen level of significance ($alpha$), a hypothesis that one optimizer is better than the other can be assumed if the p-value is lower than the $alpha$.

There are many types of statistical tests to be found in literature. For the purposes of this thesis, only two commonly used non-parametric statistical tests will be explained:

- Wilcoxon's test,
- Friedman test,

2.4.1 Wilcoxon's Signed Ranks Test

Wilcoxon's test is a pairwise comparison statistical test. Such tests are designed to compare the performance of two algorithms when applied to a common set of problems [17].

This test is used for answering the following question: do two samples represent two different populations? Therefore, it detects significant differences between two sample means.

Let the d_i be the difference between the performance scores (i.e. metric values) for the i-th degree of freedom (i.e. i-th testing problem or i-th repetition of the algorithm run). The differences are then ranked according to their absolute values. Therefore, the lowest absolute difference has a rank of one, the second to lowest difference has a rank of two, and so on. The exceptions occur when there are two or more differences tied. In such cases, the identical mean rank value is assigned to all of them.

Now, let R^+ be the sum of ranks where the first algorithm outperformed the second, and R^- the sum of ranks for the opposite:

$$R^+ = \sum_{d_i>0} \text{rank}\,(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}\,(d_i)\,, \tag{2.12}$$

$$R^- = \sum_{d_i<0} \text{rank}\,(d_i) + \frac{1}{2}\sum_{d_i=0} \text{rank}\,(d_i)\,. \tag{2.13}$$

Note that indifferent scores are split evenly among the sums. If there is an odd number of ties, one is ignored.

Finally, if the smaller of the sums, $T = \min\left(\{R^+, R^-\}\right)$, is lower than or equal to the critical value of the distribution of Wilcoxon for n degrees of freedom (e.g. number of runs of algorithms), the samples represent statistically different populations.

The Wilcoxon's test can not be calculated for $n < 5$, and the table of critical values for Wilcoxon's test [18] counts with a maximum of $n = 50$. Therefore, if the number of degrees of freedom $n > 50$, the calculation using z-score should be used as follows:

$$z = \frac{T - \frac{n(n+1)}{4}}{\sqrt{\frac{n(n+1)(2n+1)}{24}}}. \tag{2.14}$$

If the $z > -1.96$, then the samples represent statistically different populations. Normal cumulative distribution function can be used to obtain a p-value from the z-value.

2.4.2 Friedman's Test

Friedman's test [19] is a multiple comparison test that aims to detect significant differences between the behavior of two or more algorithms.

Similarly as in Wilcoxon's test, the performance scores are converted into ranks. However, ranks are assigned for each degree of freedom i (e.g. run) separately. If there are $k = 3$ algorithms, than ranks for the first degree of freedom are r_1^1, r_1^2, and r_1^3. Afterward, an average of the ranks over all $i \in [1, n]$ is obtained for each algorithm $R_j = \frac{1}{n}\sum_i r_i^j$.

The Friedman statistic F_f can be then computed as:

$$F_f = \frac{12n}{k\,(k+1)}\left[\sum_j R_j^2 - \frac{k\,(k+1)^2}{4}\right], \tag{2.15}$$

which is distributed according to the χ^2 distribution with $k-1$ degrees of freedom. Therefore, obtaining the p-value is possible by cumulative distribution function for χ^2 distribution. If the p-value is lower than the chosen level of significance, the samples represent statistically different populations.

If there are significant differences in the population, unadjusted p-values can be obtained by the following formula:

$$z = (R_i - R_j)\,/\sqrt{\frac{k\,(k+1)}{6n}}, \tag{2.16}$$

where R_i and R_j are the average rankings by the Friedman test of compared algorithms. In practice, there is often a control algorithm among the set to which all the p-values are related.

Afterward, the unadjusted p-values can be adjusted by various post-hoc procedures. The Bonferroni-Dunn procedure is the simplest one, has a little power, but offers a simple visualization of the results. The most often used procedures are Hochberg's or Holm's.

Holm's procedure adjusts the p-value in a step-down manner. It means that the unadjusted p-values are sorted from the smallest to the largest $(p_1, p_2, \ldots, p_{k-1})$. To obtain the i-th adjusted p-value we use $p_{i,adj} = \min(\{v, 1\})$, where $v = \max((k-j)\, p_j)$ and $j \in [1, i]$. Hochberg's procedure is calculated similarly. However, it is done so in a step-up manner.

3 SURVEY OF THE PREVIOUS WORK

The thought of the optimization problems with a variable number of dimensions is almost as old as the optimization algorithms themselves [20]. However, for many years, researchers aimed for the simplicity of the optimization process and worked with fixed-length decision vectors. In 1989, the author of the original Genetic Algorithms stated in [21] that "Nature has formed its genotypes by progressing from simple to more complex life forms" and proposed the Messy Genetic Algorithm (mGA) - a first algorithm to be found working with a variable number of dimensions.

3.1 As the Time Went By

Authors of the mGA [21] call an optimization algorithm with fixed coding the neat optimization algorithm. Their publication explains that there are real-world problems whose optimal solutions have deceptive genomes, which makes it hard for the neat genetic algorithm to find the optimal solution. An example of such deceptive decision space may be a local optimum with string $\{1, 0, 0, 0, 0, 1\}$ and a global optimum with string $\{0, 1, 1, 1, 1, 0\}$. It is improbable that the algorithm by coincidence falls upon both solutions. Reorganizing the chromosome such that the local solution is $\{0, 0, 1, 1, 1, 1\}$ and the global is $\{1, 1, 0, 0, 0, 0\}$ is much easier to explore, considering the common crossover operation. Authors used the variable-length genome for stochastic reordering of the genome to prevent the search from falling into sub-optimal solutions. In other words, the algorithm works with a fixed number of decision variables but uses variable-length chromosomes. The length of the genome is changed by the cut-and-splice crossover operator. After such an operator, some decision variable might occur multiple times in the chromosome or be missing entirely. The redundant variables of the chromosome are discarded (either the first or the last occurrence is preserved). The missing variables are retrieved from the universal template.

Another frequently cited algorithm is the Speciation Adaptation Genetic Algorithm (SAGA) [22]. When the fixed-length crossover operator is used, no genes from the genome can be eliminated. However, in a variable-length crossover, such a risk exists, and essential parts of the genome might be missing. Therefore, the solution becomes unfeasible. Nonetheless, SAGA algorithms deal with it by implementing the Longest Common Subsequence metric (LCSS). In principle, the crossover position in the first parent solution is chosen randomly. Afterward, for every possible crossover position in the second chromosome, the LCSS metric is calculated, and the position with the highest score is used. Therefore, it explicitly attempts to preserve a complete genetic sequence.

A Structured Genetic Algorithm was proposed in [23]. This algorithm uses the second string along with the genome string. The second string determines which genes are activated or deactivated, therefore the number of components in the fitness function evaluation can vary. Nonetheless, the genome string length remains fixed in crossover and mutation operator. Such an approach can be called the hybrid VND technique and is

frequently used in the open literature to solve VND problems (either with GA algorithms or other EAs). The drawback of the fixed-length chromosome is that the algorithm works with the maximal dimensionality of the decision space all the time. Also, the algorithm is called structured because it uses a multi-layered chromosome, therefore disabling the gene in the higher layer disables all the subordinate genes.

A significant class of optimization problems insoluble with neat optimization algorithms is evolutionary programming. The modified genetic algorithm is used to generate a program in an arbitrary programming language in [24]. The number of instructions of the program can arbitrarily grow. Therefore, the problem has a variable number of dimensions. Koza used a genetic algorithm for generating the program for autonomous robots following a wall.

Another class of problems frequently tackled in the open literature is a grouping problem. The aim of the grouping problem is to group members of a set into a small number of families in order to optimize a fitness function. Clustering problems can be seen as a subclass of grouping problems. A Grouping Genetic Algorithm was proposed in [25] and exploited on many grouping problem applications.

Authors of Virtual Virus (VIV) [26] tried to mimic the biological genetics more closely. Genes in the genome should be independent of position, genomes might change its length, and the genome can contain non-coding regions along with duplicative and competing genes. Genomes in VIV adopt a four-letter alphabet, $G = \{\text{A,T,C,G}\}$, which corresponds better with biological systems. In crossover operator, they dealt with preserving the essential parts of the genome similarly to [22].

The crossover operator preserving the essential parts of the genome was further improved in [27]. The authors compared their Synapsing Variable-Length Crossover (SVLC) with crossover used in mGA, SAGA, and VIV. Unlike its predecessors, it searches for common subsequences in both parent chromosomes before determining the crossover positions. It uses the same metric LCSS, finds all the LCSSs in both chromosomes, and allows offsprings to inherit the entirety of common subsequences. Therefore, the SVLC exchanges only the differences between parent genomes. Although the SVLC is more computationally expensive than SAGA crossover (on the genomes of identical length), it imitates the biological crossover far better. Moreover, the authors claim that the length of produced solutions is generally smaller compared to the SAGA. Therefore, the computational cost is comparable if not lower.

Naturally, single-objective variable-length optimization algorithms were discovered at first, but multi-objective variants of VND algorithms emerged soon [28, 29, 30]. Although there are many modifications of Genetic Algorithms with the variable-length genome to be found in the open literature, most of them differ in genome representation and crossover and mutation operators (although the operators might be called differently): [23, 31, 32, 33, 34, 35, 36, 37, 38].

Until now, only genetic algorithms were covered. However, researchers soon tried to modify other algorithms as well. A cornerstone for Particle Swarm Optimization (PSO)

for a variable number of dimensions was laid by the author of the original PSO algorithm in [39]. The original real-coded PSO algorithm was transformed into the binary-coded algorithm. Although the authors did not consider the use of binary PSO for problems with a variable number of dimensions, other authors adopted it later for such applications. The trick is that there are header bits for each decision variable in the decision vector, which enables or disables the corresponding decision variable for the fitness evaluation. This corresponds to the activation string used in [23, 40]. It is a common ploy used not only in PSO-based algorithms (see [34, 41, 42, 43]).

A pure VND algorithm was proposed in [44] – Dimension Adaptive Particle Swarm Optimization. The problem in Particle Swarm Optimization is the velocity update equation:

$$\mathbf{v}_g = W \cdot \mathbf{v}_{g-1} + c_1 \cdot r_1 \cdot (\mathbf{x}_{\text{pbest}} - \mathbf{x}_{g-1}) + c_2 \cdot r_2 \cdot (\mathbf{x}_{\text{gbest}} - \mathbf{x}_{g-1}) \,, \tag{3.1}$$

where the current particle $\mathbf{x}_{g-1}$ mingles with its attractors (personal best position $\mathbf{x}_{\text{pbest}}$ and global best position $\mathbf{x}_{\text{gbest}}$. Note that w is inertia weight, c_1 and c_2 are cognitive and social learning factors, respectively, r_1 and r_2 are random values, and g is a time step. In a pure VND algorithm, any particle may have a different number of decision variables. Therefore, vectors with a different number of components are to be subtracted in (3.1). This mapping procedure handles the complicated subtraction as follows:

- If the number of components of particle and its attractors is equal, a generic velocity update equation is used.
- If the number of components of the particle is lower than the number of components of the attractor, only corresponding components of the attractor are used.
- If the number of components of the particle is higher than the number of components of the attractor, these components are updated only by the inertia weight of the particle (and not its attractors).

Note that the dimensionality of each particle can vary by one in each iteration within $[D_{\min}, D_{\max}]$ with a certain probability.

Multidimensional Particle Swarm Optimization (MD-PSO) was proposed in [45]. According to [46], the basic PSO is unable to work with high-dimensional problems. Also, according to [47], it is prone to be trapped in a local optimum due to the attraction of particles to the global best solution. MD-PSO algorithm uses Fractional Global Best Formation technique, which utilizes an artificial global best (aGB) particle by collecting the best components among the population and fractionally creating aGB. It is claimed that artificial global best has a better potential to lead particles to the global optimum. Finally, a special velocity update equation allows a particle to navigate through dimensions. Accordingly, an algorithm keeps track of all the visited dimensionalities of global and personal bests. Therefore, when the Equation (3.1) is used afterward, the particle is mated with attractors of equivalent dimensionality.

The PSO-VND algorithm was proposed in [48]. The PSO-VND algorithm was used as the foundation for the proposed methodology for multi-objective optimization algorithms and will be thoroughly covered in Section 3.2.

Modifications of Differential Evolution for optimization with a variable number of dimensions can also be found in the literature. For example, [49] proposed a Grammatical Differential Evolution for evolutionary programming, Variable-size Multi-objective Differential Evolution proposed in [50] utilizes the secondary vector to handle the number of active decision variables. Note that this is the same impure-VND approach as described earlier. Another Differential Evolution-based algorithm was proposed in [51].

Genetic Algorithm, Particle Swarm Optimization, and Differential Evolution are not the only foundations of algorithms for a variable number of dimensions. There are other methods to be found in the literature: [52] proposed the Grouping Harmony Search and [53] extended the Grouping Harmony Search for multi-objective problems, [54] presented the Grouping Coral Reef Optimization algorithm, or [55] adapted the Simulated Annealing for a specific VND multi-objective problem.

3.2 Particle Swarm Optimization for Variable Number of Dimensions

The VND-PSO algorithm [48] is based on a conventional PSO method [56]. Particle Swarm Optimization is an evolutionary algorithm that simulates the movement of a swarm of bees searching for food.

The general flowchart of the PSO-VND method is shown in Figure 3.1.

The position of each agent is changed according to its own experience and that of its neighbors. The position $\mathbf{x}_g$ in subsequent iteration is changed by adding velocity $\mathbf{v}_g$ to a

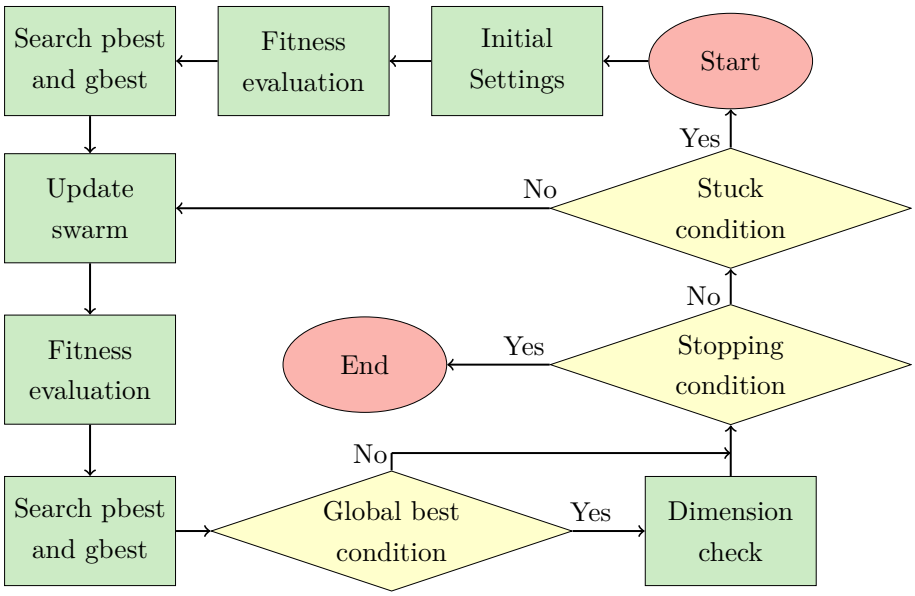


Figure 3.1: Flowchart of the PSO-VND method [48].

previous position $\mathbf{x}_{g-1}$:

$$\mathbf{x}_g = \mathbf{x}_{g-1} + \mathbf{v}_g, \tag{3.2}$$

where the velocity vector $\mathbf{v}_g$ is defined by equation:

$$\mathbf{v}_g = w \cdot \mathbf{v}_{g-1} + c_1 \cdot r_1 \cdot (\mathbf{x}_{\text{pbest}} - \mathbf{x}_{g-1}) + c_2 \cdot r_2 \cdot (\mathbf{x}_{\text{gbest}} - \mathbf{x}_{g-1}) \,. \tag{3.3}$$

Note that this equation is the same as used in the previous section. Therefore, the meaning of individual components in the equation can be found there.

3.2.1 Position Update

In the conventional PSO method, all $\mathbf{x}_{g-1}$, $\mathbf{x}_{\text{pbest}}$ and $\mathbf{x}_{\text{gbest}}$ position vectors from equations (3.2) and (3.3) are of the same size. However, the velocity update equation in PSO-VND mixes the decision vectors with a different number of components into a new velocity vector. Three probabilities are introduced for handling the variable number of dimensions:

- p_{pbest} probability of agent's position to take the dimension of the personal best D_{pbest},
- p_{gbest} probability of agent's position to take the dimension of the global best D_{gbest},
- p_x probability of agent's position to keep its dimensionality $D_{\mathbf{x}}$.

Probabilities p_{pbest}, p_{gbest}, and p_x decides which of the three dimensions (D_{pbest}, D_{gbest}, and $D_{\mathbf{x}}$) will an actual particle follow. Note, that $p_{\text{gbest}} + p_{\text{pbest}} + p_x = 1$ and the dimensionality to follow (D_{new}) is chosen based on the randomly generated number in the unit interval divided into three subparts according to the probabilities p_{gbest}, p_{pbest}, and p_x.

When the D_{new} is resolved, three artificial solutions are derived from a current particle, the personal best particle, and the global best particle. However, they all have the same size D_{new}. Note that missing components are replenished randomly according to:

$$x_j = x_j^{(\min)} + \text{rnd}\left(x_j^{(\max)} - x_j^{(\min)}\right), \tag{3.4}$$

where $\text{rnd}\,(\cdot)$ is a randomly generated number from interval $[0, 1]$, and $x_j^{(\max)}$ and $x_j^{(\min)}$ are upper and lower bound of n-th decision variable, respectively. Now, the generic equation (3.3) can be used because all the artificial solutions has the same number of components.

3.2.2 Dimension Check

The dimension check feature is introduced in PSO-VND to enhance the dimensionality exploration properties of the algorithm. It is based on the idea that optimal solutions (e.g. local optimum and global optimum) can have a different number of decision variables but can share several common components. Imagine the clustering problem where the three

clusters are properly determined, but the solution should have four clusters. One of the clusters needs to be divided in order to achieve the global optimum.

There are two types of dimension check: best and random. If the best dimension check is enabled, global best solutions are taken as a template. Afterward, other solutions from the current iteration are examined whether they have more decision variables than the template already has. If so, the template is repeatedly replenished until the maximal number of components is reached. In case that no solution has the number of decision variables equal to the maximal possible dimension of a problem, the remaining components are assigned randomly. If the random dimension check is enabled, only the global best solutions are taken as a template, and the remaining dimensions are filled in randomly. Therefore, the computational burden of searching through other solutions is saved.

When the template solution is known, the fitness function is evaluated for every possible dimensionality of the problem from the components of the template solution. If any of the evaluated partial solutions outperform the current global best solution, it is employed as a new global best solution. Also, the corresponding solution in population and its personal best is replaced.

Although the dimension check can significantly increase the overall number of fitness function evaluations, it positively affects the performance of the optimization method.

3.3 Motivation

As was mentioned at the beginning of this chapter, most optimization algorithms proposed in previous decades use fixed decision space. There are many evolutionary algorithms and their modifications, but only a few of them can handle problems with a variable number of dimensions. Unfortunately, many applications require a VND approach. Moreover, as Goldberg stated in [21], even life on earth evolved through dealing with problems with a variable number of dimensions.

Speaking of VND applications, there are numerous publications dealing with VND problems. Frequently tackled are evolutionary programming problems or grouping problems. The problem presented in [25] is the bin packing problem where an arbitrary number of objects are to be packed into a minimal possible number of bins. Another problem presented there is the line balancing problem where the distribution of tasks over workstations is being optimized so that no workstation takes longer than the cycle time to complete all the tasks. Any clustering problem where the number of clusters is not known a priori is a VND optimization problem. In [57], a container pre-marshaling problem is being solved.

Focusing on electrical engineering, there is the term Evolutionary Electronics for evolutionary optimization of problems related to electronics [34]. The paper tackles several problems:

- Digital circuits where sum of product terms are sought. However, it is not known a priori how many product terms for a given digital circuit is needed.

- Analog circuits where either passive filters (algorithm finds the number, value, and position of resistors, capacitor, and inductors) or transistor-based amplifiers (algorithm finds the value, type of component, and connection points) are designed.

In [MM1], the PSO-VND algorithm is used for the PCB decoupling problem. Wireless transmitter placement is a common optimization problem requiring variable decision space in order to find the optimal number of transmitters as well [MM2], [58, 54, 30, 41].

Applications listed here are mostly optimized by single-objective optimization algorithms, but a multi-objective definition of an optimization problem is more natural, because the nature itself is full of contrasts. Leaving out any criterion to fit the problem for a single-objective optimization algorithm may be tricky, if not odd, entirely.

However, only a few multi-objective algorithms were adapted to work with a variable number of dimensions so far [28, 59, 60, 58, 30, 55, 53]. It is also important to note that some papers claim a multi-objective VND algorithm is being proposed, but in truth they are either:

- quasi multi-objective – aggregates several objectives into one [59],
- impure-VND – performs update position operator with fixed-length decision vectors [60, 58, 53].

Aggregating multiple objectives into a single one is a tricky issue. One has to have some additional knowledge about the optimized problem in order to satisfyingly set the aggregating method and therefore obtain a good trade-off solution. However, the fact that the problem properties are unknown is often the original reason the optimization is performed.

The method is called impure-VND in this thesis if a VND problem is being optimized, but a fixed decision space is eventually used in the update position operator. Although such an algorithm may be able to find optimal dimensionality, it necessarily performs position update with the whole position vectors. Therefore, the performance of an algorithm is wasted on exploring unfeasible regions of decision space (regions unused in fitness function evaluation).

Finally, most of the algorithms listed in this chapter are modifications of genetic algorithms. Therefore, the decision space is discrete (binary-coded). However, our proposed methodology for VND algorithms is suitable for different optimization algorithms, including the real-coded ones.

3.4 Dissertation Objectives

The following list summarizes the most important objectives of this thesis:

- develop an optimization framework suitable for algorithms with a variable number of dimensions,
- create a library of benchmark problems with a variable number of dimensions,

- propose new algorithms for optimization with a variable number of dimensions,
- verify the performance of the proposed methods on a set of benchmark problems,
- exploit new algorithms on several real-world applications.

Implementing novel optimization techniques in any optimization framework is a beneficial step in the design process. The optimization framework not only simplifies the designing of the algorithm but its setting and verifying as well. However, maintaining problems with a variable number of dimensions casts a special requirement on the framework itself. Due to the requirement that any agent in a population can have a different number of components, no existing optimization framework was suitable for our cause. Therefore, a new optimization framework in MATLAB was developed, which makes it easier to implement an algorithm, run a simulation, view its results, or compare its performance to other algorithms.

The library of benchmark problems is a necessary part of the verification of the algorithm's convergence properties. A proper set of benchmarking problems has several individual problems with all kinds of difficulties:

- various number of decision variables,
- various number of objective functions,
- different shapes of Pareto-fronts,
- uni-modal vs. multi-modal problems,
- separable vs non-separable problems.

Advantageously, if the true Pareto-front of a benchmark problem is known, it is possible to compare the found Pareto-fronts to the true Pareto-front.

Deriving new stochastic optimization methods is a crucial part of this thesis. New methods will be exploitable on a special class of optimization problems – problems with a variable number of dimensions. Such an optimization algorithm not only determines proper values of the decision vector but the number of decision variables as well.

Verifying the performance of proposed methods is a substantial step in the design process. Each novel algorithm will be compared to the algorithm with a fixed number of dimensions and a hybrid-VND method in a scenario that tries not to favor any of the methods.

Finally, novel algorithms will be exploited on several real-world applications related to the field of electrical engineering:

- Anisotropic band-stop filter design,
- Antenna array design,
- Transmitter placement problem,
- Digital circuit design,
- Automated image thresholding,
- Clustering problem.

4 FOPS

With the increasing complexity of the optimization problems, new optimization procedures were developed over the past decades [61, 11]. In recent years, great effort has been made in the development of novel optimization methods, especially multi-objective evolutionary algorithms (MOEAs). Note that parts of this chapter are reprinted from [62, 63, 64].

The motivation of researches for the further development of optimization techniques is that there simply does not exist a universal method suitable for all kinds of optimization problems. Wolpert proposed the "no free lunch" theorem for the optimization [65]. It says that for any algorithm, any elevated performance over one class of problems is exactly paid for over another class in performance. Therefore, when an unknown optimization problem is given, it is common to use various optimization methods and see which one serves the best to our needs. Such practice encourages the development of optimization tools.

There are several MOEA frameworks proposed in [66, 67, 68, 69, 70]. Most of the optimization frameworks have none at all or complex GUI. Therefore, its user-friendliness, especially for beginners, is low. Also, only a few of the frameworks provide the user with the ability to implement his own optimization method or even configure its controlling parameters. The paper [70] further reviewed and compared the experimental environments. In paper [MM2], we present a new optimization framework – Fast Optimization ProcedureS (FOPS).

Optimization algorithms that can handle the variable number of decision variables are being developed these days. To find an optimization framework where procedures with a variable number of dimensions are covered leads to the ones employing surrogate modeling techniques. An example of such a framework is SurrogateModeling Lab [71] or Altair HyperStudy [72]. Surrogate modeling reduces the number of degrees of freedom. In other words, dimensions that have little or no effect on the fitness values are discarded from the search. However, this can not be considered as a pure-VND approach because an inaccurate estimate of the surrogate model discarding certain dimensions may cripple the following search. Moreover, a pure-VND multi-objective algorithm can locate a Pareto-front where each of the solutions can have a different number of components. The FOPS, on the contrary, is able to work with problems where each agent has a different number of decision variables.

Like any framework, FOPS offers many optimization algorithms to use, but only one of these algorithms can be used on a single problem at one time. Contrarily, the FOPS framework enables the user to join multiple optimization algorithms to create a *chain*. Algorithms in the *chain* are sequentially launched on a single problem. The following algorithm starts where the preceding algorithm ends. Therefore, the advantages of different optimization methods can be exploited within a solution to a single problem. For example, the connection of a VND method with a non-VND optimization method, where the first one finds the optimal dimensionality of the problem (e.g., the number of transmitters covering the area) and the second one explores the estimated decision space (e.g., the

exact transmitters' positioning).

Another rare feature, called fullControl, allows the user to access a simulation object inside the fitness function evaluation. By the term "simulation object" is meant the set of variables where all the position vectors, fitness values, and all the settings of an algorithm are held. Therefore, the user has full control over the optimization process and changes its effective performance on the fly. This functionality allows him to e.g. change dynamically the setting of an algorithm during an optimization run, and it is even possible to perform another, nested optimization (see Subsection 9.2).

The FOPS is a standalone MATLAB toolbox. Therefore, it is available for various operating systems. It includes methods for single- and multi-objective optimization. Currently, there are seven single-objective (plus one single-objective VND) and four multi-objective (plus two multi-objective VND) optimization methods, almost 110 benchmark problems of various types, and many performance metrics such as generational distance, hypervolume, spread, etc. The FOPS can be controlled from the command line or by Graphic User Interface (GUI).

Visualization of the results is an important part of the optimization process since it is capable of disclosing a relationship between different quantities. An uncommon feature is that the user can choose which dimension of an objective space or a decision space he wants to see. Both spaces can be combined. Moreover, all the benchmark problems included in the FOPS allows the user to plot its true Pareto-front (objective space) and/or true Pareto set (decision space). Therefore, the obtained results can be intuitively compared with the true optimal results. Note that the adverb *true* is used throughout the paper to distinguish between the solutions found by the optimization run and the optimal solutions to be found for a given problem, e.g. Pareto-front vs. true Pareto-front.

The user can extend the FOPS by his own optimization methods, set up his own problems, and also implement new performance metrics. All the optimization methods in FOPS can handle the discrete decision space because a simple but effective discretization method is available. Verification of the discretization method can be found in Section 4.3. The FOPS toolbox also includes a very simple surrogate optimization method called the Tolerance-based Surrogate Method. It is more thoroughly covered in Section 4.4. Moreover, additional surrogate optimization methods can be added by the user without difficulties.

4.1 Simulations in FOPS

The FOPS toolbox currently includes 16 optimization algorithms. They are listed in Table 4.1.

The FOPS framework includes an extensive benchmark problems library. Each problem is defined as a separate MATLAB file. There are 17 single-objective problems, 16 single-objective VND problems, 44 multi-objective problems, and 30 multi-objective VND benchmark problems. They are briefly summarized in Table 4.2.

Table 4.1: Optimization methods in FOPS.

Abbreviation	Algorithm name	Reference
SONEW	Newton method	[6]
SONEME	Nelder-Mead method	[73]
SOGA	Genetic Algorithm	[74]
SOPSO	Particle Swarm Optimization	[3]
SODE	Differential Evolution	[4]
SOSOMA	Self-Organizing Migration Algorithm	[75]
SOCMAES	Covariance Matrix Adaptation Evolution Strategy	[76]
VND-PSO	Particle Swarm Optimization for VND	[48]
NSGA-II	Elitist Non-dominated Sorting Genetic Algorithm	[8]
MOPSO	Multi-objective Particle Swarm Optimization	[9]
GDE3	Generalized Differential Evolution	[10]
MOSOMA	Multi-objective Self-Organizing Migration Algorithm	[77]
VLGDE3	Variable-length Generalized Differention Evolution	[50]
VLMOPSO	Variable-length MO Particle Swarm Optimization	[40]
VND-GDE3	Generalized Differential Evolution for VND	[62]
VND-MOPSO	Multi-objective Particle Swarm Optimization for VND	

Table 4.2: Benchmark problems in FOPS.

Abbreviation	Problem name	Reference
SOROS	Rosenbrock's function	[78]
SORAS	Rastrigin's function	[78]
SOACK	Ackley's function	[78]
SOEAS	Easom's function	[78]
and others from		[78, 79]
VNDKRAS	VND modified Rastrigin's function	[80]
VNDKROS	VND modified Rosenbrock's function	[80]
VNDMACK	VND modified Ackley's function	[48]
VNDMALP	VND modified Alpine function	[48]
and others from		[80, 48]
MODTLZ	Deb, Thiele, Laumanns, and Zitzler	[81]
MOLZ	Li, Zhang	[82]
MOUF	CEC 2009	[83]
MOWFG	Walking Fish Group	[84]
MOZDT	Zitzler, Deb, and Thiele	[85]
and others from		[5]
VNDMODTLZ	modified Deb, Thiele, Laumanns, and Zitzler	[MM4]
VNDMOZDT	modified Zitzler, Deb, and Thiele	[MM4]
VNDMOLZ	modified Li, Zhang	[MM4]
VNDMOUF	modified CEC 2009	[MM4]
VNDMOLI	multi-objective VND problems	[86]

Apart from the benchmark problems, which are mainly useful for the comparison of

optimization methods, the user can define his own optimization problem. The optimization problem is described by a MATLAB function which returns a structure. The structure contains several fields describing the optimization problem. There are many types of fields (see the FOPS documentation [MM4]), but only two fields are mandatory — `'fitness'` and `'limits'`.

4.1.1 Simulation Types

There are three different types of simulation in FOPS — *comparative study*, *chain*, and *task*. A *comparative study* can be seen as a set of several *chains* solving multiple problems repetitively. A *chain* is a set of *tasks* connected together to optimize a single problem. A *task* is a single algorithm optimizing a single problem.

The user can create multiple simulations of various types in the FOPS, modify their settings if necessary, and then launch the optimization process. When the optimization process is finished, the user can search through the results with a results suite (see Section 4.2).

There are several methods for maintaining the simulations in the FOPS. They are summarized in Table 4.3. The table also shows input arguments of individual methods, where the input arguments typed in bold font are mandatory, while the arguments in normal font are optional. This roughly indicates the fundamental concept of the FOPS: "type in only the necessary things and let the FOPS fill in the rest". There are numerous possibilities within the FOPS, but if the user does not want to or does not know how to type in advanced commands, the FOPS will still provide results.

Table 4.3: Simulation maintenance methods in FOPS.

addTask(**problem**, **algorithm**, settings, name, userData)
changeTaskSettings(**id**, **settings**)
renameTask(**id**, **newName**)
runTask(**id**)
deleteTask(**id**)
addChain(**problem**, **algorithms**, settings, name, userData)
changeChainSettings(**id**, **settings**)
renameChain(**id**, **newName**)
runChain(**id**)
deleteChain(**id**)
addCompStudy(**problems**, **chains**, nRuns, metrics, settings, name, userData)
changeCompStudySettings(**id**, **settings**)
renameCompStudy(**id**, **newName**)
runCompStudy(**id**)
deleteCompStudy(**id**)

4.1.2 Optimization Execution

The creation of a *task* that exploits the NSGA-II algorithm to optimize the `'MOPOL'` problem (Poloni's study [5]) is shown in Listing 4.1. The listing also shows how the task is run, and its results can be displayed by method `displayResults`. The visualization of results is described in Section 4.2. Note that the `'MO'` in the problem's name is used to distinguish between multi-objective and single-objective problems. The string `'MOPOL'` is the exact name of a MATLAB file that defines the optimization problem.

Listing 4.1: Minimal working example of optimization task in FOPS.

```
1 fops = FOPS(); % initialization of FOPS
2 fops.addTask('MOPOL', 'NSGA-II')
3 fops.runTask()
4 fops.displayResults(1, 1)
```

More examples of use can be seen in the documentation of FOPS [MM4].

4.2 Visualisation of Results

The content of the results suite is different for the results of tasks and chains, and the results of the comparative study. Therefore, this section is divided into two subsections — *Results of Tasks and Chains* and *Results of Comparative Studies*.

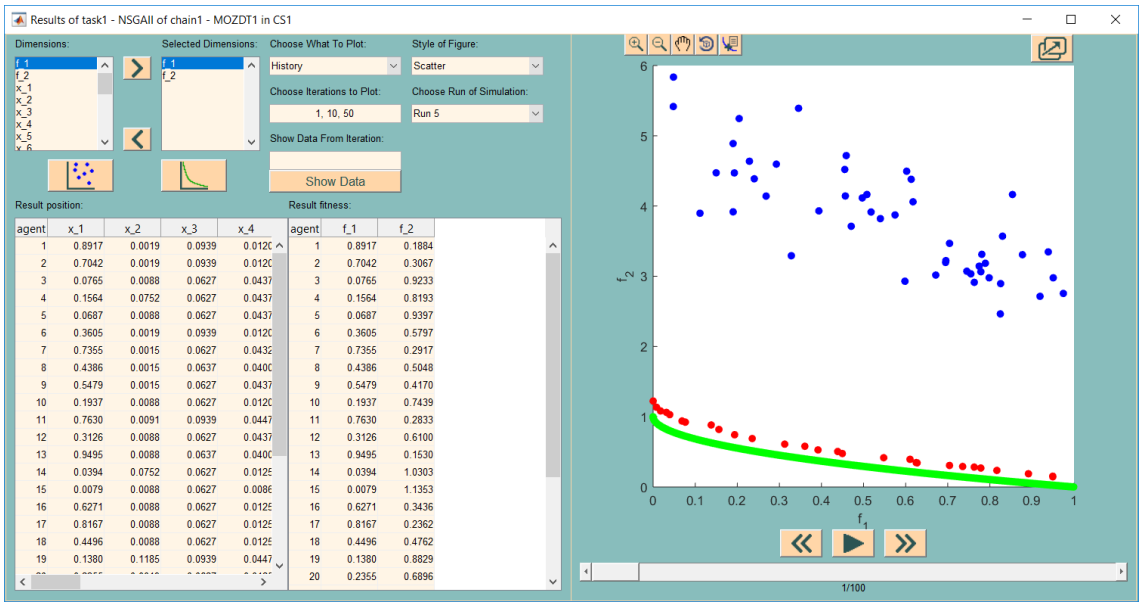


Figure 4.1: Results suite of FOPS on the left side, and the animation of the fitness values of MOZDT1 problem on the right.

4.2.1 Results of Tasks and Chains

Figure 4.1 shows the results suite when results of *task* or *chain* are visualized. Tables in the lower half of the figure contain positions and fitness values of the non-dominated set. The upper half of the figure shows a control panel. There is an animation of the fitness values of a custom optimization task presented on the right side of the figure. Green points depict the true Pareto-front, red points build the non-dominated set, and blue points depict the fitness values from consecutive iterations.

The control panel is dominated by a selection of dimensions to the plot. It can be seen that the dimensions of fitness values (`'f_1'` or `'f_2'`) can be chosen as well as the dimensions of position vectors (`'x_1'`, `'x_2'`, etc.). The `Choose What To Plot` popup menu allows the user to choose among the results (i.e. found Pareto-set), history (i.e. solutions over iteration), or metric convergence plot (e.g. generational distance over iterations).

It is also possible to search through individual iterations using the slider below the results plot and adjacent push-buttons. The results visualization tool has many capabilities that can be further examined in the FOPS documentation [MM4].

4.2.2 Results of Comparative Study

A *comparative study* with three unique chains and two different problems — `'MOZDT1'` and `'MOFON'` is shown in Figure 4.2. All combinations of chains and problems create six

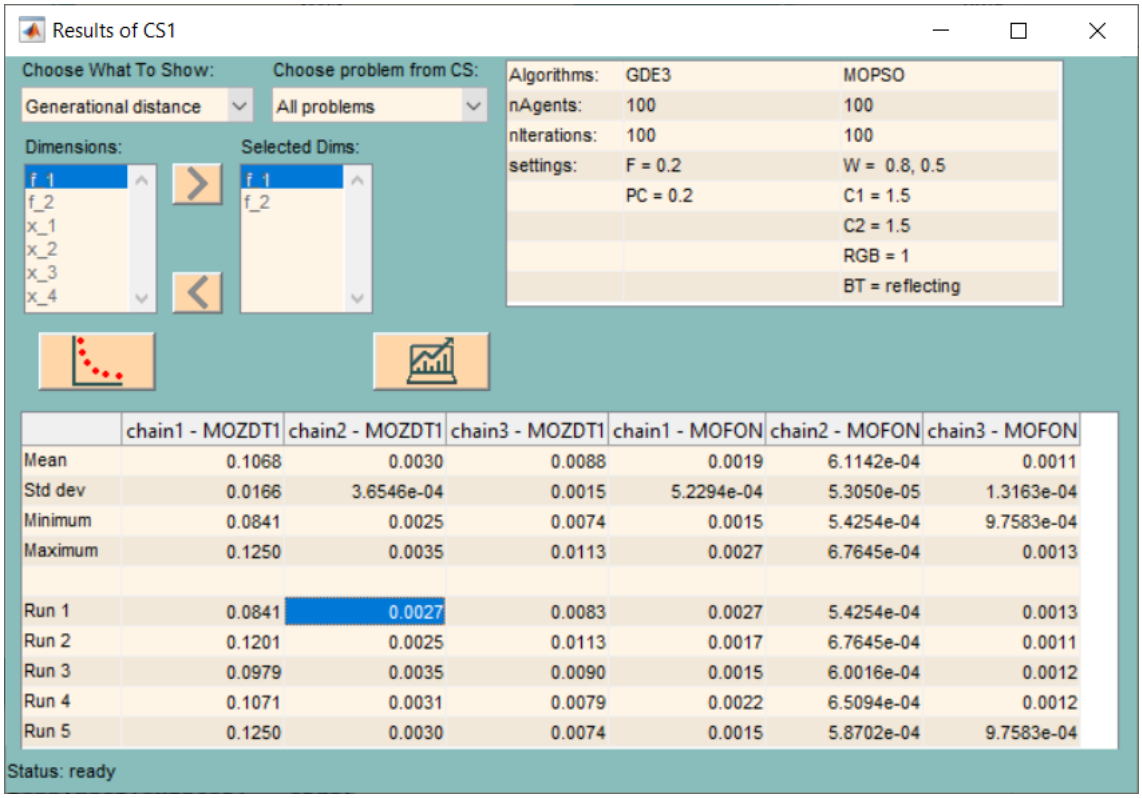


Figure 4.2: Results of the comparative study in FOPS GUI.

columns in the table in the lower part of the figure. The table shows the computational time of the optimization process for each repetition of each *chain*. The first four rows in the table are dedicated to the mean value, the standard deviation, the minimal value, and the maximal value. The type of metric is chosen by the popup menu in the top-left part of the results figure. The top-right part is dedicated to a hint panel — each time some cell in the lower table is selected, properties of the corresponding *chain* are displayed in the hint panel for a better orientation in the *comparative study* results.

4.2.3 Statistics

The Figure 4.2 also shows a push-button (bellow the bottom left corner of the hint panel), which launches the statistics figure (see Figure 4.3).

This feature allows the user to calculate easily the statistics described in Section 2.4. Currently, it covers Friedman's, Wilcoxon's, and Multiple Sign non-parametrical tests. The type of non-parametrical test is chosen in the first popup menu. The second popup menu chooses the level of significance α, and the third popup menu selects the control algorithm used in Friedman's test and Multiple Sign test.

Calculated p-values are shown in the bottom left table. There are four options:

- If the p-value is below the level of significance α (the control algorithm is better then the algorithm under test), the corresponding cell has a green background color.
- If the p-value is above α (the control algorithm performed worse than the algorithm under test), the cell background color is orange.
- If the p-value is above α, but the control algorithm may be statistically better with the use of a different α value, the background color of the cell is yellow.
- If the result is statistically insignificant (e.g. Friedman's statistic p-value is above α) or invalid, the background color of the cell is red.

4.3 Discretization Method

Parameters of the optimized system - decision variables - in a common optimization process can be any value from an interval limited by prespecified extremes. However, some parameters of the system can be restricted, somehow. In the real world, there are always some manufacturing precision limits, only a limited number of components available in stock, etc. Therefore, a given parameter is discrete, and it is not possible to change it continuously. For example, only several dielectric substrates meet design requirements, or capacitors are available in manufacturing series. Therefore, the permittivity and capacitance used as a decision variable should be discrete.

The naïve approach is to let the optimization run as if all decision variables were continuous and pick final values closest to a set of feasible, discrete values afterward. However, an optimization algorithm computes with all the possible values. Therefore,

puts an effort to find the best solution with unsuitable values that the user will discard anyway.

A smart approach is to use an optimization method that can work with discrete decision variables that, in other words, do not waste computational resources in operating with unfeasible solutions.

Most of the optimization algorithms work with continuous decision variables, but a well known genetic algorithm [2] (and most of its modifications) works with discontinuous decision variables. Therefore, it uses discrete decision variables by its nature. The use of continuous decision variables in genetic algorithms is limited by the binary precision of discrete variables.

This suggests that when there is a need to use discrete decision variables, only a limited number of optimization algorithms can be exploited. However, the discretization method allows every optimization algorithm to work with discrete decision variables.

The validity of the method is verified by the comparison of NSGA-II [8], MOPSO [9], and GDE3 [10] algorithms' performance on several well-known test problems with various discrete decision variables settings. All exploited algorithms are multi-objective, but the discretization method works with single-objective optimization methods as well.

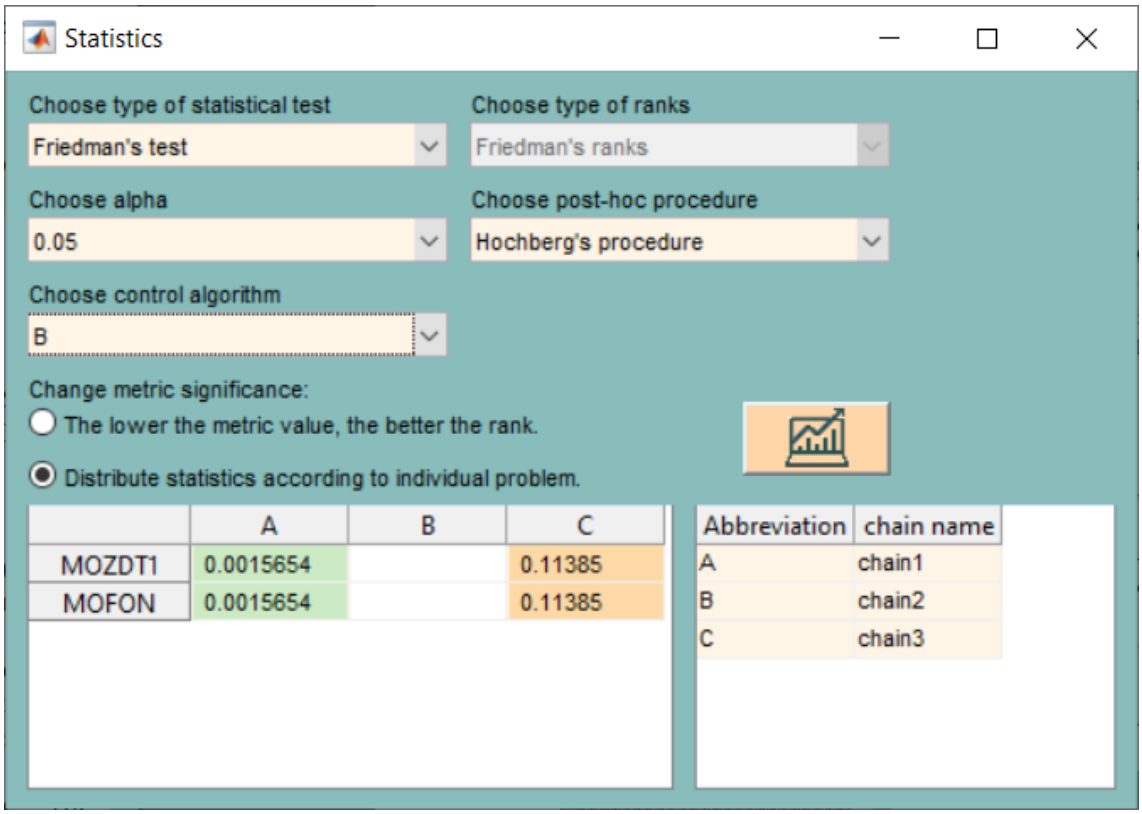


Figure 4.3: Statistics figure on the comparative study from Figure 4.2.

4.3.1 Discretization Method

Genetic algorithm - an algorithm that works with discrete decision variables by nature - describes the position of an individual in generation by a set of binary values. The density of discrete decision variable samples is given by binary precision, i.e. the number of binary values that represent the given decision variable. The position of the agent is varied by inverting randomly picked binary places of the position vector (crossover and mutation operation).

On the other hand, algorithms with continuous decision variables usually vary the position vector by summing it with another randomly picked position vector (cf. Equation (3.1), Algorithm 6.1). Therefore, real numbered positions are combined and create a new real numbered position. The presented discretization method lets an algorithm to generate the new agent's position on its own and then forces the decision variable to take one value from the discrete decision values set.

The simplest solution to find a corresponding discrete value for a given real value would be to find the closest one in an absolute measure. But if discrete samples are non-homogeneously spaced between decision variables limits, then some discrete value would occur more often than the other.

The range of the decision variable is divided by the number of discrete samples N (4.1). Then the floored difference between the real decision variable and the lower limit divided by discrete samples step δ gives us the index of discrete sample corresponding to a real value I.

$$\delta = \frac{x_{\max} - x_{\min}}{N}, \tag{4.1}$$

where $x_{\max}$ and $x_{\min}$ are the limits of a decision variable and N is the number of discrete samples of the decision variable. The index of the corresponding discrete value is defined by:

$$I = \left\lfloor \frac{x - x_{\min}}{\delta} \right\rfloor, \tag{4.2}$$

where $\lfloor \cdot \rfloor$ defines the floor function and x is the real-numbered position.

4.3.2 Test Problems

The comparative study performed to verify the discretization method exploited 7 testing problems: DLTZ4, DTLZ6, DTLZ7, ZDT1, ZDT2, ZDT6, and Poloni's problem [81, 5].

ZDT and DTLZ families of testing problems were chosen for their convenient location of the true optimal set. In ZDT problems, the solution is optimal if all decision variables are zeros except the first variable, which utilizes true Pareto-front. DTLZ problems are similar, except that the first and also the second decision variables utilize the true Pareto-front and in the case of DTLZ4, remaining decision variables (except the first and the second) have to be 0.5. Therefore, such values are attainable even with roughly discretized decision space. On the other hand, Poloni's test problem has a more complicated true optimal set (can be seen in [5, Chapter 8]), and the discretization of decision space might prevent the

optimization algorithm from locating it. All ZDT and DTLZ problems had 10 decision variables during the tests. Poloni's test problem has only two decision variables.

4.3.3 Results

The discretization method was verified with a comparative study. There were 3 optimization algorithms – NSGA-II, MOPSO, and GDE3 – exploited on 7 testing problems. Also, there were four different settings of discrete decision variables. The first setting was continuous decision variables. The remaining settings sampled each decision variable to 5001, 501, and 51 discrete samples, respectively.

Every simulation was 100 times repeated in order to obtain independent realizations of stochastic processes. All three algorithms processed 40 agents over 100 iterations. The setting of other controlling parameters can be found in Table 4.4.

Table 4.5 contains average values of generational distances (see Subsection 2.3.1) obtained in a comparative study multiplied by 1000.

Each problem was optimized with four different discretization settings. It is obvious, that the lower the density of samples is, the easier is to find the true Pareto-front. Therefore, the lower the generational distance value should be. The fact that generational

Table 4.4: Controlling parameters of algorithms

NSGA-II		**MOPSO**	
Probability of crossover	$P_C = 0.9$	Inertia weight	$W \in [0.8, 0.5]$
Probability of mutation	$P_M = 0.7$	Cognitive learning factor	$c_1 = 1.5$
Binary precision*	$\mathbf{BP} = 20$	Social learning factor	$c_2 = 1.5$
GDE3		Boundary type	reflecting
Probability of crossover	$P_C = 0.2$		
Scaling factor	$F = 0.2$		

* for each decision variable if continuous space is used.

Table 4.5: Thousandfold the generational distance with various discretization scenarios.

Algorithm	N	**ZDT1**	**ZDT2**	**ZDT6**	**DTLZ4**	**DTLZ6**	**DTLZ7**	**MOPOL**
NSGA-II	2^{20}	21.45	29.49	1919.04	56.58	4227.23	70.43	54.88
	5001	14.98	20.58	1619.15	96.92	2768.14	48.92	73.72
	501	8.75	9.25	683.47	55.20	879.09	32.60	51.52
	51	0.23	0.05	0.05	12.59	3.93	12.58	57.92
MOPSO	∞	17.13	10.15	5465.71	245.31	5226.11	70.86	12.47
	5001	12.31	5.01	385.78	164.69	2073.57	35.27	19.35
	501	8.93	4.06	100.77	106.91	1632.72	35.84	20.52
	51	13.71	4.98	151.23	17.97	845.12	75.06	26.21
GDE3	∞	2.42	1.56	24.58	62.73	72.83	23.77	42.71
	5001	2.30	2.29	72.55	62.86	177.89	22.58	52.48
	501	0.35	1.32	0.13	63.79	116.69	21.34	51.27
	51	0.03	0.04	0.05	0.69	22.60	11.18	63.79

distance values produced by MOPSO and GDE3 algorithms decrease with decreasing density of discrete samples verifies the functionality of the used discretization method. The same pattern is visible in most cases in Table 4.5. However, there are several exceptions.

Most of the exceptions occur with the use of the MOPSO algorithm and only 50 samples per decision variable. The reason is hidden in the behavior of the MOPSO algorithm rather than in the discretization method itself. MOPSO uses an external archive of found non-dominated solutions. The majority of continuous non-dominated solutions are dominated due to discretization, therefore an external archive holds only a few entries. This paralyzes the exploration capabilities of the algorithm because external archive members are used as x_{gbest} in (3.1).

Poloni's problem also shows a few exceptions. However, in this case, it is caused by an unsuitably located true Pareto-front (see in [5, Chapter 8]). Therefore, the discretization of decision space prevents the algorithm from reaching the optimum.

4.3.4 Conclusion

If the binary precision of decision variables in the binary-coded NSGA-II algorithm decreases, the number of possible positions in decision space also decreases. Therefore, it is easier for the algorithm to find the optimum.

The discretization method enables the optimization algorithm to work with continuous decision variables as usual. However, the decision vector is discretized right after the agent's position update. The results obtained in the comparative study prove that such a technique has a similar effect on the difficulty of an optimization problem. In other words, the overall decision space shrinks.

4.4 Surrogate Optimization

The fitness functions can have various forms. If the fitness function is expressed as a closed-form formula, it can be computed almost immediately. However, the fitness functions in real-world optimization problems can take considerably more time to compute. A common assumption is that the computation of the fitness values is the most time demanding operation during the optimization process.

An example of such a complex optimization can be the synthesis of the cavity resonator structure used in [87]. An optimization algorithm generates decision variables (e.g. design dimensions), and the calculation of the fitness values involves a full-wave simulation of the designed structure with dimensions determined by the optimization algorithm.

Since evolutionary algorithms generally require a large number of fitness function evaluations during the optimization process, and each evaluation can take a significant amount of time, it is desirable to skip some unnecessary fitness function evaluations. The tolerance-based surrogate method allows an optimization algorithm to skip some fitness function evaluations. The question is, which evaluations can be skipped?

If an electromagnetic structure design is considered, there are always some manufacturing precision limits. Therefore, it is useless to evaluate the fitness values for dimensions (decision variables) that differ at e.g. the sixth decimal place. Moreover, the fitness values of such similar dimensions would most likely be very similar too, and an overall contribution to the optimization process would be minimal.

This is the essential idea of the tolerance-based surrogate method. At the beginning of the optimization run, no fitness values are known, and all the fitness function has to be evaluated. Each evaluated solution (i.e. decision variables and corresponding fitness values) is stored in the archive. At some point in the optimization process, an algorithm converges close to the optimum (i.e. the true Pareto-front in MOOP), and new solutions with yet unknown fitness values begin to be similar (or equal) to some members of the archive. Evaluation of the fitness functions of such a solution has a negligible contribution to the optimization process. Therefore, the fitness functions are not evaluated. The fitness values of the closest solution in the archive are taken from the archive instead.

The tolerance vector defines how close the new solution from a member of the archive has to be. The tolerance vector has the same number of elements as the decision vector of the problem. Each time the differences between all the decision variables of some member of the archive and the new solution are lower than the vector of tolerances, the fitness function evaluation is skipped.

Figure 4.4 further clarifies the tolerance-based surrogate method. It depicts a decision space of a simple two-objective optimization problem and several solutions stored in the archive. The grid denotes the limits of the decision variables, i.e. $x_1 \in [0.1, 1]$ and $x_2 \in [0, 1]$. The fitness functions are defined as follows:

$$f_1(\mathbf{x}) = x_1, \tag{4.3}$$

$$f_2(\mathbf{x}) = \frac{1 + x_2}{x_1}. \tag{4.4}$$

The thick line (bottom edge of the grid) marks the true Pareto-front of the problem in the decision space. There are 15 solutions stored in the archive (their positions are marked with the thick "+" signs). The tolerance vector was $\{0.05, 0.1\}$ and areas within the tolerance are depicted with the hatched boxes around solutions. Five of the solutions are indexed from **1** to **5**.

The solution **1** is the true Pareto-optimal solution and its fitness values are $\{0.1,\ 10\}$. The solution **2** has the fitness values $\{0.3,\ 3.67\}$. The solution **3** is not far from optimality (see that the tolerance box covers a part of the true Pareto-front) and its fitness values are $\{0.5,\ 2.15\}$. The solution **4** has the fitness values $\{0.7,\ 1.643\}$ and the solution **5** has the fitness values $\{1,\ 1\}$ (also the true Pareto-optimal solution). All the indexed solutions are non-dominated (in the objective space).

If a newly generated solution has the position e.g. $\{0.94,\ 0\}$ (marked with the $\times$ signs in Figure 4.4), it will fall in the tolerance area of the solution **5** ($\{1,\ 0\}$) and even if its fitness values according to (4.3) and (4.4) should be $\{0.94,\ 1.064\}$, the fitness values

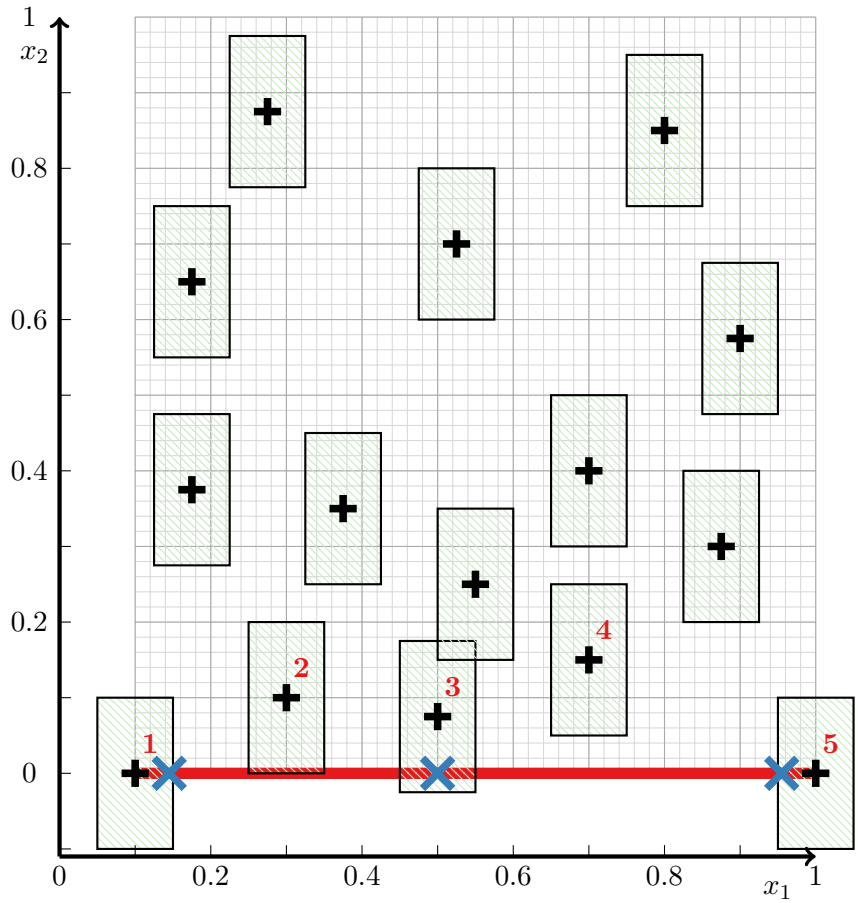


Figure 4.4: Decision space of a two-objective problem with solutions stored in the archive.

$\{1,\ 1\}$ of the solution **5** will be assigned to it. The difference in fitness values caused by tolerance-based surrogate method is relatively small in this case.

Another generated solution has the position e.g. $\{0.14,\ 0\}$ (marked with the $\times$ sign in Figure 4.4). Such solution will fall in the tolerance area of the solution **1** ($\{0.1,\ 0\}$) and even if its fitness values according to (4.3) and (4.4) should be $\{0.1,\ 7.143\}$, the fitness values $\{0.1,\ 10\}$ of the solution **1** will be assigned to it. The difference between the true fitness values and the surrogate fitness values is rather large here, although the absolute distance between the archive member and the generated solution in the decision space is identical as in the previous pair. This suggests that the setting of the tolerance vector can be sometimes a difficult task.

The solution **3** in Figure 4.4 indicates the main drawback of the tolerance-based surrogate method. The border of the hatched box of this solution lies on the true Pareto-front, but the solution itself is rather far away ($\{0.5,\ 0.075\}$). Therefore, if a new solution is generated within the hatched box, e.g. $\{0.5,\ 0\}$ (marked with the $\times$ sign), then the fitness values of the known solution are assigned to it. But the fitness values of the true Pareto-optimal solution with the position $\{0.5,\ 0\}$ according to equations (4.3) and (4.4) are $\{0.5,\ 2\}$, while the fitness values of the solution **3** from the archive are $\{0.5,\ 3.5\}$. Af-

terward, the solution $\{0.5,\ 0\}$ (which is, as we know, better then the archive member) will be supressed in the optimization process due to its downgraded fitness values. Therefore, that part of the true Pareto-front under the solution **3** in Figure 4.4 is inaccessible due to the tolerance-based surrogate method if too large values are used in the tolerance vector.

There exists no methodology to estimate the proper tolerance vector. The tolerance vector depends on an optimized problem and the user's knowledge about the problem. Therefore, the tolerance-based surrogate method can make the parts of the true Pareto-front inaccessible, and therefore introduces uncertainty into the optimization process. However, it is balanced by the reduction of the number of fitness evaluations, i.e. the overall cost of optimization. In other words, the setting of the tolerance vector is a trade-off between the time-saving properties and the inaccessible area that might occur around the true global optimum. Note that an optimization algorithm can still reach any point within the decision space. It can not reach only a close neighborhood of the archive members.

The drawback can be suppressed with the use of a discrete decision space. When the decision space is discrete, the tolerance vector can be set to almost zero values. Therefore, the new solution can be either identical or differ by the whole step of the discrete decision variable. If the new solution generated by the optimization algorithm already exists in the archive, it is not calculated again. In this scenario, some regions of the decision space are inaccessible for the optimization algorithm.

4.4.1 Performance Assessment

The Multi-objective Particle Swarm Optimization (MOPSO) algorithm has been employed to obtain presented results. The Elitist Non-dominated Sorting Genetic Algorithm (NSGA-II) and the Third Generalized Differential Evolution algorithm (GDE3) were also tried, but the produced results were practically similar to those from the MOPSO algorithm.

Minor differences in the results were related to algorithms' performance rather than to the tolerance-based surrogate method itself. Therefore, the results of the NSGA-II and GDE3 algorithms are not presented.

The validation of the tolerance-based surrogate method was performed on several two-objective optimization benchmark problems. However, the tolerance-based surrogate method is independent of the number of objectives. A summary of benchmark problems can be seen in Table 4.6 (MOFON stands for Fonseca and Fleming's study, MOKUR stands for Kursawe's study, MOPOL stands for Poloni's study, and MOZDT1 and MOZDT6 stands for Zitzler, Deb and Thiele's studies). The benchmark problems are further described in [5].

The performance of the tolerance-based surrogate method is tested from two points of view. The first one is the computational time required to perform a particular simulation run. For a better insight into the time-saving property of the tolerance-based surrogate method, Table 4.8 contains an average count of fitness values obtained by our

Table 4.6: Summary of used benchmark problems.

Problem	MOFON	MOKUR	MOPOL	MOZDT1	MOZDT4
Number of decision variables	3	3	2	30	10
Limits of decision variables	$[-4, 4]$	$[-5, 5]$	$[-\pi, \pi]$	$[0, 1]$	$[-5, 5]^*$

* Limits of the first decision variable are $[0, 1]$.

surrogate method. The second point of view is the value of a generational distance (see Subsection 2.3.1).

4.4.2 Results

Controlling parameters of the MOPSO algorithm were set as follows: the inertia weight w was linearly decreased from 0.6 to 0.4 over all iterations, the cognitive learning factor was $c_1 = 1.5$, and the social learning factor was $c_2 = 1$.

There were 100 agents in each simulation run over 100 iterations. Therefore, the fitness function would be evaluated 10 000-times if the tolerance-based surrogate method was disabled. All values presented in Tables 4.7–4.10 are an average of 100 repetitions.

An evaluation of the fitness function in case of the benchmark problems is almost immediate. Therefore, the usage of the tolerance-based surrogate method would have no benefit. Nevertheless, there were delays inserted to the fitness functions. The delays were 0, 1, and 10 milliseconds. Due to the nested delays, all evaluations of the fitness functions alone took 0, 10, and 100 seconds, respectively, for each simulation run if the tolerance-based surrogate method was disabled.

The tolerance vector was defined as a fraction of the range of the problem's decision variables, i.e. 0, 0.001, 0.01, 0.05, and 0.1 times the range of the decision variable. The first one means that the range of each decision variable is "divided" into an infinite number of sections. In other words, the tolerance-based surrogate method is disabled. The last one means that the range of each decision variable is "divided" into 10 sections. The quotation marks refer to the fact that the tolerance-based surrogate method has nothing to do with the discretization of the decision variables. The tolerance-based surrogate method only takes positions of two randomly generated solutions and checks whether its difference is lower than the tolerance or not.

Table 4.7 contains an average computational time of particular simulations. It is obvious from the first three lines (where the tolerance-based surrogate method is disabled) that the computational time of an optimization method alone is almost independent of a problem. It is also evident how the nested delays affect the computational times. The higher the tolerance is, the larger the time saving is. Contrarily, the computational time when the tolerance-based surrogate method is enabled depends on the problem.

On several occasions, the computational time is larger when the tolerance-based surrogate method is enabled, compared to the computational times with the tolerance vector elements set to zero. The most obvious items are the ones where no delay and only small

Table 4.7: An average computational time in seconds.

Delay [ms]	Tolerance	MOFON	MOKUR	MOPOL	MOZDT1	MOZDT4
0	0	2.95	2.86	2.62	2.74	2.68
1	0	13.66	13.75	12.82	13.36	13.13
10	0	103.42	103.11	103.09	103.10	103.10
0	0.001	5.07	5.26	4.53	16.61	5.18
1	0.001	10.94	15.36	11.14	27.76	13.21
10	0.001	58.19	100.36	65.50	117.85	78.53
0	0.01	3.79	4.13	3.79	17.09	5.02
1	0.01	5.40	6.63	6.14	27.84	11.00
10	0.01	17.97	24.97	22.78	116.15	54.68
0	0.05	2.98	3.16	2.95	14.78	4.60
1	0.05	3.48	3.76	3.26	21.69	8.04
10	0.05	7.37	8.47	5.39	74.52	34.11
0	0.1	2.81	2.91	2.70	6.00	4.30
1	0.1	3.04	3.20	2.83	10.99	6.60
10	0.1	4.81	5.24	3.84	45.97	23.53

tolerance values were used. This behavior is caused by a number of comparison operations required by the tolerance-based surrogate method. If the tolerance is small and no surrogate fitness values are found in the archive (MOKUR and MOZDT1 problems, see Table 4.8), the number of the comparison operations quickly increases during the optimization. It slows the entire process.

Especially, in the case of the MOZDT1 problem and the tolerance vector elements set to 0.001, no surrogate solutions were found (see Table 4.8). This problem has 30 decision variables. Therefore, the probability that a new solution is within the tolerance of some solution stored in the archive is lower compared to other problems. The number of the comparison operations quickly grows from 100×30 after the first iteration to 10000×30 after the last iteration. Therefore, the overall deceleration is almost 13 seconds.

Table 4.8 shows, how many fitness values were taken from the archive during each simulation run. The content of Table 4.8 correlates with the content of Table 4.7. The first three lines of the table were omitted because they contain zeros if the tolerance-based surrogate method is disabled. With increasing tolerance values, the number of surrogate solutions also increases because there is a higher probability of finding a close enough solution in the archive.

The differences between values where the same tolerance is used are caused mainly by the number of decision variables. If a problem has only three decision variables (MOFON), it is easier to randomly generate a solution that is close to the member of the archive, compared to the problem with 30 decision variables (MOZDT1).

Secondly, the differences are given by the speed of convergence to an optimum. If agents rapidly approach global optimum, then new solutions are almost similar to the previous ones. Therefore, surrogate solutions can be found in the archive (MOFON).

Table 4.8: An average number of surrogate solutions.

Delay [ms]	Tolerance	MOFON	MOKUR	MOPOL	MOZDT1	MOZDT4
0	0.001	4777	597	4024	0	2870
1	0.001	4812	596	4045	0	2755
10	0.001	4775	587	3992	0	2751
0	0.01	8620	7991	8177	230	4788
1	0.01	8626	7976	8171	228	4801
10	0.01	8624	7980	8170	199	5117
0	0.05	9576	9488	9778	4014	7080
1	0.05	9575	9497	9777	3970	7160
10	0.05	9574	9493	9775	4101	7135
0	0.1	9805	9774	9889	6159	8192
1	0.1	9806	9775	9889	6148	8201
10	0.1	9806	9779	9889	6128	8152

Table 4.9: An average generational distance - real-coded decision space.

Delay [ms]	Tolerance	MOFON	MOKUR	MOPOL	MOZDT1	MOZDT4
0	0	<0.001	0.032	0.011	0.247	20.822
1	0	<0.001	0.032	0.012	0.232	20.649
10	0	<0.001	0.032	0.013	0.221	19.965
0	0.001	<0.001	0.033	0.012	0.234	23.354
1	0.001	<0.001	0.031	0.012	0.245	22.911
10	0.001	<0.001	0.034	0.010	0.225	23.507
0	0.01	0.005	0.110	0.037	0.226	46.445
1	0.01	0.005	0.109	0.036	0.238	49.949
10	0.01	0.005	0.111	0.039	0.232	47.377
0	0.05	0.051	0.893	0.322	0.261	61.551
1	0.05	0.052	0.869	0.306	0.262	60.981
10	0.05	0.052	0.888	0.313	0.259	60.895
0	0.1	0.129	2.153	0.903	0.337	70.144
1	0.1	0.130	2.068	0.929	0.334	68.461
10	0.1	0.123	2.064	0.965	0.335	68.306

Contrarily, if agents approach the optimum slowly, the positions are continuously drawn to optimum (MOKUR), and the surrogate solutions cannot be found in the archive.

When the tolerances are increased, the generational distance also increases due to the drawback of the tolerance-based surrogate method. Differences between values of the generational distance with the same tolerance are caused by the difficulty of the problem. The MOFON problem is a relatively simple one. On the other hand, the MOZDT4 problem with only 10 decision variables (in comparison with the MOZDT1 problem) has many local optima, where an algorithm can be caught. Therefore, the values of the generational distance are large.

Table 4.10 contains generational distance values of simulations when a discrete decision

Table 4.10: An average generational distance - discrete decision space.

Delay [ms]	Tolerance	MOFON	MOKUR	MOPOL	MOZDT1	MOZDT4
0	0	<0.001	0.032	0.011	0.235	21.542
1	0	<0.001	0.033	0.010	0.231	19.827
10	0	<0.001	0.032	0.014	0.242	17.982
0	0.001	<0.001	0.030	0.011	0.110	11.954
1	0.001	<0.001	0.030	0.011	0.111	14.233
10	0.001	<0.001	0.030	0.011	0.129	12.823
0	0.01	0.001	0.057	0.027	0.055	11.033
1	0.01	0.001	0.058	0.027	0.043	11.574
10	0.01	0.001	0.057	0.027	0.041	12.019
0	0.05	0.026	0.219	0.277	0.041	0.145
1	0.05	0.026	0.219	0.311	0.045	0.146
10	0.05	0.026	0.219	0.311	0.038	0.146
0	0.1	0.102	0.017	0.288	0.074	0.164
1	0.1	0.102	0.017	0.288	0.046	0.167
10	0.1	0.102	0.017	0.288	0.063	0.160

space is used. Note that the second column in Table 4.10 is now called Fraction. In this case, the tolerances were set to very low values (1e−6). However, the discretization of the decision variables corresponds with the tolerances from the simulation with a real-coded decision space. Therefore, the fraction of 0.1 denotes that each decision variable was sampled to 11 points. When the discrete decision space is used, the surrogate is found only if a new solution is identical to a member of the archive. Otherwise, the fitness functions have to be evaluated.

Some problems (MOPOL and MOFON) in Table 4.10 show that if the fraction value is increased, meaning that the decision variable is sampled more sparsely, the generational distance downgrades. This is caused by the fact that the true Pareto-optimal solutions do not correspond with the sampling of the decision variables.

The true Pareto-optimal set of MOZDT problems corresponds to $x_1 \in [0, 1]$, while all the other decision variables are zero. Therefore, the discrete samples of the decision variables can match the true Pareto-optimal set.

Analogous tables to Table 4.7 and Table 4.8 with the discrete decision space are not presented because their content is similar to those with the real-coded decision space.

4.4.3 Conclusion

The advantage of the tolerance-based surrogate method is that certain fitness function evaluations can be skipped. Therefore, an overall computational time of an optimization process can be reduced. The drawback of skipping the fitness function evaluation can lead to a loss of precision. However, the precision loss may be reduced if a discrete decision space is used with an appropriate tolerance vector. It was also discussed that the tolerance-based surrogate method can even slow down the optimization process if it is improperly

used.

The real-world optimization task, where the advantages of the surrogate method are used, is the anisotropic band-stop filter design (see Section 9.1). Each fitness function evaluation of this problem takes around 10 minutes. Overall optimization time is reduced from around 2 months to approximately 2 weeks thanks to the tolerance-based surrogate method.

Since an evaluation of fitness functions can be very time consuming, the proposed tolerance-based surrogate method can accelerate the whole optimization process even if only a few surrogate solutions are found.

The tolerance-based surrogate method can also be exploited in cases of recurrent optimization tasks, either after altering algorithm settings or the crash of a simulation. The archive of known solutions can be loaded before the beginning of the optimization process, and surrogate solutions can be used from early stages of the optimization process.

4.5 Conclusion

The main advantage of an in-house optimization toolbox for MATLAB is that the implementation of a new optimization algorithm is simple and straightforward. Within moments, the user implements the unique properties of a given algorithm. The procedures that are common for many optimization algorithms (e.g. initialization of population, non-dominated sorting, etc.) are already defined and can be utilized without effort. Afterward, the verification of a new algorithm is easily performed by predefined optimizing routines on a set of benchmark problems that are already defined too.

The framework includes a library with many optimization problems used mainly for the verification of the optimization methods. However, the main focus was set on the versatility of the framework. Therefore, it can be employed to various real-world challenges with requirements of all kinds. It is extensible – algorithm-wise, problem-wise, metric-wise, and even surrogate method-wise.

The FOPS framework, for the first time, implements optimization procedures that can solve problems with a variable number of dimensions. Supporting the problems with a Variable Number of Dimensions enables to solve challenging EM optimization problems with lower computational cost compared to the common optimization approach with a fixed number of dimensions.

Many real-life optimization problems have discrete decision space, but only a few evolutionary algorithms can naturally work with discrete decision space. FOPS toolbox contains discretization method, which is universal for every optimization algorithm. Comparison of implemented method with a naturally discrete optimization algorithm – NSGA-II – is presented in [63].

Evolutionary Algorithms process numerous agents through decision space over many iterations. Therefore, a great number of fitness function evaluations are usually needed. It is a common assumption that fitness function evaluation is a time-consuming procedure.

Therefore, it is advantageous to keep the number of fitness evaluations at a minimum. FOPS optimizer includes a simple surrogate method that enables the optimizer to skip some fitness function evaluations. Method was presented in [64].

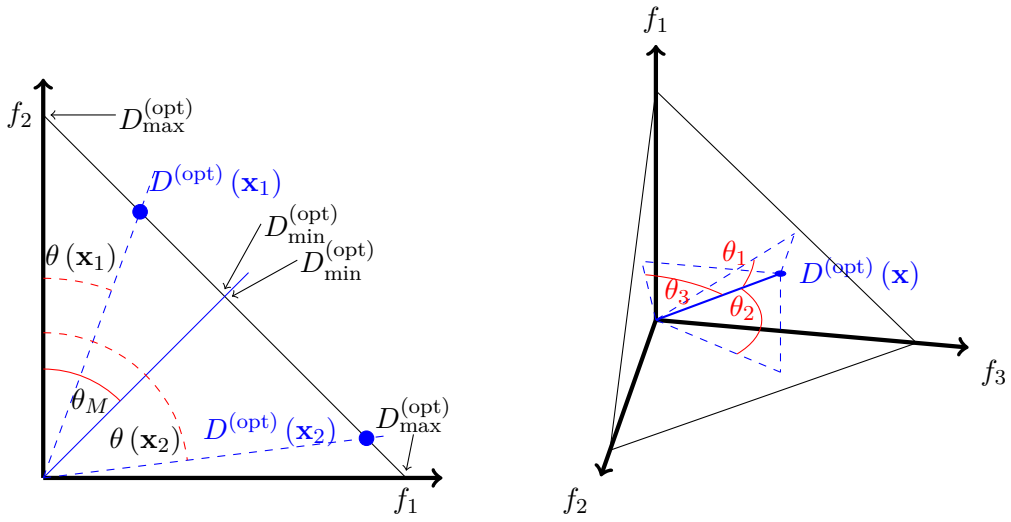
The versatility of the framework is demonstrated in Chapter 9. Various features of the FOPS framework are exploited there. It was utilized in many research papers [MM7, 63, 64, 62, MM8, MM9, MM1, MM10, MM11, MM12, MM13, MM14].

5 MULTI-OBJECTIVE TESTING PROBLEMS WITH VARIABLE NUMBER OF DIMENSIONS

When comparing the performance of multiple optimization algorithms, it is convenient to use testing problems with known true Pareto-front. Therefore, the non-dominated sets can be compared and qualified.

Single-objective VND benchmark problems were proposed in [88] or [45]. Before these publications, most of the studies were validated by arbitrary and often simplified real-world problems. Speaking of a multi-objective problems with a variable number of dimensions, no such library of benchmark problems can be found in the open literature (to the authors' knowledge).

This subsection proposes the methodology for creating multi-objective VND benchmark problems based on the idea of [86]. Authors of that paper say that different parts of the Pareto-front may have different sizes in real-world optimization problems. Afterward, they constructed a few problems with linearly-shaped Pareto-fronts where the number of decision variables of the optimal solution is determined by the angle between the line connecting the solution with the coordinate's origin and the f_2 axis (2-dimensional Pareto-fronts) or individual coordinate planes (3-dimensional Pareto-fronts).



(a) Two-dimensional objective space. **(b)** Three-dimensional objective space.

Figure 5.1: Construction of Pareto-fronts of general two- and three-objective VND problems.

The dimensionality of an arbitrary solution on the Pareto-front is determined by the following process:

$$D^{(\mathrm{opt})}(\mathbf{x}) = D_j^{(\mathrm{opt})}, \tag{5.1}$$

where the dimensionality is selected from the list $D^{(\mathrm{opt})} = \left\{D_{\min}^{(\mathrm{opt})}, \ldots, D_{\max}^{(\mathrm{opt})}\right\}$ with N_D

members and the index is defined by:

$$j = \begin{cases} 1 + \left\lfloor \left(1 - \frac{\theta(\mathbf{x})}{\theta_M}\right)(N_D - 1) \right\rfloor, \text{for} \quad \theta(\mathbf{x}) <= \theta_M \\ \\ 1 + \left\lfloor \quad \left(\frac{\theta(\mathbf{x})}{\theta_M}\right)(N_D - 1) \right\rfloor, \text{for} \quad \theta(\mathbf{x}) > \theta_M, \end{cases} \tag{5.2}$$

where θ_M is the maximal angle shown in Figure 5.1 and is set to 45° in our study. Pareto-fronts are divided into two regions, where the dimensionality of the solution gradually decreases in the first region and gradually increases in the second region. The angle $\theta(\mathbf{x})$ for two-dimensional Pareto-fronts is defined as:

$$\theta(\mathbf{x}) = \arccos\left(\frac{f_2}{\sqrt{f_1^2 + f_2^2}}\right) \tag{5.3}$$

and for three-dimensional Pareto-fronts:

$$\theta(\mathbf{x}) = \arccos\left[\max_{i=\{1,2,3\}}\left(\frac{f_i}{\sqrt{f_1^2 + f_2^2 + f_3^2}}\right)\right]. \tag{5.4}$$

Note that f_1, f_2, and f_3 denote fitness values of the first, second, and third objective for vector $\mathbf{x}$, respectively.

5.1 Sample Problem – VND-MOZDT1

Here we show an example of a modified benchmark problem from the [85] test suite. The problem is scalable in the decision space, and the number of decision variables can vary within $D \in [3, 30]$. True Pareto-front dimensionalities are $D^{(\text{opt})} = \{4, 5, 6, 7\}$. All decision variables reside within interval $[0, 1]$.

Fitness functions are similar to the original version, although the second fitness functions contain the dimensionality penalization constituent $\Psi = \Psi_r \Psi_0$:

$$f_1(\mathbf{x}) = x_1, \tag{5.5}$$

$$f_2(\mathbf{x}, D) = h(\mathbf{x}, D)\left[1 - \sqrt{\frac{x_1}{h(\mathbf{x})}}\right] + 0.1\Psi_0, \tag{5.6}$$

$$h(\mathbf{x}) = 1 + 9\frac{\sum_{j=2}^{D} x_j}{D - 1}, \tag{5.7}$$

where D is the number of decision variables of the agent and $\Psi_0 = \left[D - D^{(\text{opt})}(\mathbf{x})\right]^2$. Value $D^{(\text{opt})}(\mathbf{x})$ is the optimal number of decision variables of the agent from (5.1). Note that the penalization constant for VND-MOZDT1 is $\Psi_r = 0.1$.

5.2 Sample Problem – VND-DTLZ2

Here we show an example of a modified benchmark problem from the [81] test suite. The problem is scalable in the decision space, and the number of decision variables can vary

within $D \in [3, 12]$. True Pareto-front dimensionalities are $D^{(\text{opt})} = \{3, 4, 5\}$. All decision variables reside within interval $[0, 1]$.

Fitness functions are similar to the original version, although all the fitness functions contain the dimensionality penalization constituent $\Psi = \Psi_r \Psi_0$:

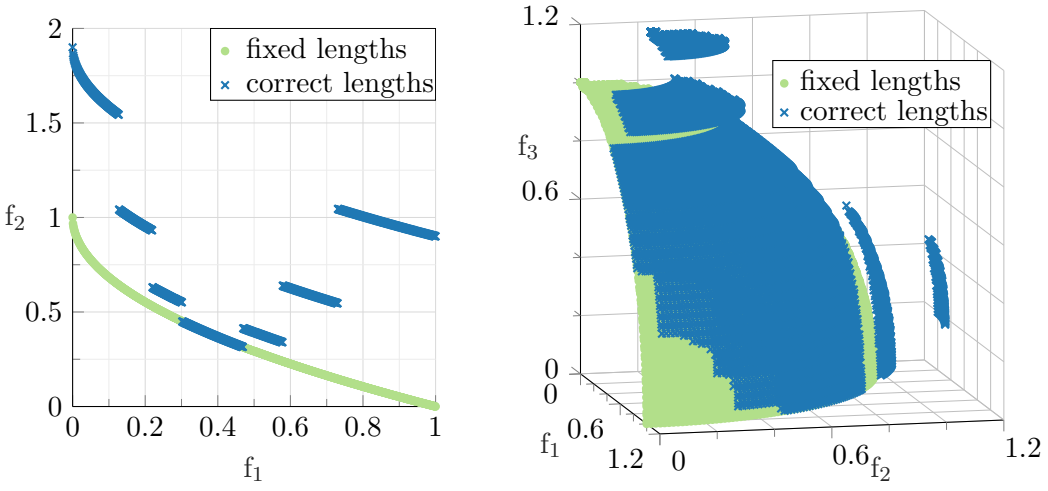
$$f_1(\mathbf{x}, D) = [1 + h(\mathbf{z}, D)] \cos(x_1 \pi/2) \cos(x_2 \pi/2) + 0.05\Psi_0, \tag{5.8}$$

$$f_2(\mathbf{x}, D) = [1 + h(\mathbf{z}, D)] \cos(x_1 \pi/2) \sin(x_2 \pi/2) + 0.05\Psi_0, \tag{5.9}$$

$$f_3(\mathbf{x}, D) = [1 + h(\mathbf{z}, D)] \sin(x_1 \pi/2) + 0.05\Psi_0, \tag{5.10}$$

$$h(\mathbf{z}, D) = \sum_{j=M}^{D} (x_j - 0.5)^2, \tag{5.11}$$

where M is the number of objectives and $\Psi_0 = \left[D(\mathbf{x}) - D^{(\text{opt})}(\mathbf{x})\right]^2$. Value $D^{(\text{opt})}(\mathbf{x})$ is optimal dimension of a solution from equation (5.1). Note that the penalization constant for VND-MODTLZ2 is $\Psi_r = 0.5$.



(a) Modified ZDT1. **(b)** Modified DTLZ2.

Figure 5.2: An example of Pareto-fronts of modified ZDT1 and DTLZ2 problems. Decision vectors of correct values but fixed lengths are marked by "•" signs, and decision vectors of correct values and correct lengths are marked by "×" signs.

Figure 5.2 shows two- and three-dimensional Pareto-fronts. Solutions with correct decision vector values and also the correct number of decision variables are represented by "×" signs. Solutions with correct decision vector values, but the number of components of all decision vectors is fixed ($D = 7$ for VND-MOZDT1 problem and $D = 5$ for VND-MODTLZ2 problem) are marked by "•" signs.

The complete definition of VND benchmark problems used in the thesis is beyond its extent. It can be found in [MM4, Section 15.4]. Table 5.1 lists the relevant properties of the benchmark problems used in the remainder of the thesis.

Table 5.1: List of modified VND problems. Note that $\Psi_0 = \left[D - D^{(\text{opt})}(\mathbf{x})\right]^2$.

Problem	M	D	$D^{(\text{opt})}$	**Penalization**	$\text{HV}_{\text{reference}}$	HV^{P^*}
VND-MODTLZ1	3	$\{3, 4, \ldots, 7\}$	$\{3, 4, 5\}$	$0.50\Psi_0$	$[11, 11, 11]$	1330.97864
VND-MODTLZ2	3	$\{3, 4, \ldots, 12\}$	$\{3, 4, 5\}$	$0.05\Psi_0$	$[11, 11, 11]$	1330.47640
VND-MODTLZ3	3	$\{3, 4, \ldots, 12\}$	$\{3, 4, 5\}$	$0.80\Psi_0$	$[11, 11, 11]$	1330.47640
VND-MODTLZ4	3	$\{3, 4, \ldots, 12\}$	$\{3, 4, 5\}$	$0.05\Psi_0$	$[11, 11, 11]$	1330.47640
VND-MODTLZ5	3	$\{3, 4, \ldots, 12\}$	$\{3, 4, 5\}$	$0.05\Psi_0$	$[11, 11, 11]$	1319.05684
VND-MODTLZ6	3	$\{3, 4, \ldots, 12\}$	$\{3, 4, 5\}$	$0.08\Psi_0$	$[11, 11, 11]$	1319.05684
VND-MODTLZ7	3	$\{3, 4, \ldots, 22\}$	$\{4, 5, 6, 7\}$	$0.30\Psi_0$	$[11, 11, 11]$	993.404326
VND-MOLZ1	2	$\{3, 4, \ldots, 10\}$	$\{3, 4, 5\}$	$0.05\Psi_0$	$[11, 11]$	120.666667
VND-MOLZ2	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.06\Psi_0$	$[11, 11]$	120.666667
VND-MOLZ3	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.06\Psi_0$	$[11, 11]$	120.666667
VND-MOLZ4	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.05\Psi_0$	$[11, 11]$	120.666667
VND-MOLZ5	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.05\Psi_0$	$[11, 11]$	120.666667
VND-MOLZ6	3	$\{5, 6, \ldots, 10\}$	$\{5, 6, 7\}$	$0.12\Psi_0$	$[11, 11, 11]$	1330.47640
VND-MOLZ7	2	$\{3, 4, \ldots, 10\}$	$\{3, 4, 5\}$	$0.10\Psi_0$	$[11, 11]$	120.666667
VND-MOLZ8	2	$\{3, 4, \ldots, 10\}$	$\{3, 4, 5\}$	$0.35\Psi_0$	$[11, 11]$	120.666667
VND-MOLZ9	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.06\Psi_0$	$[11, 11]$	120.666667
VND-MOUF4	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.05\Psi_0$	$[11, 11]$	120.333333
VND-MOUF5	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.15\Psi_0$	$[11, 11]$	120.475000
VND-MOUF6	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.50\Psi_0$	$[11, 11]$	120.437487
VND-MOUF7	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.15\Psi_0$	$[11, 11]$	120.499761
VND-MOUF9	3	$\{5, 6, \ldots, 30\}$	$\{5, 6, 7, 8\}$	$0.25\Psi_0$	$[11, 11, 11]$	1330.69911
VND-MOUF10	3	$\{5, 6, \ldots, 30\}$	$\{5, 6, 7, 8\}$	$0.50\Psi_0$	$[11, 11, 11]$	1330.47640
VND-MOZDT1	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.10\Psi_0$	$[11, 11]$	120.666667
VND-MOZDT2	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.10\Psi_0$	$[11, 11]$	120.333333
VND-MOZDT3	2	$\{3, 4, \ldots, 30\}$	$\{4, 5, 6, 7\}$	$0.10\Psi_0$	$[11, 11]$	128.778116
VND-MOZDT4	2	$\{3, 4, \ldots, 10\}$	$\{3, 4, 5\}$	$0.20\Psi_0$	$[11, 11]$	120.666667
VND-MOZDT6	2	$\{3, 4, \ldots, 10\}$	$\{3, 4, 5\}$	$0.10\Psi_0$	$[11, 11]$	119.828351

6 VND-GDE3

The idea of handling the variable decision space presented in Section 3.2 is applied to the multi-objective GDE3. At first, the GDE3 algorithm will be described. Afterward, a VND-GDE3 and VLGDE3 algorithms are presented where the VND-GDE3 is considered to be a pure-VND algorithm while the VLGDE3 is not.

6.1 GDE3

Generalized Differential Evolution (GDE3) is based on a Differential Evolution algorithm proposed in 1997 [4]. It is a population-based real-numbered optimization algorithm with selection and crossover operators. A random initial population is created at the beginning. Agents' positions are then altered to find better solutions in every iteration until a stopping criterion (usually predefined number of iterations) is met.

The crossover procedure is performed by constructing a trial vector ($\mathbf{u}_{i,g}$) for each decision vector ($\mathbf{x}_{i,g}$) of the population. Here, i is the index of an agent in the population, and g is an iteration (generation) index. The trial vector is derived with following pseudocode:

Algorithm 6.1: Pseudocode of crossover operator in GDE3.

r_1, r_2, $r_3 \in \{1, 2, \dots N\}$;
r_1, r_2, $r_3 \neq i$;
$j_{\text{rand}} \in 1, 2, \dots D$;
for $(j = 1 : D)$ **do**
  if $(\text{rnd}\,(1) < P_C \vee j = j_{\text{rand}})$ **then**
    $u_{j,i,g} = x_{j,r_3,g} + F \cdot (x_{j,r_1,g} - x_{j,r_2,g})$;
  else
    $u_{j,i} = x_{j,i}$;
  end
end

where N denotes the number of agents in population, j denotes the j-th decision variable, D is the number of decision variables, F is the scaling factor, r_1,r_2, and r_3 are randomly selected agents' indices (mutually different and different from i). Not all the trial vectors replace all the old vectors. The ratio of replaced vectors is controlled by the crossover probability (P_C).

In multi-objective optimization, where the objectives are conflicting, a set of trade-off solutions constitute the Pareto-front. GDE3 selects trade-off solutions based on the dominance principle [5, Chapter 2] as follows:

- If the old vector dominates the trial vector, the old vector remains as it is for the next iteration.

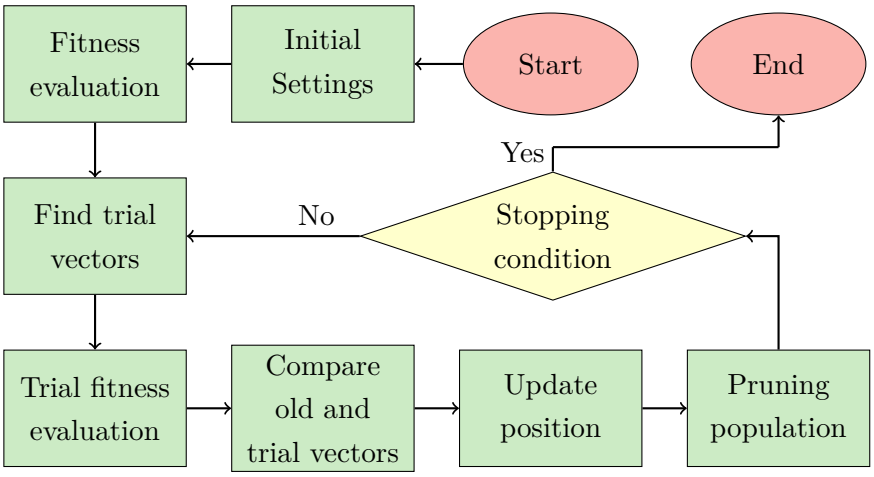


Figure 6.1: A flowchart of the VND-GDE3 algorithm.

- If the trial vector dominates the old vector, the trial vector replaces the old vector for the next iteration.
- If the old vector and the trial vector are non-dominated, both vectors are selected, and the population is temporarily extended.

However, the size of the population has to be limited due to increasing computational demands. When the trial and old solutions are non-dominated, both solutions remain in the population. If there are more non-dominated solutions than the number of agents, the extended population is trimmed using the crowding distance metric. This approach enhances the diversity of the solution.

6.2 VND-GDE3

Algorithm VND-GDE3 [62] is very similar to its predecessor – GDE3. Nonetheless, its agents can have a different number of decision variables in any iteration. The only differences are related to the crossover of the decision vectors of different sizes. Moreover, if a problem has a fixed number of decision variables, the VND-GDE3 acts identically as the GDE3 algorithm. Although, it might be slightly more computationally demanding. Variability of solution's dimensionality introduces a new user-defined parameter probability of dimension transition (P_{DT}).

The crossover operator in Algorithm 6.1 mixes the decision variables of three different agents into a single trial decision variable with a probability of crossover P_C. Otherwise, the decision variable remains as it is.

The number of decision variables may differ between the three vectors. Therefore, the dimensionality of a trial vector D_{new} has to be determined beforehand. Dimensionality D_{new} is one of the following:

- dimensionality of the current agent D_i,
- dimensionality of the first randomly picked agent D_{r_1},

- dimensionality of the second randomly picked agent D_{r_2},
- dimensionality of the third randomly picked agent D_{r_3}.

The dimensionality of the trial solution D_{new} is equal to D_i with the probability P_{DT}. Otherwise, it is one of the dimensionalities D_{r_1}, D_{r_2}, or D_{r_3} (picked with equal probability).

Afterward, four artificial agents are derived from the current agent and random agents (r_1, r_2, and r_3), but they all have the same size D_{new}. Note that missing decision variables are filled randomly and that only an undivided part of the decision space vector can be deleted from the ending part of it (please refer to [48], Figure 1).

6.3 VLGDE3

Variable-length Generalized Differential Evolution (VLGDE3) was inspired by [40] and [50]. The algorithm is based on the GDE3 algorithm.

The decision vectors of all agents are extended by $|D|$ decision variables. These additional decision variables (called padding variables) define which original variables (called sample variables) are active or inactive in the fitness function evaluation. This, in principle, is identical to the approach where a secondary vector accompanies the decision vector, as described in Section 3.3.

The number of padding decision variables is not equal to the maximal dimensionality of the problem. There are specific applications where several decision variables are combined into clusters. For example, in the transmitter placement problem [MM2], each transmitter may be described by x-position, y-position, and its power. Therefore, one cluster consists of three decision variables, but only one padding variable is needed.

Positions of agents (including the padding variables) are set randomly during the initialization. Padding decision variables are bounded between zero and one. As mentioned before, padding variables determine which sample decision variables participate in fitness function evaluation. Therefore, if only one padding cell contains a value greater than the threshold ($T_{\text{pad}} = 0.5$), then only the first cluster is involved in the fitness function evaluation.

Positions of agents are updated using standard DE equations because all particles have the same number of decision variables. However, padding variables extend the decision vector. Therefore, the overall decision space becomes larger, which makes the problem more difficult to overcome.

7 VND-MOPSO

In this chapter, the multi-objective version of VNDPSO (see Section 3.2) is shown. The generic MOPSO algorithm is described first. Afterward, a VND-MOPSO and VLMOPSO algorithms are presented where the VND-MOPSO is considered to be a pure-VND algorithm while the VLMOPSO is not.

7.1 MOPSO

Particle Swarm Optimization is an evolutionary algorithm, which simulates the movement of a swarm of bees searching for food. It was originally proposed for use in neural networks [3] and was later adopted as a global optimizer. Multi-objective Particle Swarm Optimization is described in [9].

The position of each agent is changed according to its own experience and that of its neighbors. The position $\mathbf{x}$ of agent in the subsequent iteration is changed by adding velocity $\mathbf{v}$ to a current position:

$$\mathbf{x}_g = \mathbf{x}_{g-1} + \mathbf{v}_g, \tag{7.1}$$

where the velocity vector $\mathbf{v}_g$ is defined by equation:

$$\mathbf{v}_g = W \cdot \mathbf{v}_{g-1} + c_1 \cdot r_1 \cdot (\mathbf{x}_{\text{pbest}} - \mathbf{x}_{g-1}) + c_2 \cdot r_2 \cdot (\mathbf{x}_{\text{gbest}} - \mathbf{x}_{g-1})\,. \tag{7.2}$$

The current particle $\mathbf{x}_{g-1}$ mingles with its attractors (personal best position $\mathbf{x}_{\text{pbest}}$ and global best position $\mathbf{x}_{\text{gbest}}$). Note that W is the inertia weight, c_1 and c_2 are the cognitive and social learning factors, respectively, r_1 and r_2 are random values, and g is the time step.

The main difference compared to the single-objective version is the selection of global best for individual agents. In a single-objective version, only one solution is said to be the global best, but a multi-objective problem presents multiple trade-off solutions that are said to be the good ones.

Non-dominated (trade-off) solutions are stored in an external archive. The external archive holds all the non-dominated solutions found so far, and its members are used as global bests in velocity update (7.2).

At the beginning of the algorithm, the random population is generated, and an external archive is initialized with non-dominated members of the random population. Personal bests for each agent are updated in each iteration. If the solution dominates the solution from the previous iteration, the new solution is chosen as a new personal best. Otherwise, the old solution remains as the personal best. If both solutions are non-dominated, the personal best is updated. Global bests are selected among external archive members. The global best is a member of an external archive selected with equal probability.

The overall number of agents in the external archive has to be limited. External archive size is equal to the population size N. If there are more than N non-dominated

candidates for the external archive, solutions have to be pruned with regard to the diversity of the non-dominated set. The pruning method uses crowding distance as a metric for the pruning [8]. Unfortunately, the crowding distance selection works satisfactorily only with two-dimensional space, i.e. two-objective functions. To make it work also for many-objective problems (3 or more), the concept of ENNS (see Subsection 2.2.3) has to be adopted, which converts M-dimensional space into 2-dimensional space.

7.2 VND-MOPSO

The algorithm VND-MOPSO is very similar to the PSO-VND algorithm (Section 3.2). It uses the same position update equation (7.1), (7.2) as the MOPSO algorithm. However, the position vectors $\mathbf{x}_{g-1}$, $\mathbf{x}_{\text{pbest}}$, and $\mathbf{x}_{\text{gbest}}$ can have different number of decision variables. The VND-MOPSO algorithm uses probabilities p_{pbest}, p_{gbest}, and p_x to decide the dimensionality of a new solution identically to the PSOVND algorithm.

The VND-MOPSO also uses the external archive and the pruning method based on crowding distance or ENNS.

7.3 VLMOPSO

Variable-length Multi-objective Particle Swarm Optimization (VLMOPSO) was proposed in [40]. It is based on the MOPSO algorithm proposed in [9]. It uses an external archive, where non-dominated solutions found so far are stored. In order to keep a reasonable amount of non-dominated solutions in the external archive, the pruning method based on crowding distances is used. The global best solution for each particle in the swarm is picked by random from the external archive.

The particle is encoded by $|D|$ padding decision variables and $D_{\max}$ sample decision variables. However, the number of padding variables is not equal to the maximal dimensionality of the problem. There are specific applications, where several decision variables are combined into clusters. For example, in the transmitter placement problem, each transmitter may be described by x-position, y-position, and its power. Therefore, one cluster consists of three decision variables, but only one padding decision variable is needed.

Positions of particles (including the padding decision variables) are initialized with random values during initialization. Padding decision variables are bounded between zero and one. Velocities of particles are set to zeros.

Padding variables in VLMOPSO determine which decision variables (sample variables) participates in the fitness function evaluation. Therefore, if only one padding decision variable contains a value greater than threshold ($T_{\text{pad}} = 0.5$), then only the first cluster is involved in the fitness function evaluation.

Position and velocity vectors are updated using standard PSO equations because all particles have the same number of decision variables. However, padding decision variables

extend the decision vector. Therefore, the overall decision space becomes larger, which makes the problem more difficult to overcome.

8 PERFORMANCE ASSESSMENT

This section discusses the setting of controlling parameters of both VND-GDE3 and VND-MOPSO algorithms. Afterward, both algorithms are compared against each other, their impure-VND peers (VLGDE3 and VLMOPSO), and also Clustered-GDE3. The Clustered-GDE3 represents a non-VND approach used in problems with a variable number of dimensions.

Table 8.1 shows the controlling parameters of GDE3-based and MOPSO-based algorithms that are common for all simulations in this chapter. These controlling parameters are well studied in the literature. Studies of the influence of parameter probability of dimension transition P_{DT} and probabilities to follow P_{TF} are shown later in this chapter. The number of agents N and the number of iterations G used in the simulations can be found in Tables 8.2, 8.6, and 8.10.

8.1 Influence of Probability of Dimension Transition Parameter of VND-GDE3

A thorough comparative study with a set of 27 multi-objective VND benchmark problems (shown in Table 5.1) is performed to find the P_{DT} value as suitable as possible. Ten representatives (see problems in Table 8.3) from the complete problem set are selected for a more elaborate study.

The influence of the parameter P_{DT} can be seen in Figure 8.1. Each row of the heatmap combines 101 values of normalized distance hypervolume (dHV) for a given problem where the parameter P_{DT} is linearly increased from 0 to 1. Although the value $P_{\mathrm{DT}} = 0.35$ might be imperfect for some problems, it was chosen as a good trade-off value for further analyses.

Table 8.1: Controlling parameters of algorithms

MOPSO		**GDE3**	
Inertia weight	$W \in [0.8, 0.5]$	Probability of crossover	$P_C = 0.2$
Cognitive learning factor	$c_1 = 1.5$	Scaling factor	$F = 0.2$
Social learning factor	$c_2 = 1.5$		
Boundary type	reflecting		

Table 8.2: List of parameter settings used for the study of influence of P_{DT}.

Settings	N	G	P_{DT}	**Settings**	N	G	P_{DT}
SET A.1	100	100	0.05	SET A.6	50	50	0.35
SET A.2	100	100	0.15	SET A.7	50	50	0.95
SET A.3	100	100	0.35	SET A.8	200	200	0.35
SET A.4	100	100	0.65	SET A.9	200	200	0.95
SET A.5	100	100	0.95				

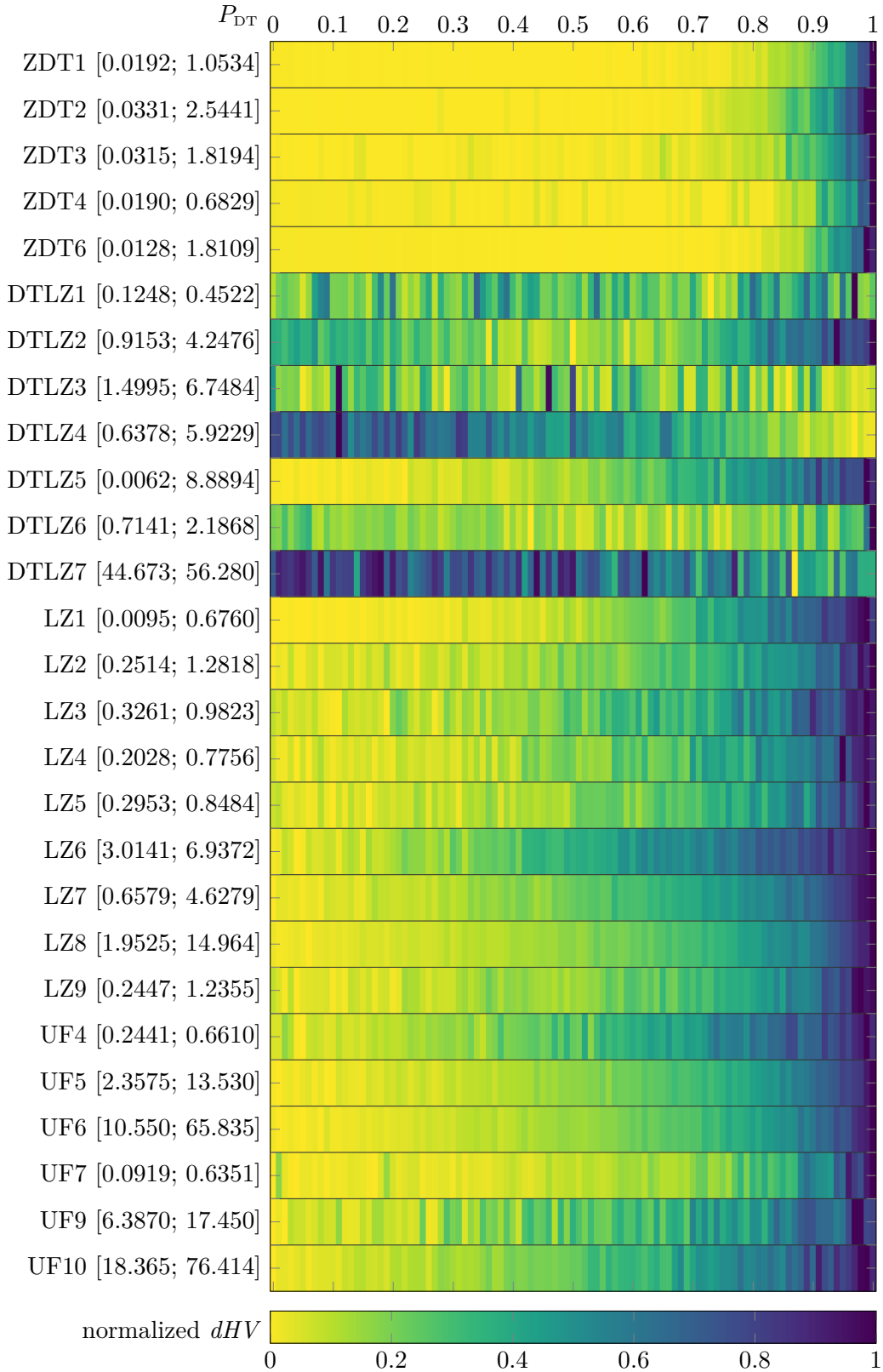


Figure 8.1: Heatmap of normalized dHV values of the complete benchmark problems set. Parameter P_{DT} grows linearly from 0 to 1, dHV values were normalized separately for each problem. Therefore, the values are scaled from 0 to 1 for each problem. Values inside square brackets following the problem name denote the minimal and maximal absolute dHV value, respectively.

Tables 8.3, 8.4, and 8.5 show values of distance hypervolume dHV, generational distance GD, and the number of variables deviation nVD for different settings from Table 8.2 for several benchmark problems. These benchmark problems were selected intentionally for their variety in Pareto-front shapes. It can be seen that parameter P_{DT} has a major impact on the algorithm's performance. For most of the problems, the best performance is obtained with lower values of P_{DT}. However, there are such exceptions as DTLZ2, DTLZ4, or LZ8 problems. Also, by comparing Tables 8.3 and 8.5, a relation is to be seen between the quality of the Pareto-front and the deviation from the optimal dimensionality. Settings with a different number of iterations and a number of agents (SET A.6 – SET A.9) are here to demonstrate that the influence of P_{DT} is independent of the number of fitness function evaluations.

A graphical expression of the influence of P_{DT} is shown in Figure 8.2 with standard boxplots. It is advantageous to use lower values of P_{DT} for most of the problems. However, the trend seems reversed in the case of DTLZ2, DTLZ4, and LZ8 problems.

Table 8.3: Average distance hypervolume (dHV) for a given set of settings.

SET	ZDT2	ZDT4	ZDT6	DTLZ2	DTLZ4	LZ6	LZ8	UF4	UF6	UF10
A1	0.0544	0.0313	0.0391	2.0754	0.6688	4.7568	2.0030	0.2585	7.6110	19.650
A2	0.2199	0.0426	0.0555	1.6283	0.8428	4.7006	1.5463	0.2623	6.1610	15.591
A3	0.7961	0.0641	0.0663	1.6040	0.6517	4.6321	1.1308	0.2256	5.4450	18.324
A4	1.9198	0.0880	0.0910	1.3261	0.5497	4.2029	1.0218	0.2098	5.4970	17.942
A5	2.1500	0.0911	0.0963	1.1443	0.4029	4.2441	0.8643	0.2347	5.8970	20.971
A6	3.7755	0.2562	0.1186	2.7076	2.7194	6.0357	4.7140	0.5700	16.330	72.986
A7	5.4164	0.2350	0.1247	2.2250	4.3400	5.8901	4.1700	0.5242	17.018	79.452
A8	0.0732	0.0381	0.0571	1.8130	0.4728	4.9965	0.3850	0.1086	1.3715	10.387
A9	0.7407	0.0880	0.0944	1.1366	0.4602	3.9993	0.3271	0.1173	1.8960	9.8990

Table 8.4: Average generational distance (GD) for a given set of settings.

SET	ZDT2	ZDT4	ZDT6	DTLZ2	DTLZ4	LZ6	LZ8	UF4	UF6	UF10
A1	0.0073	0.0032	0.0162	0.0109	0.0288	0.2102	1.0777	0.0785	1.9590	2.1590
A2	0.0168	0.0079	0.0229	0.0115	0.0275	0.2300	1.0023	0.0822	1.7580	2.4230
A3	0.0360	0.0179	0.0261	0.0127	0.0323	0.1968	0.8446	0.0825	1.5480	2.3280
A4	0.0420	0.0296	0.0349	0.0129	0.0307	0.2580	0.7784	0.0841	2.2650	2.5670
A5	0.0516	0.0317	0.0364	0.0153	0.0342	0.2664	0.6613	0.0938	2.0150	2.4730
A6	0.0319	0.0359	0.0341	0.0268	0.0412	0.3087	2.7783	0.1053	4.7590	2.6150
A7	0.0344	0.0383	0.0414	0.0297	0.0417	0.3466	2.8407	0.1156	5.7380	2.7800
A8	0.0174	0.0068	0.0261	0.0074	0.0224	0.1079	0.2020	0.0584	0.5262	3.0065
A9	0.0560	0.0306	0.0373	0.0075	0.0270	0.2082	0.1381	0.0679	0.9000	2.6110

Table 8.5: Average deviation from optimal dimensionality (nVD) for a given settings.

SET	ZDT2	ZDT4	ZDT6	DTLZ2	DTLZ4	LZ6	LZ8	UF4	UF6	UF10
A1	0.1544	0.0373	0.2621	0.0645	0.2453	0.0803	0.4259	0.7860	1.3316	0.0122
A2	0.3106	0.0919	0.3577	0.0709	0.2338	0.0848	0.4283	0.7851	1.2155	0.0116
A3	0.6092	0.2101	0.4081	0.0835	0.2820	0.0760	0.4134	0.7445	1.2991	0.0076
A4	0.6773	0.3492	0.5260	0.0765	0.2727	0.1007	0.4514	0.8090	1.2262	0.0162
A5	0.8127	0.3727	0.5542	0.0956	0.3077	0.1332	0.4615	0.9303	1.3877	0.0113
A6	0.5340	0.3168	0.5192	0.2564	0.3918	0.1057	0.3485	0.8070	1.2753	0.0838
A7	0.5252	0.3722	0.6156	0.2616	0.4044	0.1363	0.3951	0.9478	1.2599	0.0778
A8	0.3403	0	0.4020	0.0185	0.1478	0.0418	0.4405	0.6948	1.2116	0
A9	0.8897	0	0.5622	0.0152	0.2132	0.0691	0.4982	0.8446	1.3227	0

8.2 Influence of Probabilities to Follow Parameter of VND-MOPSO

The parameter P_{TF} of MOPSO consists of three values – p_1, p_2, and p_3. The first value defines the probability that the new position vector will be of the size of global best. The second one corresponds to the probability that the size of the personal best will be picked. The last one (p_3) corresponds to the probability that the number of decision variables of the particle will be unchanged.

The authors of [48] studied the parameter in the single-objective PSOVND. They made

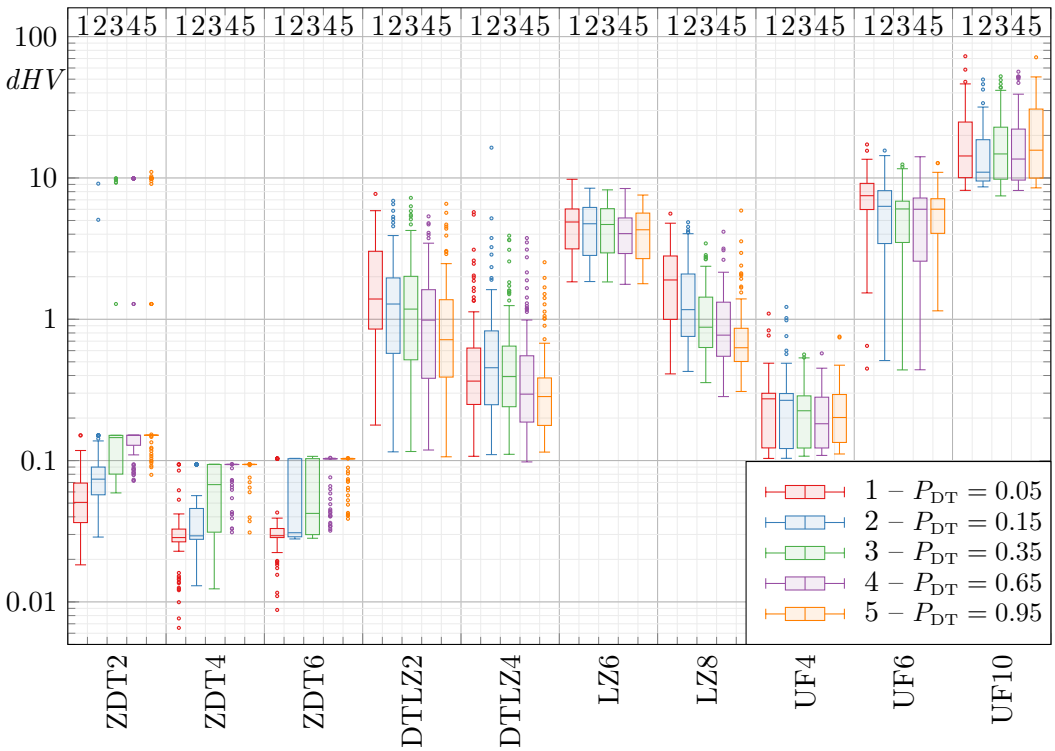


Figure 8.2: Standard boxplots expressing the influence of P_{DT} on the dHV value.

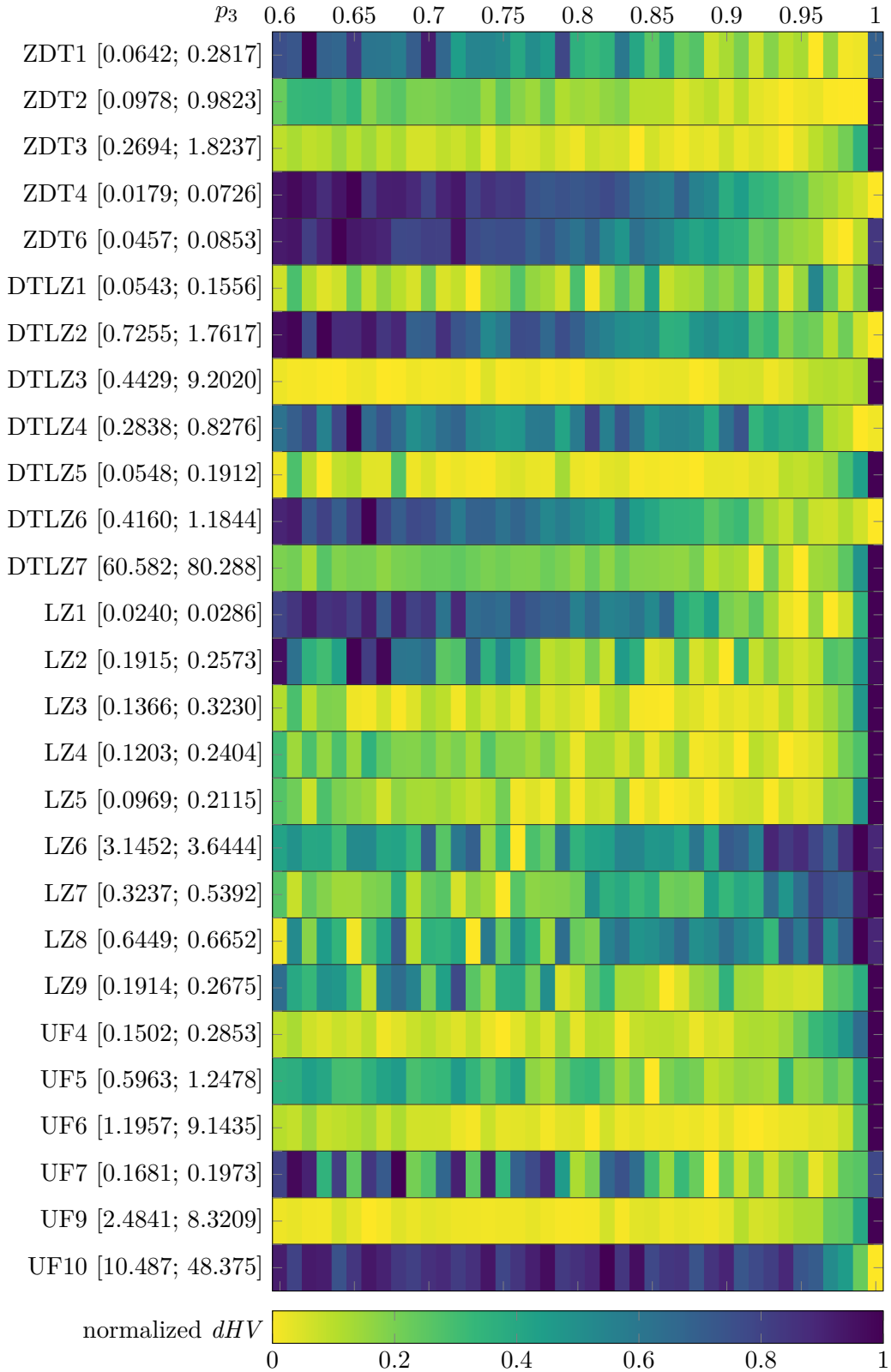


Figure 8.3: Heatmap of normalized dHV values of the complete benchmark problems set. Parameter p_3 grows linearly from 0.6 to 1, dHV values were normalized separately for each problem. Therefore, the dHV values are scaled from 0 to 1 for each problem. Values inside square brackets following the problem name denote the minimal and maximal absolute dHV value, respectively.

several recommendations. The first one is that settings p_1 too high leads all the particles to follow the dimensionality of the global best particle. Therefore, the algorithm will be prone to get stuck in sub-optimal dimensionalities. They recommended that $p_3 >= 0.6$, $p_1 = 0.5p_2$, and $p_1 + p_2 = 1 - p_3$. These recommendations allow us to select only the p_3 and derive the rest.

Parameter p_3 is linearly increased from 0.6 to 1 in a comparative study with the set of 27 multi-objective VND benchmark problems (see Table 5.1). Afterward, ten representatives are selected for a more elaborate study.

Figure 8.3 shows the influence of the parameter p_3. Each row of the heatmap com-

Table 8.6: List of parameter settings used for the study of influence of P_{TF}.

Settings	N	G	p_1	p_2	p_3	**Settings**	N	G	p_1	p_2	p_3
SET B.1	100	100	0.133	0.267	0.60	SET B.6	100	100	0	0	1
SET B.2	100	100	0.107	0.213	0.68	SET B.7	50	50	0.133	0.267	0.60
SET B.3	100	100	0.080	0.160	0.76	SET B.8	50	50	0.027	0.053	0.92
SET B.4	100	100	0.053	0.107	0.84	SET B.9	200	200	0.133	0.267	0.60
SET B.5	100	100	0.027	0.053	0.92	SET B.10	200	200	0.027	0.053	0.92

Table 8.7: Average distance hypervolume (dHV) for a given set of settings.

SET	**ZDT2**	**ZDT4**	**ZDT6**	**DTLZ2**	**DTLZ4**	**LZ6**	**LZ8**	**UF4**	**UF6**	**UF10**
B1	0.3049	0.0695	0.0825	1.7226	0.6237	3.3561	0.6454	0.1629	2.0456	45.396
B2	0.3290	0.0681	0.0767	1.6510	0.6368	3.3571	0.6595	0.1562	2.1674	42.225
B3	0.2587	0.0640	0.0759	1.5242	0.5457	3.1452	0.6549	0.1560	1.5172	42.574
B4	0.1904	0.0529	0.0708	1.2337	0.6198	3.4151	0.6547	0.1617	1.4989	46.922
B5	0.1333	0.0371	0.0549	1.0292	0.4736	3.4236	0.6549	0.1670	1.1957	39.947
B6	0.9823	0.0179	0.0790	0.7255	0.2938	3.5908	0.6630	0.2853	9.1435	10.487
B7	0.5976	0.0867	0.1189	2.4526	0.5946	4.4886	0.6620	0.2814	4.6129	25.699
B8	0.6241	0.0676	0.1672	1.8984	0.4792	4.4753	0.6618	0.4160	6.2237	21.004
B9	0.2209	0.0435	0.0623	0.8007	0.5761	2.5865	0.6203	0.1240	1.2030	36.327
B10	0.0861	0.0198	0.0321	0.4246	0.4345	2.7117	0.6294	0.1218	0.7827	40.604

Table 8.8: Average generational distance (GD) for a given set of settings.

SET	**ZDT2**	**ZDT4**	**ZDT6**	**DTLZ2**	**DTLZ4**	**LZ6**	**LZ8**	**UF4**	**UF6**	**UF10**
B1	0.0676	0.0219	0.0271	0.0528	0.0383	0.4902	0.0437	0.1091	0.7580	5.8482
B2	0.0639	0.0211	0.0237	0.0502	0.0386	0.5215	0.0518	0.1100	0.5462	6.1140
B3	0.0653	0.0189	0.0227	0.0479	0.0371	0.4821	0.0465	0.1091	0.4593	5.9665
B4	0.0556	0.0120	0.0209	0.0470	0.0366	0.4918	0.0594	0.1095	0.5343	6.1800
B5	0.0445	0.0066	0.0126	0.0428	0.0297	0.4764	0.0504	0.1136	0.5061	6.0576
B6	0.0383	0.0010	0.0048	0.0423	0.0333	0.4632	0.0255	0.1353	3.5909	3.1644
B7	0.0789	0.0310	0.0338	0.0623	0.0358	0.6606	0.0265	0.1329	2.6958	4.4839
B8	0.0678	0.0200	0.0195	0.0629	0.0286	0.6414	0.0133	0.1426	2.7098	4.2233
B9	0.0527	0.0091	0.0181	0.0273	0.0359	0.3141	0.0748	0.0897	0.2634	5.9516
B10	0.0231	0.0010	0.0065	0.0226	0.0270	0.3023	0.0736	0.0942	0.2613	6.0670

bines 41 values of normalized distance hypervolume dHV for a given problem where the parameter p_3 is linearly increased from 0.6 to 1. The optimality of the parameter p_3 is strongly dependent on the problem. Value $p_3 = 0.92$ was chosen as a good trade-off for all the problems.

Tables 8.7, 8.8, and 8.9 show values of distance hypervolume dHV, generational distance GD, and the number of variables deviation nVD for different settings from Table 8.6 for chosen benchmark problems. The parameter P_{TF} has a major impact on the algorithm's performance. Settings with a different number of iterations and the number of agents (SET B.7 – SET B.10) are here to demonstrate that the influence of P_{TF} is independent of the number of fitness function evaluations. Figure 8.4 shows a graphical expression of the influence of the P_{TF} parameter on the algorithm's performance.

8.3 Comparison of Clustered, Pure-VND, and Impure-VND Approaches

To perform a fair comparative study between the VND algorithm and a standard non-VND GDE3 algorithm, a reasonable approach is to use several simple GDE3 runs (clusters), each with a different number of decision variables. The number of agents of all the clusters summed together is identical to the number of agents of VND-GDE3 and VLGDE3 algorithms. The number of iterations remains fixed for all clusters. The Pareto-fronts from each separate run are combined, and the non-dominated solutions constitute the resulting non-dominated set. We have modified algorithm GDE3 accordingly and it is called the Clustered-GDE3 algorithm.

Five algorithms – VND-GDE3, VLGDE3, VND-MOPSO, VLMOPSO, and Clustered-GDE3 – were exploited on a set of ten benchmark problems with four dimensionality settings (see Table 8.10). Results can be seen in boxplots in Figures 8.5, 8.6, 8.7, 8.8, and 8.9 and Tables 8.11, 8.12, and 8.13. There are 20 boxes for each problem in the boxplots, where each quintet corresponds to a single dimensionality scenario. It is clearly visible

Table 8.9: Average deviation from optimal dimensionality (nVD) for a given settings.

SET	ZDT2	ZDT4	ZDT6	DTLZ2	DTLZ4	LZ6	LZ8	UF4	UF6	UF10
B1	1.1424	0.2548	0.4673	0.4493	0.3748	0.0765	0.1264	1.1176	1.3469	0.0172
B2	1.0521	0.2437	0.4279	0.4104	0.3796	0.0738	0.1528	1.0957	1.3612	0.0193
B3	1.0876	0.2177	0.4126	0.3728	0.3700	0.0667	0.1419	1.1660	1.2363	0.0210
B4	0.9451	0.1374	0.3814	0.3726	0.3626	0.0700	0.1850	1.1314	1.1802	0.0249
B5	0.7698	0.0709	0.2868	0.3028	0.2893	0.0438	0.1486	1.1179	1.1031	0.0193
B6	0.5723	0.0008	0.0985	0.2084	0.1691	0.0264	0.0433	1.0243	0.7618	0.0014
B7	1.3305	0.3538	0.5352	0.4272	0.3149	0.1137	0.0783	1.0475	1.0832	0.0095
B8	1.1732	0.2172	0.3546	0.3986	0.2252	0.0937	0.0300	1.0808	0.7985	0.0350
B9	0.8712	0.1058	0.3338	0.1672	0.3978	0.0634	0.2248	1.1752	1.1413	0.0200
B10	0.4342	0.0081	0.1385	0.1037	0.2846	0.0474	0.2030	1.1726	1.1039	0.0204

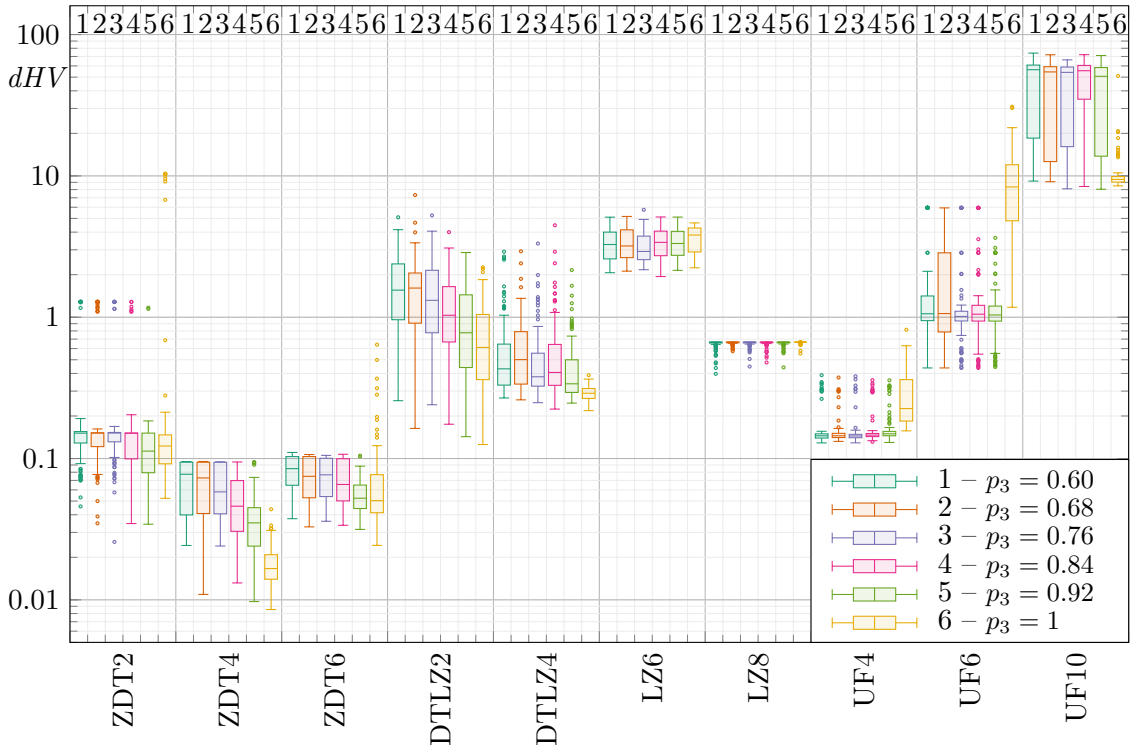


Figure 8.4: Standard boxplots expressing the influence of P_{TF} on the dHV value.

that metric values deteriorate as the dimensionality of the problem grows.

The decision space in the case of sets C.1 – C.5 is much smaller than the decision

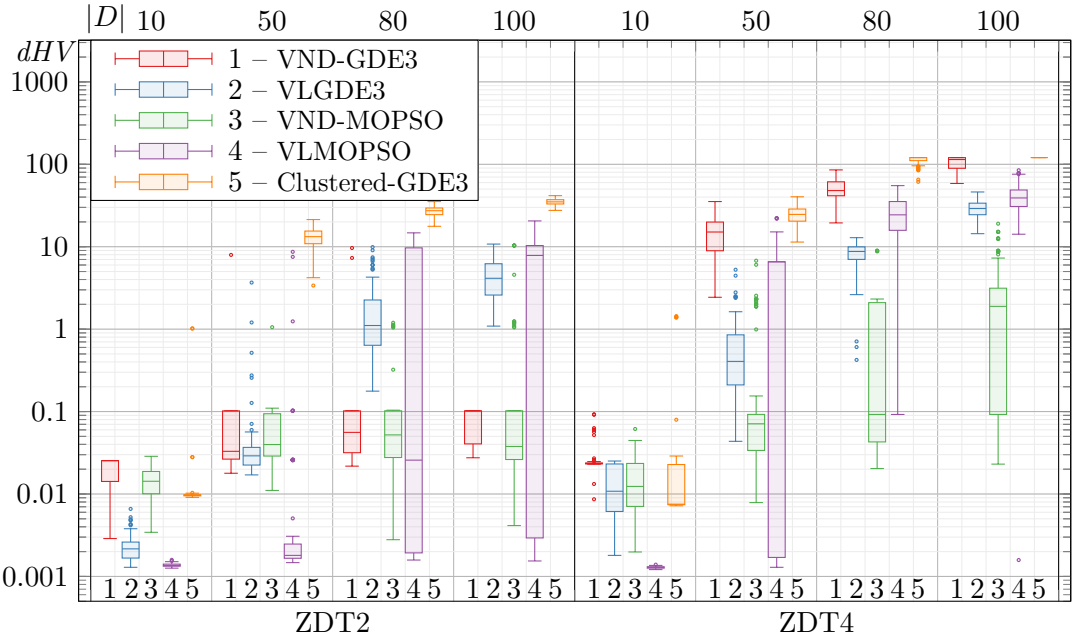


Figure 8.5: Comparison of VND-GDE3, VLGDE3, VND-MOPSO, VLMOPSO, and Clustered-GDE3 on ZDT2 and ZDT4 problems.

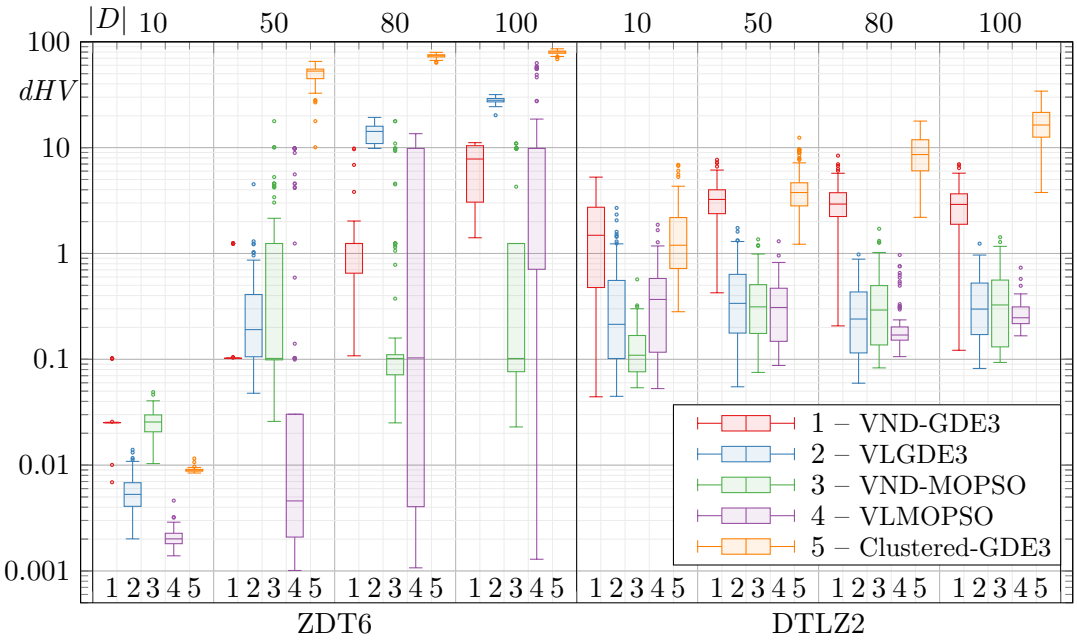


Figure 8.6: Comparison of VND-GDE3, VLGDE3, VND-MOPSO, VLMOPSO, and Clustered-GDE3 on ZDT6 and DLTZ2 problems.

Table 8.10: List of parameter settings used for VND-GDE3 simulations.

SET	N	G	**Algorithm**	$\|D\|$	D	$D^{(opt)}$
C1	400	200	VND-GDE3	10	{3,4,...,12}	{3,4,5}
C2	400	200	VLGDE3	10	{3,4,...,12}	{3,4,5}
C3	400	200	VND-MOPSO	10	{3,4,...,12}	{3,4,5}
C4	400	200	VLMOPSO	10	{3,4,...,12}	{3,4,5}
C5	40*	200	Clustered-GDE3	10	{3,4,...,12}	{3,4,5}
C6	400	200	VND-GDE3	50	{3,4,...,52}	{10,11,12}
C7	400	200	VLGDE3	50	{3,4,...,52}	{10,11,12}
C8	400	200	VND-MOPSO	50	{3,4,...,52}	{10,11,12}
C9	400	200	VLMOPSO	50	{3,4,...,52}	{10,11,12}
C10	8*	200	Clustered-GDE3	50	{3,4,...,52}	{10,11,12}
C11	400	200	VND-GDE3	80	{3,4,...,82}	{15,16,17}
C12	400	200	VLGDE3	80	{3,4,...,82}	{15,16,17}
C13	400	200	VND-MOPSO	80	{3,4,...,82}	{15,16,17}
C14	400	200	VLMOPSO	80	{3,4,...,82}	{15,16,17}
C15	5*	200	Clustered-GDE3	80	{3,4,...,82}	{15,16,17}
C16	400	200	VND-GDE3	100	{3,4,...,102}	{20,21,22}
C17	400	200	VLGDE3	100	{3,4,...,102}	{20,21,22}
C18	400	200	VND-MOPSO	100	{3,4,...,102}	{20,21,22}
C19	400	200	VLMOPSO	100	{3,4,...,102}	{20,21,22}
C20	4*	200	Clustered-GDE3	100	{3,4,...,102}	{20,21,22}

*Number of agents N of each cluster.

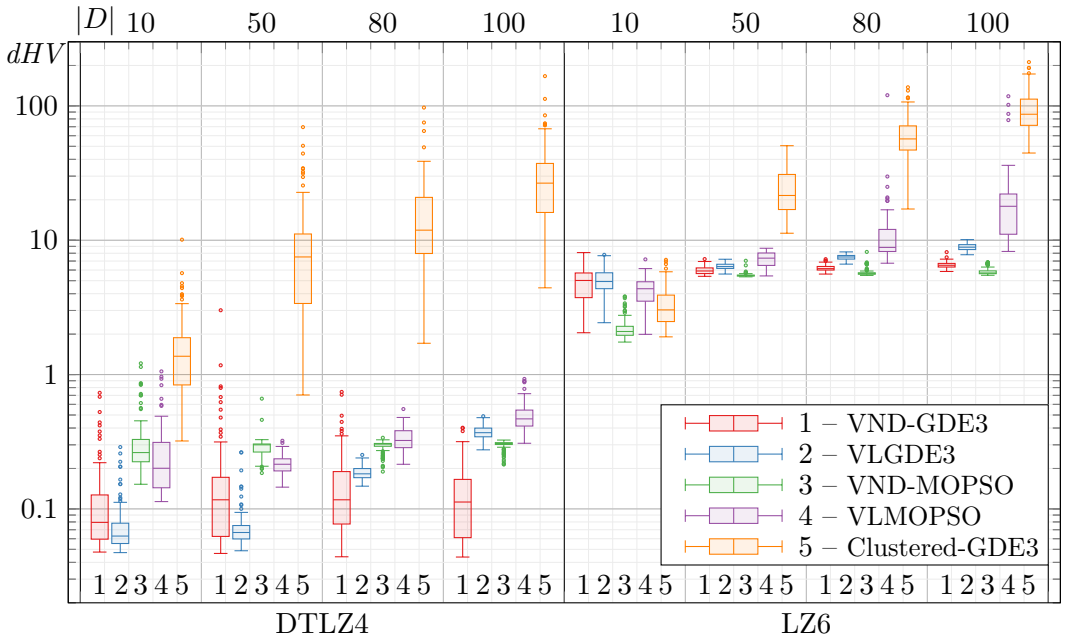


Figure 8.7: Comparison of VND-GDE3, VLGDE3, VND-MOPSO, VLMOPSO, and Clustered-GDE3 on DTLZ4 and LZ6 problems.

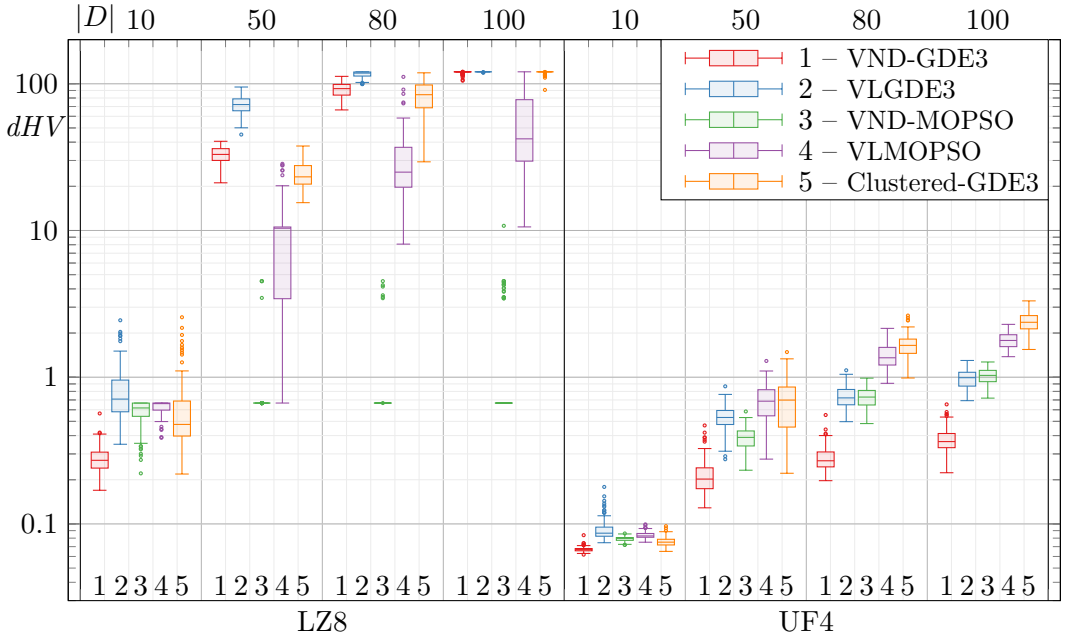


Figure 8.8: Comparison of VND-GDE3, VLGDE3, VND-MOPSO, VLMOPSO, and Clustered-GDE3 on LZ8 and UF4 problems.

space in the case of sets C.16 – C.20. Therefore, the Clustered-GDE3 was able to find decent Pareto-fronts because the algorithm deliberately searched every dimensionality of the problem in the low dimensionality scenario. Forty agents for each cluster was enough to explore the corresponding dimensionality sufficiently. Contrarily, the Clustered-GDE3 spent most of its efforts searching in non-optimal dimensions in the high dimensionality

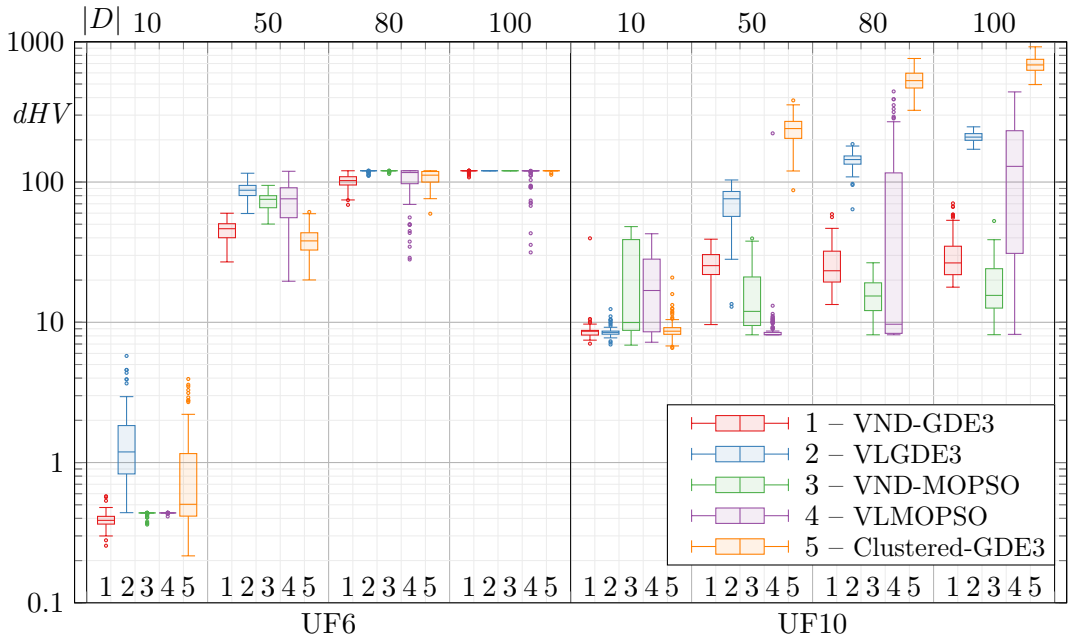


Figure 8.9: Comparison of VND-GDE3, VLGDE3, VND-MOPSO, VLMOPSO, and Clustered-GDE3 on UF6 and UF10 problems.

Table 8.11: Average distance hypervolume (dHV) for a given set of settings.

SET	ZDT2	ZDT4	ZDT6	DTLZ2	DTLZ4	LZ6	LZ8	UF4	UF6	UF10
C1	0.0207	0.0271	0.0310	1.7304	0.1236	4.9107	0.2828	0.0671	0.3867	8.8742
C2	0.0024	0.0128	0.0058	0.4432	0.0792	5.0128	0.8391	0.0929	1.4852	8.5796
C3	0.0148	0.0149	0.0259	0.1312	0.3269	2.2327	0.5748	0.0791	0.4333	20.021
C4	0.0014	0.0013	0.0021	0.4270	0.2643	4.2661	0.6257	0.0839	0.4371	19.485
C5	0.0302	0.0701	0.0090	1.7117	1.7053	3.3450	0.6461	0.0760	0.9408	8.9979
C6	0.1313	15.022	0.2731	3.2684	0.1977	5.9720	32.947	0.2162	45.531	25.819
C7	0.0894	0.7085	0.3519	0.4510	0.0748	6.3874	71.850	0.5374	87.468	70.869
C8	0.0657	0.5341	1.2675	0.3952	0.2875	5.4900	0.8101	0.3910	73.713	16.175
C9	0.1842	4.7453	1.3338	0.3543	0.2155	7.2379	8.8346	0.6951	74.178	10.765
C10	13.139	24.618	49.465	4.3554	10.258	24.974	24.252	0.6988	38.402	237.10
C11	0.2345	50.995	1.2437	3.1595	0.1588	6.2069	91.999	0.2826	101.51	26.826
C12	1.9930	8.2417	14.052	0.2922	0.1864	7.4858	115.75	0.7452	119.90	143.09
C13	0.1230	1.1673	1.2240	0.3933	0.2924	5.7777	0.8616	0.7309	120.21	15.743
C14	3.0382	26.657	4.2786	0.2195	0.3358	11.683	29.660	1.4092	104.10	71.925
C15	26.892	113.62	73.608	9.1589	16.830	61.452	83.214	1.6444	107.76	539.15
C16	0.0777	102.55	6.9278	2.9256	0.1372	6.5641	119.66	0.3813	119.79	30.886
C17	4.5649	29.304	27.997	0.3758	0.3771	8.9248	120.63	0.9875	120.44	209.01
C18	0.6967	3.2041	2.5666	0.4087	0.2979	5.9130	1.3658	1.0172	120.44	19.180
C19	6.2390	41.394	11.981	0.2743	0.4990	20.393	55.888	1.7956	113.97	155.77
C20	34.939	120.67	79.665	17.118	31.736	97.288	119.80	2.4183	120.32	692.18

scenario.

The same applies when comparing VL to VND algorithms. VLGDE3 and VLMOPSO

Table 8.12: Average generational distance (GD) for a given set of settings.

SET	ZDT2	ZDT4	ZDT6	DTLZ2	DTLZ4	LZ6	LZ8	UF4	UF6	UF10
C1	0.0110	0.0015	0.0170	0.0064	0.0142	0.0339	0.1284	0.0329	0.4160	2.4548
C2	0.0000	0.0001	0.0002	0.0052	0.0110	0.0277	0.6986	0.0545	1.2193	2.2937
C3	0.0038	0.0006	0.0033	0.0124	0.0257	0.2122	0.1148	0.0606	0.4939	5.0325
C4	0.0001	0.0001	0.0001	0.0137	0.0176	0.0742	0.1060	0.0593	0.7913	4.9736
C5	0.0005	0.0033	0.0001	0.0388	0.0423	0.4862	0.4084	0.0367	0.9339	1.2889
C6	0.0165	0.8824	0.0513	0.0067	0.0313	0.0176	8.1427	0.0902	11.133	0.8285
C7	0.0021	0.0012	0.0030	0.0065	0.0152	0.0506	10.526	0.1088	12.267	1.8487
C8	0.0191	0.0495	0.1240	0.0290	0.0370	0.0827	0.0128	0.1433	13.503	2.4911
C9	0.0055	0.3689	0.0125	0.0379	0.0391	0.1011	0.8474	0.1358	11.677	1.0286
C10	0.6477	1.8461	3.8178	0.1327	0.1117	0.1881	6.4253	0.1648	8.7900	1.9795
C11	0.0202	3.9856	0.0820	0.0066	0.0352	0.0167	20.513	0.1069	25.977	0.5614
C12	0.0361	0.0634	0.3935	0.0151	0.0520	0.0766	14.964	0.1243	17.364	2.2798
C13	0.0252	0.0815	0.0808	0.0371	0.0398	0.0547	0.0251	0.2366	31.285	1.3203
C14	0.0319	2.3137	0.0144	0.0789	0.0778	0.1763	8.0942	0.1676	14.223	1.2716
C15	1.6960	12.405	5.8774	0.4118	0.3487	0.3188	19.921	0.3756	29.014	3.6947
C16	0.0252	9.4347	0.1221	0.0067	0.0350	0.0195	34.023	0.1243	44.107	0.5244
C17	0.0919	1.7927	1.6603	0.0331	0.1199	0.0950	17.704	0.1390	19.833	2.5466
C18	0.0260	0.2164	0.0335	0.0407	0.0420	0.0578	0.0664	0.3538	56.324	0.9737
C19	0.0889	3.8378	0.5133	0.1337	0.1303	0.2403	10.867	0.1857	16.968	1.8685
C20	2.4015	31.700	6.4461	0.7774	0.7345	0.4403	40.204	0.5517	53.312	4.4521

Table 8.13: Results of Friedman's test / Wilcoxon's test (at significance level $\alpha = 0.05$): + denotes that the first setting is significantly better, – denotes that the second settings is significantly better, = denotes that the difference is not significant.

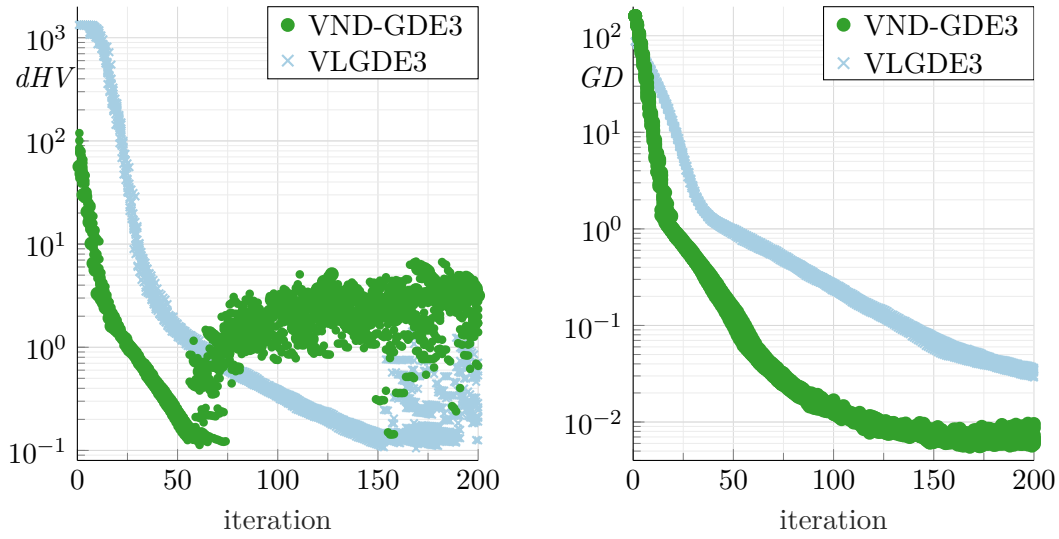
Compare SET	ZDT2	ZDT4	ZDT6	DTLZ2	DTLZ4	LZ6	LZ8	UF4	UF6	UF10
C.1 vs. C.2	-/-	-/-	-/-	-/-	=/-	=/=	+/+	+/+	+/+	=/=
C.1 vs. C.3	=/-	-/-	=/=	-/-	+/+	-/-	+/+	+/+	+/+	+/+
C.1 vs. C.5	-/-	-/-	-/-	=/=	+/+	-/-	+/+	+/+	+/+	=/=
C.3 vs. C.4	-/-	-/-	-/-	+/+	=/-	+/+	+/+	+/+	=/+	=/=
C.6 vs. C.7	-/-	-/-	=/+	-/-	-/-	+/+	+/+	+/+	+/+	+/+
C.6 vs. C.8	=/=	-/-	=/=	-/-	+/+	-/-	-/-	+/+	+/+	-/-
C.6 vs. C.10	+/+	+/+	+/+	=/+	+/+	+/+	-/-	+/+	-/-	+/+
C.8 vs. C.9	-/-	+/+	-/-	=/=	-/-	+/+	+/+	+/+	=/=	-/-
C.11 vs. C.12	+/+	-/-	+/+	-/-	=/+	+/+	+/+	+/+	+/+	+/+
C.11 vs. C.13	=/=	-/-	-/-	-/-	+/+	-/-	-/-	+/+	+/+	-/-
C.11 vs. C.15	+/+	+/+	+/+	+/+	+/+	+/+	=/-	+/+	+/+	+/+
C.13 vs. C.14	=/+	+/+	=/+	=/-	=/+	+/+	+/+	+/+	-/-	=/=
C.16 vs. C.17	+/+	-/-	+/+	-/-	+/+	+/+	=/+	+/+	+/+	+/+
C.16 vs. C.18	=/-	-/-	-/-	-/-	+/+	-/-	-/-	+/+	+/+	-/-
C.16 vs. C.20	+/+	+/+	+/+	+/+	+/+	+/+	=/=	+/+	+/+	+/+
C.18 vs. C.19	+/+	+/+	+/+	=/-	+/+	+/+	+/+	+/+	-/-	+/+

perform much better in the low dimensionality scenarios. However, padding decision variables makes the decision space of an optimization problem harder to explore. Therefore, the performance of the VL algorithms deteriorates with the growing number of decision variables of the problem quicker compared to the VND-GDE3 and VND-MOPSO algorithms.

However, Table 8.13 reveals a few exceptions. These exceptions can be divided into two groups: 1) three-objective problem DTLZ2, 2) two-objective problems that are too difficult for such a small number of fitness function evaluations (ZDT4, LZ8, and UF6, especially for SET C.16 – SET C.20).

Three objective problem DTLZ2 shows that dHV values of VND algorithms are worse than that of VL algorithms. However, GD values show that VND performed much better. This is caused by the nature of the diversity preservation ENNS algorithm (see Subsection 2.2.3). The VND-GDE3 algorithm can converge close to the true Pareto-front faster than VLGDE3. However, the diversity of the solution decreases as the algorithm exploits the proximity of the true Pareto-front. This is clearly visible in Figure 8.10 and also suggested by GD values in Table 8.12. The same applies to the VND-MOPSO and VLMOPSO comparison.

Testing problems such as ZDT4, LZ8, and UF6 are particularly demanding due to many local optima. Moreover, their count grows with the increasing dimensionality of the problem. The reason that the VLGDE3 algorithm can outperform the VND-GDE3 algorithm (at least from the GD point of view) is that VND-GDE3 quickly converges to solutions with low dimensionality. However, if the dimensionality to which the algorithm



(a) Convergence of dHV.

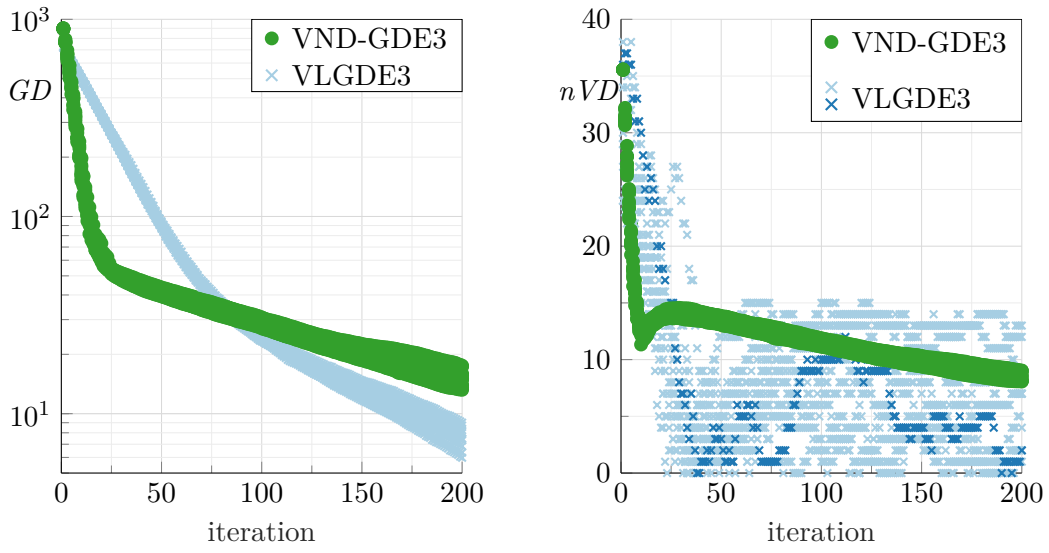
(b) Convergence of GD.

Figure 8.10: Convergence plots of modified DTLZ2 problem. Plots show 10 runs of VND-GDE3 (marked by "•") and 10 runs of VLGDE3 (marked by "×") corresponding to SET C.16 and SET C.17, respectively.

converges is locally optimal, the algorithm gets stuck there. Contrarily, VLGDE3 is naturally able to free itself from local optima dimensionality-wise either. On the other hand, the variance of the nVD values is large. Therefore, the convergence of the algorithm is not guaranteed. This situation is depicted in Figure 8.11.

A comparison between the VND-GDE3 and VND-MOPSO algorithm shows that it is almost independent of the number of decision variables of the problem. In other words, if the VND-GDE3 is better than the VND-MOPSO in the low dimensionality scenario, then it is also better in the high dimensionality scenario.

Table 8.13 shows the results of non-parametric statistical testing. Signs in the table confirm what was deduced from boxplots in Figures 8.5–8.9. The comparison seems balanced if test problems have only ten possible dimensionalities. However, the VND-GDE3 algorithm outperforms VLGDE3 in most of the problems in the case of a hundred possible dimensionalities. Note that the signs in the table are results of Friedman's and Wilcoxon's non-parametric tests with a level of significance $\alpha = 0.05$. Friedman's unadjusted p-values were adjusted by Holmberg's posthoc procedure.



(a) Convergence of GD.

(b) Convergence of nVD.

Figure 8.11: Convergence plots of modified ZDT4 problem. Plots show 10 runs of VND-GDE3 (marked by "•") and 10 runs of VLGDE3 (marked by "×") corresponding to SET C.16 and SET C.17, respectively. Note that (b) shows one of the VLGDE3 runs marked by darker "×" signs in order to show the fluctuation of nVD metric.

9 APPLICATIONS

The FOPS optimization framework was used to study the performance of VND-GDE3 and VND-MOPSO algorithms. Studies were carried out with benchmark problems, i.e., with analytically prescribed fitness functions with known minima. However, strengths of FOPS lie in a real-world application. The FOPS toolbox was designed to be as versatile as possible for real applications that have various needs. It is also used as an internal optimizer of the Antenna Toolbox for MATLAB (AToM [MM15]).

Section 9.1.1 presents an optimization aided design of a band-stop filter based on a uniplanar band-gap (UBG) planar structure. A fitness function in this optimization problem involves a full-wave analysis in the transient solver of CST Microwave Studio. This optimization problem is here to demonstrate the potential of the FOPS toolbox. It was published in [MM7, MM2].

A hybrid optimization – local optimization routine nested in a fitness function of a global optimizer is shown in Section 9.2. The decision vectors of a global optimizer are modified during the fitness evaluation by the local optimization. This feature is later exploited in the clustering problem in Section 9.6. This problem is published in [MM2].

Section 9.3, shows the first VND application – the transmitter placement problem. The problem is solved here with a multi-objective algorithm with a variable number of dimensions. The number of transmitters is to be found by the algorithm as well as the output power and the placement of each transmitter. The VND approach is compared to the standard approach with a grid-like system. This problem is published in [MM2]

Section 9.3.1 shows the design of a linear antenna array. The antenna array is designed by two different approaches – a standard approach with a uniform grid that employs the standard GDE3 with a fixed number of dimensions and a VND approach that employs VND-GDE3 algorithms. This problem is published in [62].

Digital circuits are synthesized in Section 9.4. The synthesis of a digital circuit is performed with an algorithm with a variable number of dimensions. Therefore, an arbitrary number of sums of products can be synthesized in a single run. The problem formulations permit the use of both fixed and a variable number of dimensions representation. The VND approach convincingly outperforms the fixed-length approach.

VND optimization algorithm is used in the image thresholding problem in Section 9.5. In the first subsection, the advantages of the evolutionary approach over the exhaustive thresholding method in the thresholding problem are presented. Afterward, an algorithm with a variable number of dimensions is used to find an arbitrary number of thresholds for a testing image. Finally, the evolutionary thresholding method is verified on the license-plate recognition problem.

The last application of the VND algorithm is described in Section 9.6. VND algorithm is used to clustering problem, where the number of clusters is not a priori known. This is the main drawback of the K-means method. Nonetheless, the K-means method itself is used inside the fitness function evaluation. However, the initial guess of cluster centers is

proposed by the VND algorithm.

VND algorithms were also exploited in other applications such as [MM1, MM12, MM9, MM10]. However, these applications are not discussed in this thesis.

9.1 Design of an Anisotropic Band-Stop Filter

The band-stop filter consists of a microstrip line above a ground plane with an array of etched slots of varying widths (see Figure 9.1) [89]. The transmission properties of the filter are altered by changing a number of step-impedance slot lines N, a slot period a (dimension of etched slots), and an angle φ between the microstrip line and the step-impedance slots.

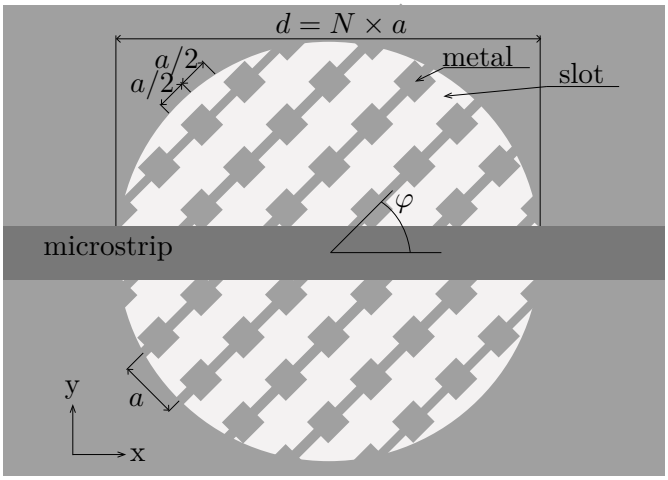


Figure 9.1: Anisotropic band-stop filter structure. Gray: metal strips, white: slots.

The transmission characteristics of the band-stop filter were obtained by a full-wave analysis in the transient solver of CST Microwave Studio. The design properties N, a, and φ were acting as decision variables, and fitness functions evaluation consists of a full-wave analysis in the CST Microwave Studio and parsing of the transmission characteristics to achieve the fitness values. An RT/Duroid substrate ($h = 0.635\,\text{mm}$, $\epsilon_r = 10.2$, $t = 35\,\mu\text{m}$) was used in the optimized structure.

Such evaluation is time demanding (approximately 5 to 30 minutes). Therefore, a great emphasis should be put on keeping the overall number of the fitness functions evaluations at minimum.

9.1.1 Optimization Parameters

Decision variables were discrete, and they were defined as follows: the number of the step-impedance slot lines $N \in \{5,\ 7,\ 9\}$, the slot period $a \in \{0.5,\ 0.6,\ \ldots, 2.0\}\,\text{mm}$, and the angle between the microstrip line and the step-impedance slots $\varphi \in \{0,\ 2,\ 4,\ \ldots, 90\}^\circ$. The fitness values were obtained from a frequency response of the transmission coefficient defined as follows (see Figure 9.2):

$$|S_{21}(\mathbf{x}, F)| > -5\,\text{dB}, F \in [0\,\text{GHz}, 4\,\text{GHz}]\,, \tag{9.1}$$

$$|S_{21}(\mathbf{x}, F)| < -20\,\text{dB}, F \in [5\,\text{GHz}, 8\,\text{GHz}]\,, \tag{9.2}$$

$$|S_{21}(\mathbf{x}, F)| > -5\,\text{dB}, F \in [9\,\text{GHz}, 10\,\text{GHz}]\,, \tag{9.3}$$

where $\mathbf{x} = \{a, N, \varphi\}^T$ is the position of a solution and F denotes the frequency.

Each frequency band forms one fitness function defined by (9.4)–(9.6). Basically, it is a sum of S_{21} values that violates the defined frequency mask (9.1)–(9.3). The CST Microwave Studio produces 5001 frequency samples of S_{21} within the interval from 0 GHz to 12 GHz.

$$f_1(\mathbf{x}) = \sum \diamondsuit \left[-S_{21}(\mathbf{x}, F) - 5\right], F \in \langle 0\,\text{GHz}; 4\,\text{GHz}\rangle \tag{9.4}$$

$$f_2(\mathbf{x}) = \sum \diamondsuit \left[20 + S_{21}(\mathbf{x}, F)\right], F \in \langle 5\,\text{GHz}; 8\,\text{GHz}\rangle \tag{9.5}$$

$$f_3(\mathbf{x}) = \sum \diamondsuit \left[-S_{21}(\mathbf{x}, F) - 5\right], F \in \langle 9\,\text{GHz}; 10\,\text{GHz}\rangle\,, \tag{9.6}$$

where the operation $\diamondsuit\,[\cdot]$ denotes that the output is equal to the argument in square brackets only if the argument is positive. Otherwise, the output is zero.

Listing 9.1 shows the definition of the band-stop filter problem. All three decision variables are discrete (`'discreteVariables'` field of the problem struct). The fitness function is defined as the function handle: `@(x, task) filterFitness(x, task)`. The property `'fullControl'` is described later in this subsection. The `'surrogate'` field enables an in-house surrogate optimization method (Section 4.4).

Listing 9.1: Band-stop filter problem definition.

```
function problem = filterProblem()
limits(:, 1) = [0.5, 2];  % slot period - a
limits(:, 2) = [0, 90];  % angle phi
limits(:, 3) = [5, 9];  % N number of slots
discreteVars{1} = 0.5:0.1:2;
discreteVars{2} = 0:2:90;
discreteVars{3} = 5:2:9;
surrogate = struct(  'type', 'tolerance-based surrogate', ...
                     'tolerance', [0.1, 2, 0.5]);
problem = struct( 'limits', limits, ...
                  'discreteVariables', {discreteVars}, ...
                  'fitness', @(x, y) filterFitness(x, y), ...
                  'fullControl', true, ...
                  'surrogate', surrogate);
end
```

Listing 9.2 shows the pseudocode of the fitness function. The `'fullControl'` option in the problem definition enables to pass the *task* reference into the fitness function evaluation. It is used here to access the CST Microwave Studio references encapsulated in *userData* of the *task* object. Dimensions of the structure in CST are changed by the `'StoreParameter'` and `'Rebuild'` commands, and the solver is started by the `'Start'` command. Commands `'SelectTreeItem'` and `'ExportPlotData'` saves the frequency response of the transmission coefficient.

Listing 9.2: Fitness function pseudocode.

```
function fitness = filterFitness(x, task)
mws = task.userData{1}(2);  % collect CST object from task
solver = task.userData{1}(3);
invoke(mws, 'StoreParameter','a', x(1));   % change a
invoke(mws, 'StoreParameter','N', x(2));   % change N
invoke(mws, 'StoreParameter','phi', x(3)); % change phi
invoke(mws, 'Rebuild');
invoke(solver, 'Start');
invoke(mws, 'SelectTreeItem', 'S21') % save S21 to *.sig files
invoke(mws, 'ExportPlotData', fileName)
S21 = readCSTFile([path, fileName]);
fitness = getFitnessValues(S21);  % parse S21 into fitness
end
```

The original paper [MM7] compares the performances of two different algorithms, GDE3 and NSGA-II, in designing the band-stop filter. Both algorithms were set to have 20 agents over 20 iterations (400 fitness function evaluations per run). Both algorithms were repeated ten times to obtain independent realizations of stochastic processes.

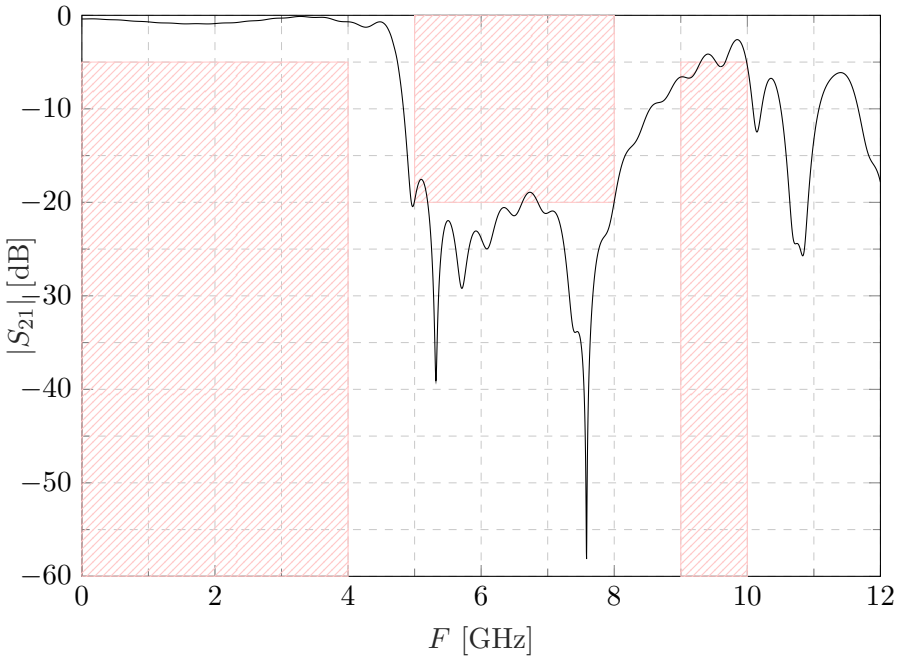


Figure 9.2: The frequency response of the transmission coefficient of the optimal solution.

The comparative study setting results in $2 \times 400 \times 10 = 8000$ fitness function evaluations (approximately 56 days of computation if the evaluation of one fitness function takes ten minutes). Fortunately, the decision space was set to a maximum of $3 \times 16 \times 46 = 2208$ possible solutions. Therefore, the Tolerance-based Surrogate Method was used here to decreases the overall number of fitness function evaluations.

Thanks to the surrogate optimization method, the whole procedure took only 15 days.

Figure 9.2 shows the most promising solution from the non-dominated set. The figure also shows the frequency mask (hatched areas) described in equations (9.1) – (9.3). The decision variables of the trade-off solution are:

- Slot period: $a = 1.5\,\mathrm{mm}$,
- Number of slots: $N = 9$,
- Rotation angle: $\varphi = 52°$.

9.2 Hybrid Optimization

The convergence rate of the optimization process can be significantly increased when hybridizing two algorithms and exploiting their strengths (e.g., one excels in exploring and one in exploiting). This will be demonstrated on a single-objective benchmark problem:

$$f(x) = 1 - \cos(10\pi x) + x^2, \tag{9.7}$$

$$x \in [-1, 1] \tag{9.8}$$

The fitness function contains eleven minima. One of them is the global one as shown in Figure 9.3. The single-objective PSO algorithm with 5 agents and 10 iterations is run (see Listing 9.3). The problem is defined using structure fields `'limits'` and `'fitness'`. Then, the option `'fullControl'` is set to the *true* value, which ensures that the task reference will be available in the fitness function. It is indicated as the second input in the definition of the fitness function handle on line 3 of Listing 9.3. Also, the reference to the FOPS object is passed to the task definition as the task's *userData*. Then, the task is added to the FOPS instance and solved using the single-objective Particle Swarm Optimization algorithm.

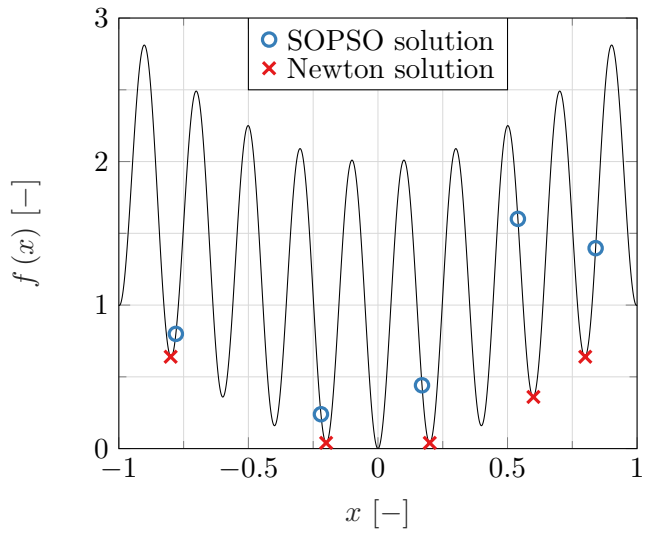


Figure 9.3: SOPSO solutions updated by performing a nested, local Newton optimization.

Listing 9.3: Benchmark problem definition.

```
fops = FOPS(false);
problem.limits =  [-1; 1];
problem.fitness = @(x, task) getF(x, task);
problem.fullControl = true;
problem.isVectorized = true;
settings = struct('nAgents', 5, 'nIters', 10);
fops.addTask(problem, 'SOPSO', settings, 'taskFC', fops);
fops.runTask('taskFC')
```

Listing 9.4: Fitness function definition.

```
1  function f = getF(vars, task)
2  fops = task.userData{1};
3  nAgents = size(vars, 1);
4  fFun = @(x) 1 - cos(pi/2*20*x) + x.^2;
5  f = fFun(vars);
6  for iA = 1:nAgents
7     problem = struct('limits', [-1; 1], ...
8        'initialPosition', vars(iA), 'fitness', fFun);
9     settings = struct('nIters', 10, 'nAgents', 1, 'diff', 1e-4);
10    fops.addTask(problem, 'SONEW', settings, 'localTask')
11    fops.runTask('localTask');
12    res = fops.tasks(2).results;
13    fops.deleteTask('localTask');
14    if res.fitness(end)< f(iA)
15       task.agents(iA).position = res.position;
16    end
17 end
18 end
```

More interesting is the content of the `getF` fitness function shown in Listing 9.4. Fitness function is evaluated for all agents at line 5. Then we loop over all agents. A new task using the Newton method `'SONEW'` [6] is created and solved for every proposed solution. The initial guess of the Newton method is set to the position found by the SOPSO algorithm. The partial derivatives needed for the Newton method are found using the central differences with the step defined in line 9 with the field `'diff'`. After the Newton method finds the local minimum, the position of the corresponding particle of the PSO task is updated if the fitness function was improved. An improvement of the fitness function is shown in Figure 9.3. The initial agent positions found by the PSO are depicted with blue circles, while their updated positions are highlighted with red crosses. Thanks to the nested local optimization, the PSO algorithms move the particles only in a sub-space of local minima. Therefore, the convergence rate can be significantly increased.

9.3 Optimal Placement of Transmitters

This example is defined as a multi-objective problem with a variable number of dimensions. It means that the optimizer works with decision space vectors of different lengths simultaneously according to the number of transmitters used. The optimization problem is defined using three objectives:

$$f_1(\mathbf{x}) = -\bigcup_{i=1}^{N_T} s_i, \tag{9.9}$$

$$f_2(\mathbf{x}) = \bigcap_{i=1}^{N_T} s_i + \sum_{i=1}^{N_T} (o_i)\,, \tag{9.10}$$

$$f_3(\mathbf{x}) = \sum_{i=1}^{N_T} (1 + r_i)\,. \tag{9.11}$$

where $\mathbf{x}$ is the decision space vector of size $3 \times N_T$:

$$\mathbf{x} = \{\mathbf{c}_1, r_1, \mathbf{c}_2, r_2, ..., \mathbf{c}_{N_T}, r_{N_T}\}\,. \tag{9.12}$$

Here, the parameter N_T stands for the number of transmitters used, s_i is the area covered by the i-th transmitter, o_i is the area covered outside the area of interest by the i-th transmitter (the waste of power), and $\mathbf{c}_i$ denotes the position vector of the i-th transmitter. For the sake of simplicity, we assume that the area covered by a single transmitter is circular with a radius r_i.

Equations (9.9) – (9.11) define individual conflicting objectives put on the design. The fitness function f_1 maximizes the area that is covered by all N_T transmitters (negatively taken union of the covered area). Objective f_2 minimizes the amount of wasted power by means of reducing overlapping area and area covered outside the area of interest. Finally, the function f_3 minimizes the costs of the design. The cost of every transmitter consists of a fixed part and a part corresponding to the amount of its power. This favors using a transmitter with higher power rather than more transmitters with lower power covering the same area. The transmitter can be placed on any point of the area of interest and radius (power) is chosen from the interval $[0.1a, 0.5a]$, where a is the side of the squared area of interest.

The problem definition is shown in Listing 9.5. Fitness values are obtained by using the function handle `@(x) transmitterFitness(x)`. The `plotTransmitters` function visualizes a single solution found by the algorithm, as shown in Figure 9.4. The content of the `plotTransmitters` function is shown in Listing 9.6.

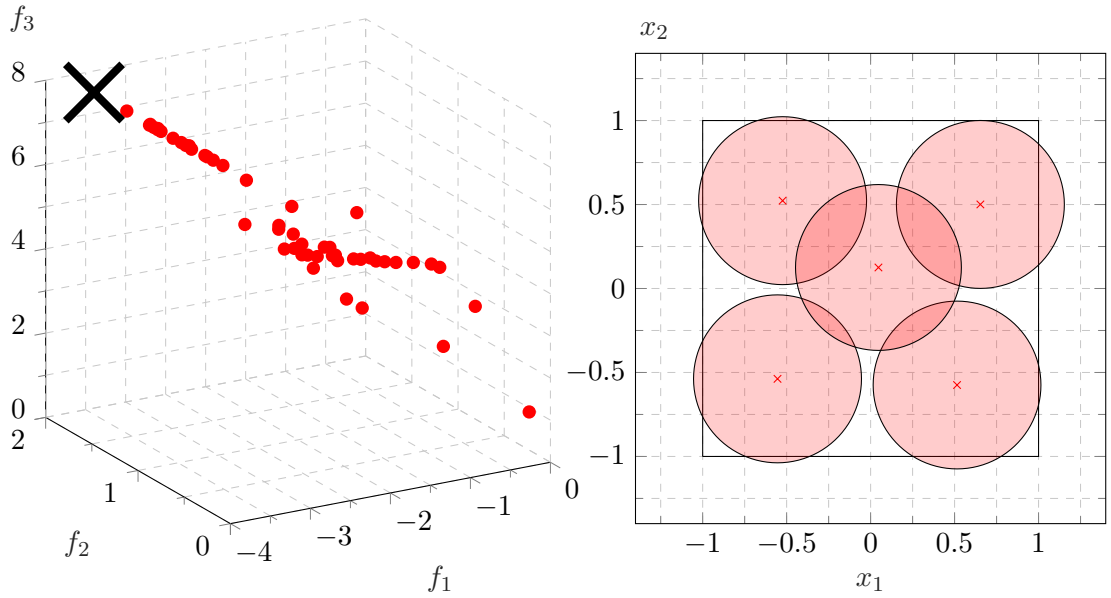


Figure 9.4: Interactive plot of the Pareto-front with the system response for the transmitter placement problem.

Listing 9.5: Optimal placement of transmitters definition.

```
function problem = VNDMOTransmitters
nVarsList = 3:3:15;
nMax = max(nVarsList)/3;
limits = [repmat([-1, -1, 0.1], 1, nMax); ...
          repmat([1,   1, 0.5], 1, nMax)];
fitness  = @(x) transmitterFitness(x);
systemPlot = @(x, f) plotTransmitters(x, f);
problem = struct( 'limits', limits, ...
                  'fitness', fitness, ...
                  'isVectorized', false, ...
                  'nVarsList', nVarsList, ...
                  'systemResponse', systemPlot);
end
```

Listing 9.6: Transmitters placement: system response plot.

```
function plotTransmitters(result, fitness)
polygon = [-1, -1; 1, -1; 1, 1; -1, 1; -1 -1];
center = [result(1,1:3:end)', result(1, 2:3:end)'];
radius = result(1, 3:3:end)';
phi = linspace(0,2*pi, 101)';
plot(polygon(:,1), polygon(:,2), 'k')
hold on
for iC = 1:size(radius, 1)
   pts = [center(iC,1) + radius(iC)*cos(phi), ...
      center(iC,2) + radius(iC)*sin(phi)];
   fill(pts(:,1), pts(:,2), 'r', 'FaceAlpha', 0.2)
   plot(center(iC,1), circle.center(iC,2), 'rx')
end
```

The session of the FOPS software for solving the transmitter problem with the VND variant of the GDE3 [10] algorithm is shown in Listing 9.7. The algorithm is set with default properties (scaling factor $F = 0.2$, probability of crossover $P_{\mathrm{C}} = 0.2$ and probabilities to follow P_{DT}). The problem is solved using 40 agents and 200 iterations.

Listing 9.7: The FOPS session for the transmitter design problem.

```
fops = FOPS(true);
set = struct('nAgents', 50, 'nIters', 200);
fops.addTask('VNDMOTransmitters', 'VNDGDE3', set, 'VNDTask')
fops.runTask('VNDTask')
```

As can be seen in Table 9.1, the Pareto-front is built by decision space vectors of different sizes (e.g., the first agent is built using only one transmitter). Trade-off solutions found by the algorithm are visualized in Figure 9.4: the Pareto-front members on the left and the system response on the right. Note that the system response is a feature of the FOPS toolbox. After the user picks any point from the left sub-figure with a mouse click, it is highlighted with a black cross marker, and the system response plot is updated according to the picked solution using the user-defined function (in this case the `plotTransmitters` function from Listing 9.6). This feature makes it easier to browse the Pareto-front and decide on the final trade-off solution.

As can be seen from the system responses of selected solutions, as depicted in Figure 9.4, the individual objectives are respected. The plot sequence shows different trade-off solutions. The sequence starts with an extreme solution where most of the area of interest is covered by the signal, but the amount of the overlapping area is high. The following plots show trade-off solutions built by a different number of transmitters until another extreme solution is reached. Such a scheme offers minimal overlaps, but the coverage of the area is poor.

Table 9.1: Optimal positions of chosen transmitters.

Agent	N_T	x_1	y_1	r_1	x_2	y_2	r_2	x_3	y_3	r_3	x_4	y_4	r_4	x_5	y_5	r_5
1	1	0.08	-0.37	0.29												
2	5	-0.55	-0.50	0.50	-0.52	0.52	0.49	0.51	-0.57	0.49	0.65	0.51	0.50	0.04	0.12	0.49
5	5	-0.61	-0.49	0.50	-0.49	0.54	0.50	0.52	-0.50	0.49	0.52	0.49	0.50	-0.06	-0.16	0.43
6	5	-0.55	-0.53	0.50	-0.52	0.52	0.50	0.51	-0.57	0.49	0.65	0.50	0.50	0.04	0.12	0.49
7	4	-0.50	-0.50	0.24	-0.49	0.52	0.46	0.48	-0.49	0.42	0.51	0.49	0.10			
8	5	-0.61	-0.50	0.50	-0.51	0.51	0.49	0.52	-0.50	0.49	0.52	0.49	0.50	-0.06	-0.16	0.43
11	3	-0.47	-0.45	0.50	-0.44	0.52	0.50	0.43	-0.11	0.49						
13	3	-0.63	-0.50	0.50	-0.55	0.52	0.50	0.48	0.03	0.42						
14	4	-0.52	-0.49	0.34	-0.49	0.47	0.49	0.34	-0.49	0.50	0.45	0.56	0.50			
16	5	-0.53	-0.50	0.50	-0.52	0.52	0.49	0.51	-0.50	0.49	0.67	0.51	0.50	0.04	0.12	0.49
19	4	-0.50	-0.48	0.49	-0.44	0.48	0.49	0.43	-0.48	0.49	0.46	0.49	0.50			
26	5	-0.52	-0.51	0.49	-0.49	0.53	0.50	0.48	-0.49	0.49	0.49	0.49	0.50	0.04	0.13	0.18
27	4	-0.50	-0.49	0.20	-0.49	0.53	0.50	0.48	-0.49	0.49	0.53	0.49	0.50			
29	4	-0.50	-0.40	0.50	-0.50	0.47	0.46	0.50	-0.49	0.50	0.46	0.50	0.50			
30	4	-0.51	-0.51	0.50	-0.50	0.46	0.49	0.50	-0.50	0.49	0.46	0.49	0.50			
32	4	-0.52	-0.49	0.46	-0.49	0.51	0.47	0.47	-0.47	0.49	0.44	0.49	0.50			
33	5	-0.50	-0.50	0.50	-0.48	0.51	0.49	0.52	-0.51	0.49	0.52	0.49	0.50	-0.06	-0.16	0.43
34	3	-0.47	-0.48	0.50	-0.50	0.52	0.50	0.43	-0.11	0.49						
35	5	-0.53	-0.49	0.50	-0.55	0.52	0.49	0.51	-0.50	0.49	0.67	0.51	0.50	0.04	0.12	0.49
36	4	-0.61	-0.47	0.43	-0.49	0.47	0.45	0.34	-0.51	0.50	0.45	0.49	0.50			
39	2	0.08	-0.37	0.29	-0.52	0.52	0.49									
40	3	0.08	-0.37	0.29	0.17	-0.60	0.16	0.47	0.03	0.42						
41	5	-0.52	-0.49	0.20	-0.49	0.40	0.47	0.47	-0.47	0.49	0.44	0.49	0.50	-0.25	-0.22	0.12
46	3	0.08	-0.37	0.29	-0.52	0.52	0.49	0.51	-0.57	0.49						
49	4	-0.50	-0.50	0.49	-0.49	0.47	0.34	0.43	-0.40	0.49	0.52	0.49	0.10			
50	4	-0.52	-0.47	0.45	-0.49	0.46	0.45	0.34	-0.51	0.50	0.41	0.49	0.50			

9.3.1 Linear Antenna Array Design

This section aims to delineate the difference between problem formulation with a fixed number of dimensions and a variable number of dimensions. A linear antenna array consists of ν antennas distributed alongside the x-axis. In this particular example, all the constituent antennas are identical. Therefore, the formulation of the total radiation vector of the antenna array is simplified to [90]:

$$\mathbf{F}_{\text{tot}}(\mathbf{k}) = \mathbf{A}(\mathbf{k})\,\mathbf{F}(\mathbf{k}), \tag{9.13}$$

where $\mathbf{k} = k\mathbf{r}$ is the wave number vector, k is the free space wave number, and $\mathbf{r}$ is the position vector. The radiation vector of an elementary antenna $\mathbf{F}(\mathbf{k})$ is multiplied by the array factor $\boldsymbol{A}(\mathbf{k})$. The array factor is determined by the array configuration, and it is defined as:

$$\mathbf{A}(\mathbf{k}) = \sum_{i=1}^{N} a_i \exp(j\mathbf{k}\cdot\mathbf{d}_i). \tag{9.14}$$

Here, j is the imaginary unit, a_i is the complex number representing excitation amplitude and phase, and $\mathbf{d}_i$ is the position of the i-th antenna.

Antenna array properties that are of interest in this study are the Side-Lobe Level (SLL) and the number of antennas ν. The problem formulation for a fixed number of decision variables commonly utilizes the uniform grid. If the uniform grid is used, a particular antenna is activated or deactivated according to the decision vector ([91, 92]). Therefore, the distribution grid of antennas has to be determined a priori, and it might affect the overall performance of the optimization. Contrarily, the VND formulation of the problem allows an algorithm to not only find the proper number of antennas but to find the optimal positioning on the x-axis as well.

The multi-objective problem using the uniform grid is formulated as follows:

$$f_1 = \text{SSL}(\mathbf{x}) \tag{9.15}$$

$$f_2 = \nu(\mathbf{x}), \tag{9.16}$$

where both objectives are to be minimized, and the $\mathbf{x}$ is the decision vector. In the case of a uniform grid, the decision vector is a binary string of fixed length equal to the possible number of antennas. In our comparative study, the array can contain up to 100 antennas, and the gap between two grid positions is one-quarter of the wavelength ($0.25\,\lambda_0$).

The variable number of dimensions formulation gives:

$$f_1 = \text{SSL}(\mathbf{x}, n) \tag{9.17}$$

$$f_2 = \nu(\mathbf{x}, n), \tag{9.18}$$

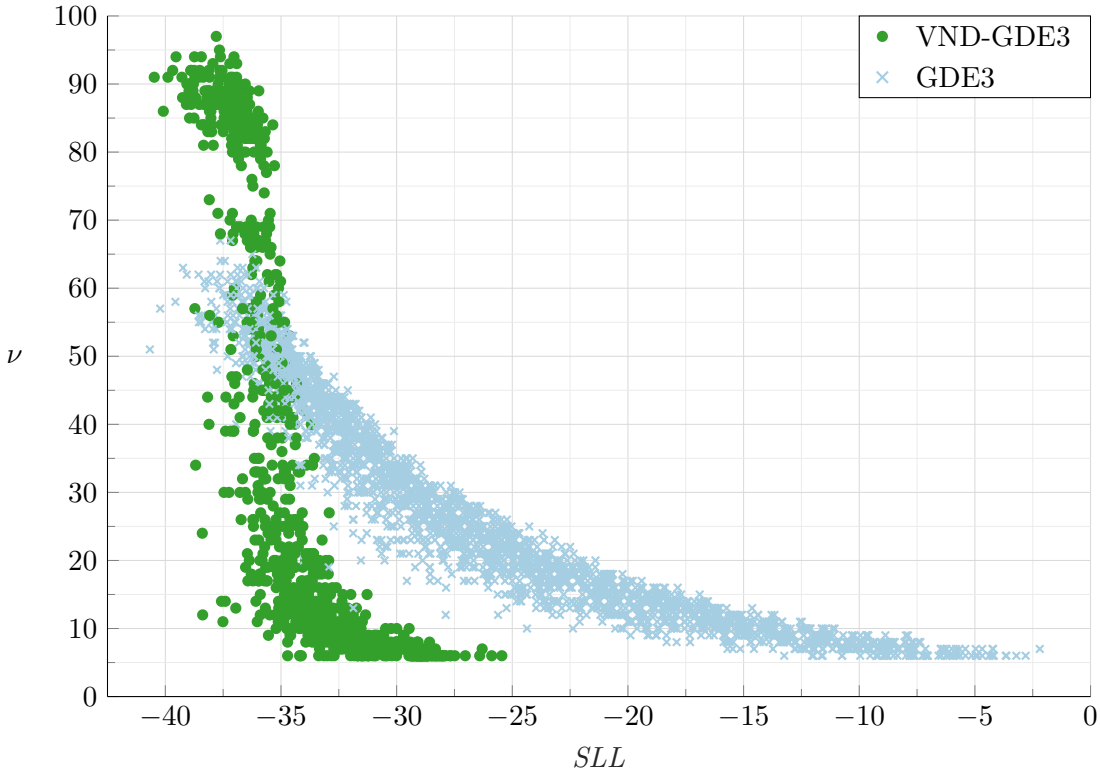


Figure 9.5: Comparison of VND-GDE3 (VND formulation) and GDE3 (uniform grid formulation) algorithms on the linear antenna array problem.

where $\mathbf{x}$ is the vector of n values expressing the gaps between consecutive elements of the array. Note that the first antenna of the array is placed at $x = 0\,\mathrm{m}$. The gap between individual elements can vary according to the interval of $x_i \in \langle 0.25\,\lambda_0, \lambda_0 \rangle$. The number of elements in the antenna array is $\nu = n + 1$.

The problem with the uniform grid was optimized by the standard GDE3 algorithm with default properties as defined in [MM4]. The VND formulation of the problem was optimized with the VND-GDE3 algorithm with default settings (i.e. SET A.3). Both algorithms used 200 agents over 200 iterations. Results shown in Figure 9.5 accumulates 100 independent runs. The solutions marked with "×" signs belong to the uniform grid formulation and the solutions marked with "•" signs belong to the VND formulation. The VND-GDE3 algorithm outperforms the standard GDE3 algorithm, especially from the viewpoint of the number of the used antennas. The only drawback of the VND-GDE3 method is that the average number of non-dominated solutions found per one run is lower ($N = 10.62$) than that of standard GDE3 ($N = 24.88$).

9.4 Synthesis of the Digital Circuits

This section describes the design of digital circuits using evolutionary algorithms. The design of digital circuits is generally a complex task [93]. A well-known method of simplifying boolean algebra expression is to use so-called Karnaugh maps [94]. However, this method can reasonably be used for expressions with no more than six input variables. The Quine-McCluskey algorithm (QMC [95]) is in principle similar to the Karnaugh maps, but it is more efficient for use in a computer algorithm. According to [96], the QMC algorithm is an NP-complete problem. Therefore, its computational complexity grows exponentially with the number of input variables.

The desired digital circuit is defined by its truth table. The truth table defines the output value of the circuit for any combination of input variables. The truth table is of length 2^{N_I}, where N_I is the number of input variables of the digital circuit. The truth table of a simple digital circuit (Multiplexer with 3 input variables) is shown in Table 9.2. An exhaustive expression of the three-input multiplexer is:

$$y = \overline{x_2}\,\overline{x_1}x_0 + \overline{x_2}x_1x_0 + x_2x_1\overline{x_0} + x_2x_1x_0. \tag{9.19}$$

This expression uses a sum of four products. The number of products corresponds to the number of logical ones in the y column of the truth table (Table 9.2). However, the exhaustive expression can be simplified into an expression:

$$y = \overline{x_2}x_0 + x_2x_1. \tag{9.20}$$

The simplification task becomes more complex as the number of input variables of the digital circuit grows (N_I).

A many-input variables digital circuit can be synthesized by using an optimization algorithm. The formulation of the problem is simple. The optimization algorithm proposes an expression (sum of products – SOP), and its solution ($\mathbf{y}_p$) is compared to the truth table ($\mathbf{y}_{\text{TT}}$) in the fitness function. However, the problem has a variable number of dimensions because the number of summed products (D) in an expression can not be defined a priori

Table 9.2: Truth table of 3-input multiplexer.

x_2	x_1	x_0	y
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

for an unknown digital circuit. Fitness function for the synthesis of the digital circuit can be expressed by:

$$f_1 = \sum_{i}^{2^{N_I}} E_i, \tag{9.21}$$

$$f_2 = D, \tag{9.22}$$

$$E_i = \begin{cases} 1 & \text{if} \quad y_{p,i} \neq y_{\text{TT},i} \\ 0 & \text{if} \quad y_{p,i} = y_{\text{TT},i}. \end{cases} \tag{9.23}$$

Note that the first fitness function compares the two vectors $\mathbf{y}_p$ and $\mathbf{y}_{\text{TT}}$ and counts the number of differences between them.

Two digital circuits are synthesized in this section – the 6-input multiplexer (MUX6) and the 11-input multiplexer (MUX11). The truth table of MUX6 has 64 combinations (see Table 9.3). The exhaustive expression contains a sum of 32 products. However, the simplified expression is:

$$y = \overline{x_5 x_4} x_0 + \overline{x_5} x_4 x_1 + x_5 \overline{x_4} x_2 + x_5 x_4 x_3. \tag{9.24}$$

As we can see, the simplified expression uses only four products.

The truth table of the11-input multiplexer has 2048 combinations. Again, half of them result in logical ones. Therefore, the exhaustive expression is a sum of 1024 products. The simplified solution is:

$$\begin{aligned} y = &\overline{x_{10} x_9 x_8} x_0 + \overline{x_{10} x_9} x_8 x_1 + \overline{x_{10}} x_9 \overline{x_8} x_2 + \overline{x_{10}} x_9 x_8 x_3 + \\ &+ x_{10} \overline{x_9 x_8} x_4 + x_{10} \overline{x_9} x_8 x_5 + x_{10} x_9 \overline{x_8} x_6 + x_{10} x_9 x_8 x_7. \end{aligned} \tag{9.25}$$

The boolean algebra expression is encoded as a vector of integer decision variables, where each variable corresponds to one product term in an SOP. The number of decision variables can vary. The range of each decision variable depends on the number of input variables of the digital circuit. Each input variable in the product term can be set, unset, or missing. Therefore, a ternary representation is used. Table 9.4 shows an example of ternary representation on a minimal expression of MUX3. A decision vector of boolean algebra expression $y = \overline{x_2} x_0 + x_2 x_1$ is $\mathbf{x} = \{7, 14\}$.

9.4.1 Experiments

This subsection describes the comparative study where the 3-input, 6-input, and 11-input multiplexers are synthesized with evolutionary algorithms. The standard GDE3 algorithm is compared to the VND-GDE3 algorithm. The ternary representation enables us to exploit both algorithms – with a fixed number of dimensions, and a variable number of dimensions – without changes.

The only difference is that the VND algorithm calculates with SOPs with the number of product terms $D = \{1, 2, \ldots, D_{\max}\}$, while the GDE3 works with SOPs with $D_{\max}$. A

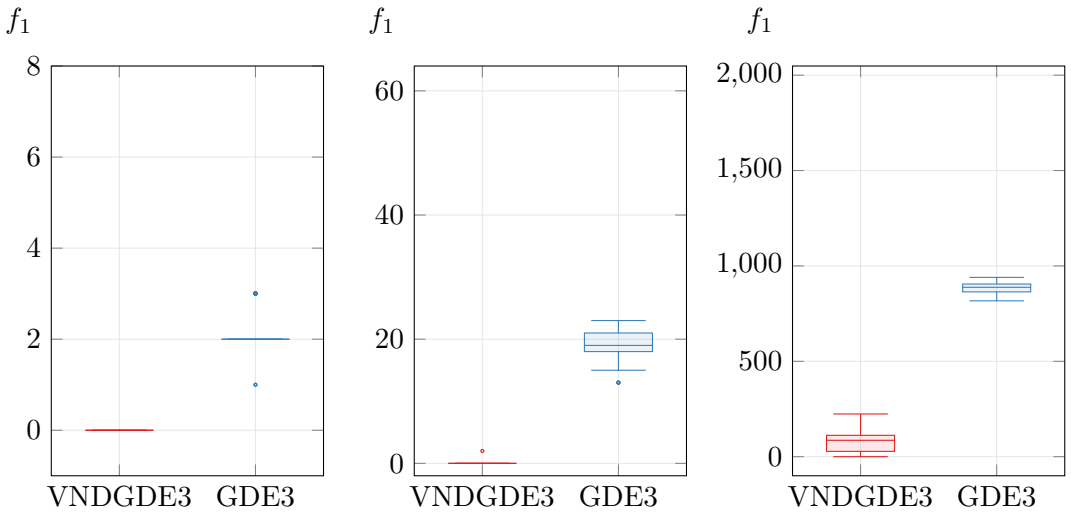
lower number of product terms can be reached if the decision variable (representing one product term) contains such a value that represents all twos after ternary conversion (c.f. Table 9.4).

Table 9.3: Truth table of 6-input multiplexer.

Index	x_5	x_4	x_3	x_2	x_1	x_0	y	**Index**	x_5	x_4	x_3	x_2	x_1	x_0	y
1	0	0	0	0	0	0	0	33	1	0	0	0	0	0	0
2	0	0	0	0	0	1	1	34	1	0	0	0	0	1	0
3	0	0	0	0	1	0	0	35	1	0	0	0	1	0	0
4	0	0	0	0	1	1	1	36	1	0	0	0	1	1	0
5	0	0	0	1	0	0	0	37	1	0	0	1	0	0	1
6	0	0	0	1	0	1	1	38	1	0	0	1	0	1	1
7	0	0	0	1	1	0	0	39	1	0	0	1	1	0	1
8	0	0	0	1	1	1	1	40	1	0	0	1	1	1	1
9	0	0	1	0	0	0	0	41	1	0	1	0	0	0	0
10	0	0	1	0	0	1	1	42	1	0	1	0	0	1	0
11	0	0	1	0	1	0	0	43	1	0	1	0	1	0	0
12	0	0	1	0	1	1	1	44	1	0	1	0	1	1	0
13	0	0	1	1	0	0	0	45	1	0	1	1	0	0	1
14	0	0	1	1	0	1	1	46	1	0	1	1	0	1	1
15	0	0	1	1	1	0	0	47	1	0	1	1	1	0	1
16	0	0	1	1	1	1	1	48	1	0	1	1	1	1	1
17	0	1	0	0	0	0	0	49	1	1	0	0	0	0	0
18	0	1	0	0	0	1	0	50	1	1	0	0	0	1	0
19	0	1	0	0	1	0	1	51	1	1	0	0	1	0	0
20	0	1	0	0	1	1	1	52	1	1	0	0	1	1	0
21	0	1	0	1	0	0	0	53	1	1	0	1	0	0	0
22	0	1	0	1	0	1	0	54	1	1	0	1	0	1	0
23	0	1	0	1	1	0	1	55	1	1	0	1	1	0	0
24	0	1	0	1	1	1	1	56	1	1	0	1	1	1	0
25	0	1	1	0	0	0	0	57	1	1	1	0	0	0	1
26	0	1	1	0	0	1	0	58	1	1	1	0	0	1	1
27	0	1	1	0	1	0	1	59	1	1	1	0	1	0	1
28	0	1	1	0	1	1	1	60	1	1	1	0	1	1	1
29	0	1	1	1	0	0	0	61	1	1	1	1	0	0	1
30	0	1	1	1	0	1	0	62	1	1	1	1	0	1	1
31	0	1	1	1	1	0	1	63	1	1	1	1	1	0	1
32	0	1	1	1	1	1	1	64	1	1	1	1	1	1	1

Table 9.4: An example of ternary representation of MUX3.

y =	$\overline{x_2}$		x_0	+	x_2	x_1	
	0	2	1		1	1	2
	$0 \cdot 3^2$	$2 \cdot 3^1$	$1 \cdot 3^0$		$1 \cdot 3^2$	$1 \cdot 3^0$	$2 \cdot 3^0$
		7				14	



(a) MUX3. **(b)** MUX6. **(c)** MUX11.

Figure 9.6: Standard boxplots of f_1 on the comparison of VND-GDE3 and GDE3.

The maximal number of product terms is $D_{\max} = 10$ in the case of 3- and 6-input multiplexer, and $D_{\max} = 20$ in the case of the 11-input multiplexer. All three problems exploited 100 agents. The scaling factor was $F = 0.02$, and the probability of crossover was $P_C = 0.02$ in both variants of the GDE3 algorithm. These values were established experimentally. Such small values are beneficial in the case of a digital circuit design problem. The main difficulty of such a problem is that the slightest change in the decision vector can lead to a great change of a fitness value. The probability of dimension transition of the VND-GDE3 algorithm was set to $P_{\mathrm{DT}} = 0.35$. The number of iterations used was $G = 100$ for the 3-input multiplexer, $G = 1000$ for the 6-input multiplexer, and $G = 10000$

Table 9.5: Boolean algebra expressions proposed by VND-GDE3 algorithm and how often the corresponding expression was found in 100 runs.

N_I	Decision vector	Boolean algebra expression	Count
3	$[7, 14]$	$\overline{x_2}x_0 + x_2x_1$	100
6	$[79, 158, 314, 377]$	$\overline{x_5}\overline{x_4}x_0 + \overline{x_5}x_4x_1 + x_5\overline{x_4}x_2 + x_5x_4x_3$	62
3	$[61, 158, 314, 377, 556]$	$\overline{x_5}\overline{x_4}\overline{x_2}x_0 + \overline{x_5}x_4x_1 + x_5\overline{x_4}x_2 + x_5x_4x_3 + \overline{x_4}x_2x_0$	4
6	$[79, 158, 314, 377, 617]$	$\overline{x_5}\overline{x_4}x_0 + \overline{x_5}x_4x_1 + x_5\overline{x_4}x_2 + x_5x_4x_3 + x_4x_3x_1$	2
6	$[73, 76, 158, 314, 377]$	$\overline{x_5}\overline{x_4}x_1x_0 + \overline{x_5}\overline{x_4}x_1x_0 + \overline{x_5}x_4x_1 + x_5\overline{x_4}x_2 + x_5x_4x_3$	2
6	$[79, 158, 314, 371, 617]$	$\overline{x_5}\overline{x_4}x_0 + \overline{x_5}x_4x_1 + x_5\overline{x_4}x_2 + x_5x_4x_3\overline{x_1} + x_4x_3x_1$	2
11	$[6559, 13118, 26234, \ldots$ $32777, 65528, 71927, \ldots$ $84563, 89666, 95498]$	$\overline{x_{10}}\overline{x_9}\overline{x_8}x_0 + \overline{x_{10}}\overline{x_9}x_8x_1 + \overline{x_{10}}x_9\overline{x_8}x_2 +$ $+\overline{x_{10}}x_9x_8x_3 + x_{10}\overline{x_9}\overline{x_8}x_4 + x_{10}\overline{x_9}x_8x_5 +$ $x_{10}x_9\overline{x_8}x_6 + x_{10}x_9x_8x_7 + x_{10}x_9x_7x_6$	1
11	$[6559, 13118, 26232\ldots$ $32777, 45916, 65528\ldots$ $71927, 84563, 89180\ldots$ $89360, 109106]$	$\overline{x_{10}}\overline{x_9}\overline{x_8}x_0 + \overline{x_{10}}\overline{x_9}x_8x_1 + \overline{x_{10}}x_9\overline{x_8}x_2\overline{x_0} +$ $+\overline{x_{10}}x_9x_8x_3 + \overline{x_{10}}\overline{x_8}x_2x_0 + x_{10}\overline{x_9}\overline{x_8}x_4 +$ $+x_{10}\overline{x_9}x_8x_5 + x_{10}x_9\overline{x_8}x_6 + x_{10}x_9x_8x_7\overline{x_5} +$ $+x_{10}x_9x_8x_7x_5\overline{x_3}x_2 + x_{10}x_8x_7x_5$	1

for the 11-input multiplexer. Each simulation was repeated 100 times.

Figure 9.6 shows the standard boxplots of the first fitness function – the number of input combinations violating the truth table. Note that the problem formulates the second fitness function as the number of SOPs. Therefore, if the optimizer finds a solution that corresponds to the truth table, but the number of product terms is sub-optimal, the chance of finding a solution with a lower number of SOPs is greater compared to the single-objective approach. It is evident from the boxplots that the VND formulation performed much better compared to the fixed-length formulation of the problem.

The synthesis of the 3-input multiplexer by the VND-GDE3 algorithm was successful in all attempts. On the other hand, the GDE3 algorithm was unable to find the correct solution (expression with only one error was found twice, two-errors expression was found 80-times, and the rest of the solutions contained 3 errors). In the case of the 6-input multiplexer, the VND-GDE3 algorithm found expressions that meet the truth table in 99 tries out of 100. On the other hand, the best expression proposed by the GDE3 algorithm contained 13 errors compared to the truth table. The difference is even more dramatic in the case of the 11-input multiplexer. The VND-GDE3 algorithm found expressions without errors exactly 10-times. Contrarily, the GDE3 algorithm has not found a solution even remotely close to the truth table.

Table 9.5 shows expressions found by the VND-GDE3 algorithm for MUX3, MUX6, and MUX11. The decision vector column contains integer values that are translated into boolean algebra expression. The optimal MUX3 expression was found in all attempts by the VND-GDE3 algorithm. In the case of the 6-input multiplexer, the VND-GDE3 algorithm found 62-times the optimal solution (with only four product terms). Other solutions had a sub-optimal number of product terms. However, the truth table was met (the first fitness value is zero) by all runs except one. Contrarily, the best solution of the 11-input multiplexer found by the VND-GDE3 algorithm has nine product terms. However, the optimal solution (see (9.25)) has only eight product terms. Note that ten different solutions were meeting the truth table of MUX11, but only two of them are shown in the table.

9.5 Image Thresholding Problem

Image thresholding is the simplest method for digital image segmentation. The idea of image thresholding is to find the optimal threshold value that divides the gray-level image into "object" pixels (gray level is greater than the threshold value) and "background" pixels (gray level is lower than the threshold value) [97].

Examples of thresholding applications can be a document image analysis (Optical Character Recognition – OCR) [99], segmentation of thermal images [100], x-ray computed tomography (CAT) [101] or license plate image recognition [102].

The most famous automated image thresholding algorithm is Otsu's method [103]. This method is a histogram shape-based method. The method assumes two distinct peaks in the histogram. Therefore, the threshold value separates the two classes of gray-levels so that the intra-class variance is minimized. Figure 9.7a shows a widely used thresholding testing image, and Figure 9.7b shows its histogram.

Otsu's method maximizes the inter-class variance defined as:

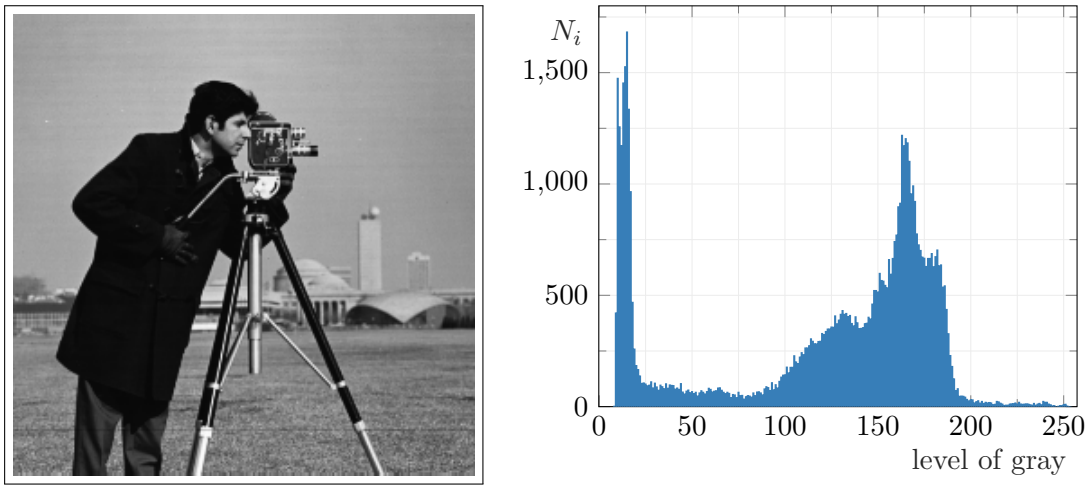
$$\sigma_w^2(t) = \omega_0(t)\,\sigma_0^2(t) + \omega_1(t)\,\sigma_1^2(t), \tag{9.26}$$

where ω_0 and ω_1 are probabilities of the two classes separated by the threshold t, and $\sigma_0^2(t)$ and $\sigma_1^2(t)$ are variances of these two classes.

The probabilities are calculated from histogram using:

$$\omega_0(t) = \sum_{i=0}^{t-1} p(i), \tag{9.27}$$

$$\omega_1(t) = \sum_{i=t}^{L-1} p(i), \tag{9.28}$$



(a) Greyscale original. **(b)** Histogram of testing image.

Figure 9.7: Cameraman testing image [98].

where $p(i)$ is calculated from histogram's bins $h(i)$ divided by the overall number of pixels in figure $\sum_{i=0}^{L} h$, and L is the number of bins in the histogram.

The inter-class variance for a given threshold t is calculated by the equation:

$$\sigma_b^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_t), \tag{9.29}$$

where:

$$\mu_0(t) = \frac{\sum_{i=0}^{t-1} ip(i)}{\omega_0(t)}, \tag{9.30}$$

$$\mu_0(t) = \frac{\sum_{i=t}^{L-1} ip(i)}{\omega_1(t)}, \tag{9.31}$$

and

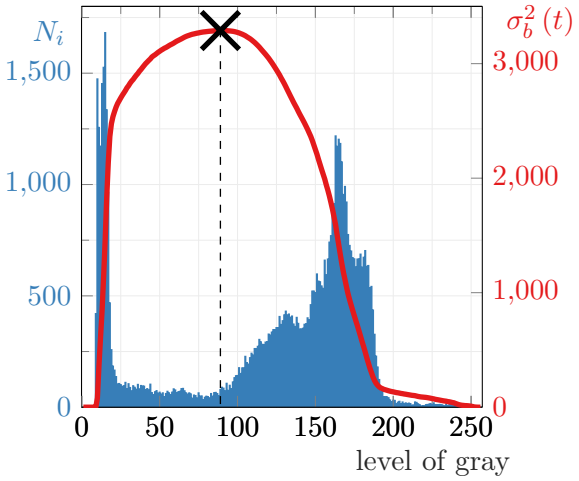
$$\mu_T = \sum_{i=0}^{L-1} ip(i). \tag{9.32}$$

Otsu's method calculates the inter-class variance exhaustively. Figure 9.8b shows the histogram of the cameraman testing image (blue bar graph) and the corresponding values of inter-class variance (red line). The black cross marker shows the threshold value found by Otsu's method – $t = 89$. Figure 9.8a shows the cameraman testing image translated into a binary image using Otsu's threshold.

Otsu's method is widely used in many applications. However, there are thresholding tasks where multiple thresholds are sought. Therefore, if we want to find two threshold values, equation (9.29) is modified into:

$$\sigma_b^2(t) = \omega_0(\mu_0 - \mu_T)^2 + \omega_1(\mu_1 - \mu_t) + \omega_2(\mu_2 - \mu_t), \tag{9.33}$$

(a) Thresholded cameraman image.



(b) Histogram of the cameraman image (blue) with inter-class variance values found by Otsu's method (red).

Figure 9.8: Otsu's thresholding method explained on the cameraman testing image.

where indices in sums in ω_0 and μ_0 calculations ranges from $i = 0$ to $t_1 - 1$, for ω_1 and μ_1 from t_1 to $t_2 - 1$, and for ω_2 and μ_2 from t_2 to $L - 1$.

For more thresholds, the equation can be easily scaled. However, if one threshold is sought, the inter-class variance has to be computed for each possible threshold, i.e 254 thresholds. However, if there is more than one threshold, the inter-class variance has to be calculated for each combination of thresholds, i.e. k-permutation of L threshold values, and k marks the number of thresholds:

$$P(L, k) = \frac{L!}{(L-k)!}. \tag{9.34}$$

Concretely, 64 262 inter-class variance calculations to find two thresholds, but 4 064 700 024 calculations to find four thresholds.

The motivation for using an evolutionary algorithm to solve a multi-threshold thresholding problem is evident. The problem is formulated as a multi-objective one with a

(a) Cameraman testing image [98].

(b) Sailboat testing image [104].

(c) Female testing image [104].

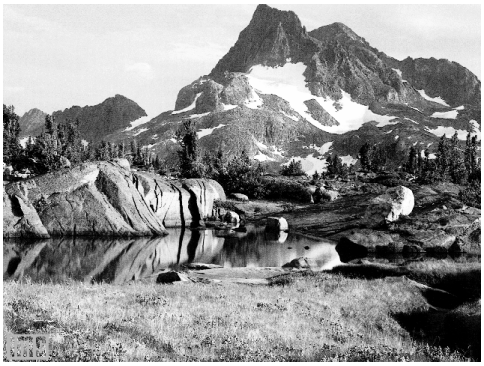
(d) Airplane testing image [104].

Figure 9.9: Thresholding testing images [104].

(e) Boat testing image [104].

(f) Fingerprint testing image [105].

(g) San Diego testing image [104].



(h) Mountain testing image [98].

Figure 9.9: Thresholding testing images.

variable number of dimensions where the value of Otsu's inter-class variance is used as the first fitness function. The number of thresholds stands for the second fitness function. The advantage of using a multi-objective formulation of the problem is that the algorithm proposes a set of solutions with a different number of thresholds. Therefore, the user can select the solution that suits the best to his needs without setting the number of thresholds a priori. Note that the variability of the number of thresholds is in accordance with the VND formulation.

9.5.1 Exhaustive vs. Evolutionary Approach

The comparative study of the standard (exhaustive) and evolutionary approach is presented. The exhaustive approach is represented by using Otsu's method for 1, 2, 3, and 4 thresholds. The evolutionary approach uses the single-objective Differential Evolution. This study is here to demonstrate the strength of the evolutionary approach, especially with more thresholds. Note that this study is inspired by [106].

Table 9.6 shows the results of the thresholding study with a fixed number of decision

variables. Threshold values can be seen for each testing images shown in Figures 9.9a – 9.9h. Results of exhaustive Otsu's method are shown in the third column, and they are considered as the reference. The fourth column shows threshold values proposed by the single-objective Differential Evolution. Note that each simulation was repeated 100 times. Therefore, the value shown in the fourth column is an average of 100 repetitions. Finally, the fifth column shows a standard deviation of each threshold value. Note that k denotes the number of thresholds.

Table 9.7 compares the computational time and inter-class variance for Otsu's method

Table 9.6: Thresholds found be the exhaustive Otsu's method and SODE algorithm with fixed number of dimensions.

Image	k	**Otsu**	**SODE** $-\ \mu$	**SODE** $-\ \sigma$
Cameraman	1	89	89	0
	2	[70, 144]	[70, 144]	[0, 0]
	3	[59, 119, 156]	[58.59, 119.0, 155.8]	[0.950, 1.203, 0.711]
	4	[42, 95, 140, 170]	[42.24, 94.59, 139.7, 169.9]	[2.173, 2.098, 1.087, 1.136]
Sailboat	1	126	126	0
	2	[86, 155]	[86, 155]	[0, 0]
	3	[80, 141, 195]	[79.60, 141, 195.0]	[0.600, 0.735, 0.546]
	4	[69, 112, 159, 199]	[68.61, 111.3, 158.9, 199.0]	[1.536, 2.387, 1.986, 1.445]
Female	1	74	74	0
	2	[58, 110]	[58, 110]	[0, 0]
	3	[34, 68, 114]	[34.09, 67.93, 114.2]	[0.567, 0.587, 0.812]
	4	[33, 61, 93, 129]	[32.67, 61.23, 94.91, 131.5]	[0.928, 1.654, 3.147, 3.892]
Airplane	1	153	153	0
	2	[112, 173]	[112, 173]	[0, 0]
	3	[92, 145, 191]	[92.21, 145.3, 191.3]	[1.291, 1.522, 1.067]
	4	[85, 130, 173, 203]	[84.88, 129.7, 172.8, 203.2]	[2.277, 2.998, 2.911, 1.495]
Boat	1	103	103	0.000
	2	[93, 155]	[93, 155]	[0, 0]
	3	[73, 126, 167]	[73.09, 126.2, 167.0]	[0.585, 0.444, 0.222]
	4	[65, 114, 147, 179]	[65.46, 113.7, 146.7, 179.2]	[1.374, 1.211, 0.890, 1.427]
Finger	1	144	143	0
	2	[115, 185]	[115, 184]	[0, 0]
	3	[99, 156, 201]	[98.59, 155.2, 199.9]	[0.750, 0.840, 0.725]
	4	[84, 131, 173, 209]	[84.22, 130.3, 171.9, 207.0]	[2.133, 2.388, 2.106, 1.628]
San Diego	1	140	140	0.000
	2	[107, 165]	[107, 165]	[0, 0]
	3	[91, 129, 176]	[91.39, 129.2, 176.5]	[0.6460, 0.8760, 0.8060]
	4	[79, 110, 144, 185]	[78.85, 110.3, 144.1, 184.7]	[2.224, 2.204, 2.236, 2.059]
Mountain	1	136	137.67	1.078
	2	[72, 168]	[73.92, 169.6]	[1.278, 1.059]
	3	[55, 122, 196]	[55.45, 125.1, 197.4]	[0.497, 1.792, 1.078]
	4	[46, 100, 149, 206]	[46.42, 102.2, 151.2, 207.6]	[2.164, 1.817, 1.800, 1.402]

and SODE optimization approach. Again, the table shows results of one run of the exhaustive method and 100 runs of the optimization algorithm, where the average value and the standard deviation is shown.

If we look at Table 9.6, we can see that the thresholds found by the evolutionary algorithm are very close to the thresholds found by the exhaustive method. The same applies to inter-class variance values. However, the computational time of the exhaustive method quickly grows with an increasing number of thresholds. Contrarily, the SODE

Table 9.7: Computational time and inter-class variance of the exhaustive Otsu's method and SODE algorithm with fixed number of dimensions.

		Computational time [s]			**Inter-class Variance**		
Image	k	**Otsu**	**SODE** $-\ \mu$	**SODE** $-\ \sigma$	**Otsu**	**SODE** $-\ \mu$	**SODE** $-\ \sigma$
Cameraman	1	0.0179	0.2801	0.0217	3289.1	3289.1	1.364E-12
	2	0.3087	0.3193	0.0161	3650.3	3650.3	8.640E-12
	3	30.367	0.3588	0.0237	3725.7	3725.7	5.880E-02
	4	2168.5	0.3801	0.0243	3780.7	3780.2	2.819E-01
Sailboat	1	0.0314	0.2796	0.0169	3620.4	3620.4	2.728E-12
	2	0.3030	0.3199	0.0158	3912.9	3912.9	5.002E-12
	3	30.109	0.3594	0.0238	4049.8	4049.8	9.302E-02
	4	2194.7	0.3801	0.0231	4117.3	4116.8	4.135E-01
Female	1	0.0128	0.2786	0.0147	1122.1	1122.1	2.046E-12
	2	0.2928	0.3216	0.0160	1318.6	1318.6	1.137E-12
	3	33.010	0.3601	0.0234	1417.4	1417.3	1.068E-01
	4	2036.0	0.3865	0.0251	1457.2	1456.6	3.558E-01
Airplane	1	0.0407	0.2815	0.0159	1773.1	1773.1	2.501E-12
	2	0.3066	0.3221	0.0149	1928.6	1928.6	2.274E-12
	3	31.795	0.3610	0.0243	2005.2	2005.1	9.235E-02
	4	2121.6	0.3863	0.0248	2050.4	2049.6	4.361E-01
Boat	1	0.0422	0.2801	0.0165	1617.7	1617.6	2.274E-13
	2	0.3136	0.3196	0.0180	1863.3	1863.3	3.411E-12
	3	37.030	0.3572	0.0239	1994.5	1994.4	3.149E-02
	4	2216.9	0.3845	0.0269	2059.9	2059.5	2.120E-01
Finger	1	0.0222	0.2781	0.0153	2478.1	2422.7	9.095E-13
	2	0.3211	0.3204	0.0146	2928.8	2855.7	5.457E-12
	3	29.765	0.3593	0.0251	3084.8	3003.9	9.646E-02
	4	2118.9	0.3852	0.0259	3156.3	3071.6	5.502E-01
San Diego	1	0.0259	0.2791	0.0158	1635.8	1635.8	2.956E-12
	2	0.3066	0.3201	0.0145	1989.9	1989.9	2.728E-12
	3	29.502	0.3586	0.0228	2103.8	2103.7	1.045E-01
	4	2006.9	0.3850	0.0258	2161.9	2161.3	3.439E-01
Mountain	1	0.0363	0.2787	0.0158	4921.0	4921.0	2.728E-12
	2	0.3298	0.3192	0.0154	5779.9	5779.9	1.455E-11
	3	30.355	0.3575	0.0234	6098.0	6098.0	9.095E-13
	4	2007.6	0.3806	0.0238	6227.2	6226.8	3.671E-01

algorithm execution time is independent of the number of thresholds. In conclusion, if there are only two threshold values to be found, it is beneficial to use the exhaustive method. To find three thresholds with the exhaustive method, it takes about 30 seconds compared to about 0.5 seconds of the execution time of SODE simulation. Note that the SODE algorithm uses 100 agents over 100 iterations (10 000 fitness function evaluations). Both parameters – the scaling factor and the probability of crossover – were set to $F = 0.2$ and $P_C = 0.2$.

9.5.2 Thresholding with Variable Number of Thresholds

This subsection shows the results of a study with a multi-objective thresholding problem with a variable number of dimensions. The study is performed on the same set of testing images as in the previous subsection. VND-GDE3 algorithm is exploited on each testing image with 100 agents over 500 iterations. This time, each simulation is performed only once. The algorithm is set as follows: the scaling factor $F = 0.2$, the probability of crossover $P_C = 0.2$, and the probability of dimension transition $P_{\mathrm{DT}} = 0.35$.

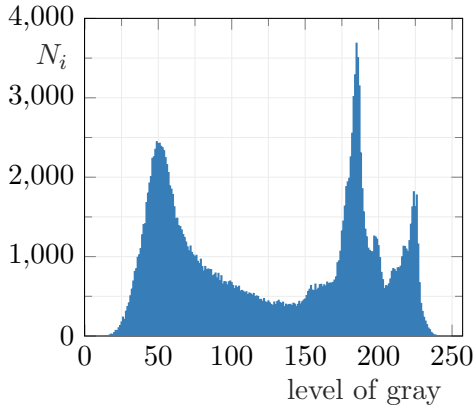
As was mentioned before, the problem is designed so that an algorithm may propose a set of solutions with a different number of thresholds as long as the value of Otsu's inter-class variance increases with the increasing number of thresholds. The maximal number of thresholds was set to five.

Table 9.8 shows the thresholds proposed by the VND-GDE3 algorithm. The thresholds proposed by the VND-GDE3 algorithm are very similar to those proposed by exhaustive Otsu's method (see Table 9.6). However, these results are carried out by just a single optimization run. Each run of the VND-GDE3 algorithm takes about 5 s. Apart from the proposed thresholds, Table 9.8 also shows the number of pixels that are assigned to individual layers of the image after the thresholding (N_t).

Figures 9.10–9.17 show testing images thresholded according to thresholds shown in Table 9.8. The first subfigure in each figure shows a histogram for the current testing image. Note that N_i denotes the number of pixels of the image belonging to the corresponding level of gray. As can be seen, there are some images where just a single threshold is sufficient to differentiate between the foreground and background of the image (Cameraman, Finger, Mountain). However, there are also images where the use of a single threshold does not divide the image into a meaningful picture (Female, San Diego, Sailboat). Fortunately, the user can select the appropriate number of thresholds for a given image post-hoc, thanks to the use of a multi-objective optimization algorithm with a variable number of dimensions. This can be useful in many applications. The following subsection presents the license-plate recognition where the multi-objective VND thresholding is exploited.

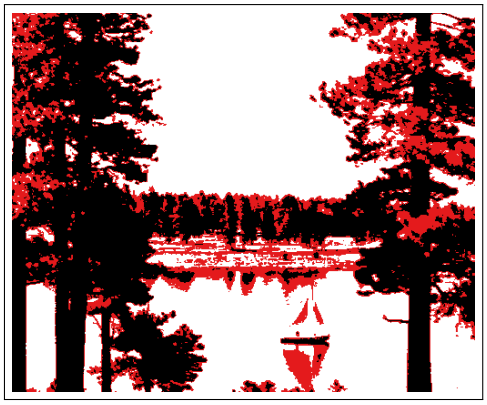
Table 9.8: Results of multi-objective VND thresholding. The table shows threshold values and the corresponding number of pixels assigned to each of the greyscale layers according to the thresholds N_t.

Image	k	Thresholds	Number of pixels per layer - N_t
Cameraman	1	89	[17506, 48030]
	2	[70, 144]	[16452, 16266, 32818]
	3	[58, 118, 155]	[15567, 7412, 15386, 27171]
	4	[44, 96, 139, 169]	[14644, 3492, 12766, 20568, 14066]
	5	[34, 86, 124, 149, 172]	[13723, 3582, 7669, 10055, 18872, 11635]
Sailboat	1	126	[106393, 105860]
	2	[86, 155]	[82542, 36770, 92941]
	3	[80, 141, 195]	[77459, 35114, 63554, 36126]
	4	[70, 115, 161, 201]	[67406, 33874, 21772, 60021, 29180]
	5	[58, 86, 127, 167, 200]	[50183, 32359, 24255, 20188, 55094, 30174]
Female	1	74	[40855, 16713]
	2	[58, 110]	[36487, 13642, 7439]
	3	[34, 68, 114]	[16971, 22071, 11796, 6730]
	4	[33, 62, 95, 131]	[16320, 21107, 9382, 6734, 4025]
	5	[33, 58, 89, 118, 157]	[16320, 20167, 8840, 6147, 4997, 1097]
Airplane	1	153	[60487, 200123]
	2	[112, 173]	[35052, 36475, 189083]
	3	[92, 145, 191]	[18442, 38174, 35862, 168132]
	4	[86, 130, 173, 203]	[14750, 34127, 22650, 69893, 119190]
	5	[71, 107, 145, 181, 204]	[7835, 23045, 25736, 21057, 69977, 112960]
Boat	1	103	[57228, 204916]
	2	[93, 155]	[51355, 134114, 76675]
	3	[73, 126, 167]	[41740, 37838, 143281, 39285]
	4	[63, 113, 147, 180]	[36582, 28944, 79803, 96971, 19844]
	5	[51, 91, 127, 152, 181]	[29002, 21149, 30823, 90125, 72134, 18911]
Finger	1	143	[48966, 109438]
	2	[115, 184]	[35433, 49734, 73237]
	3	[99, 155, 200]	[29301, 27648, 50081, 51374]
	4	[86, 133, 175, 209]	[24203, 18977, 31206, 44732, 39286]
	5	[78, 120, 158, 187, 214]	[20625, 16859, 21566, 30215, 36146, 32993]
San Diego	1	140	[139580, 72485]
	2	[107, 165]	[87218, 76386, 48461]
	3	[91, 129, 177]	[56018, 67680, 48321, 40046]
	4	[78, 110, 144, 184]	[29463, 63135, 51777, 31960, 35730]
	5	[65, 96, 124, 156, 190]	[12210, 53887, 49993, 40233, 23821, 31921]
Mountain	1	137	[151273, 153691]
	2	[74, 169]	[75299, 103428, 126237]
	3	[55, 127, 199]	[55611, 81310, 70544, 97499]
	4	[47, 103, 152, 206]	[50755, 59822, 49919, 56108, 88360]
	5	[28, 79, 123, 168, 217]	[40968, 38788, 57165, 41806, 46598, 79639]

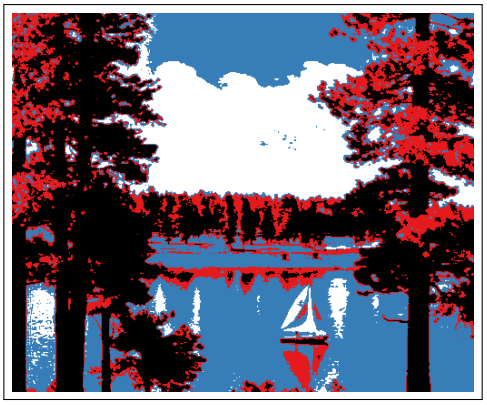


(a) Histogram of sailboat testing image.

(b) Sailboat: one threshold.



(c) Sailboat: two thresholds.

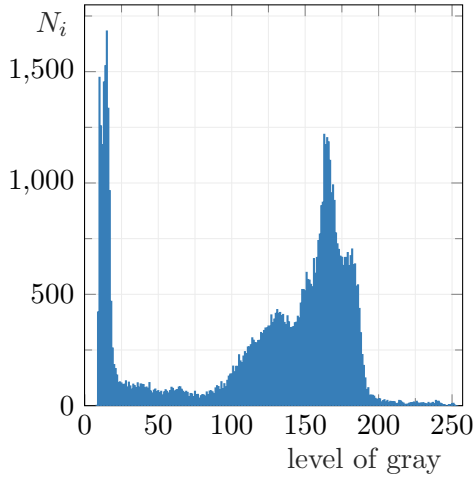


(d) Sailboat: three thresholds.

(e) Sailboat: four thresholds.

(f) Sailboat: five thresholds.

Figure 9.10: Thresholded sailboat testing image.



(a) Histogram of cameraman testing image.

(b) Cameraman: one threshold.

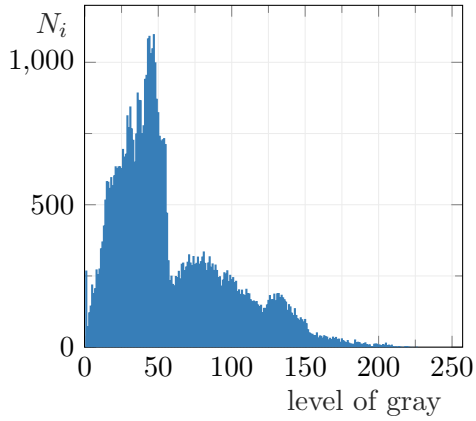
(c) Cameraman: two thresholds.

(d) Cameraman: three thresholds.

(e) Cameraman: four thresholds.

(f) Cameraman: five thresholds.

Figure 9.11: Thresholded cameraman testing image.



(a) Histogram of female testing image.

(b) Female: one threshold.

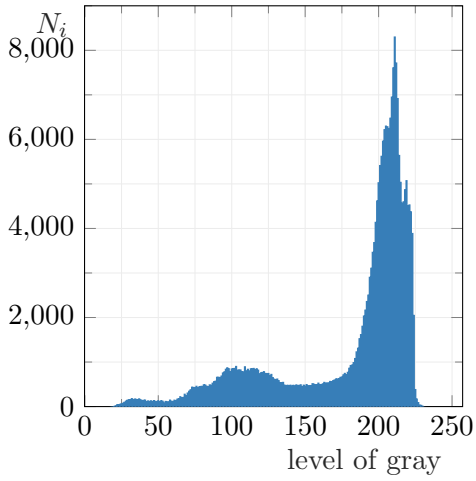
(c) Female: two thresholds.

(d) Female: three thresholds.

(e) Female: four thresholds.

(f) Female: five thresholds.

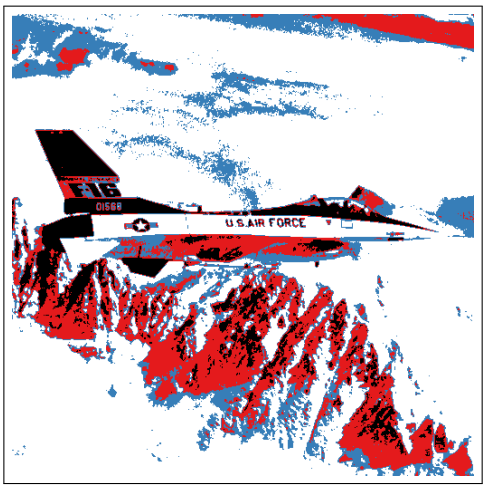
Figure 9.12: Thresholded female testing image.



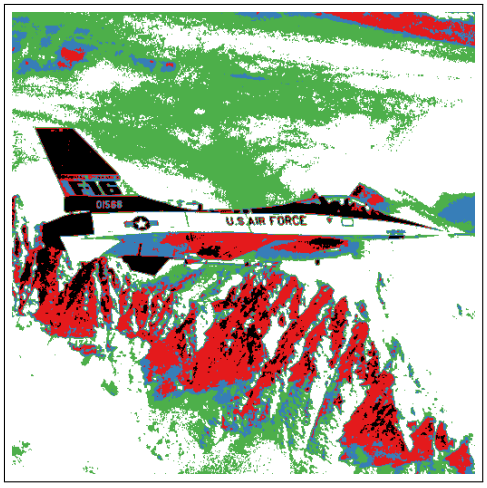
(a) Histogram of airplane testing image.

(b) Airplane: one threshold.

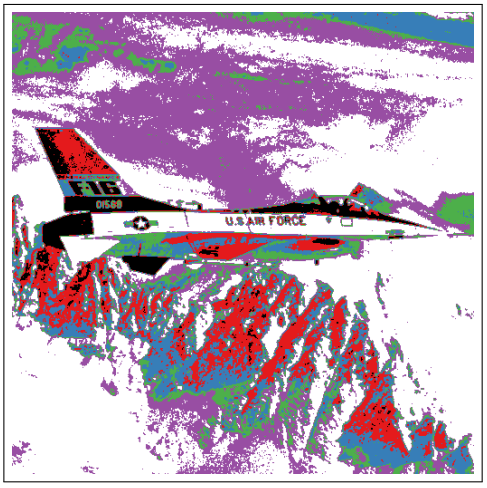
(c) Airplane: two thresholds.



(d) Airplane: three thresholds.

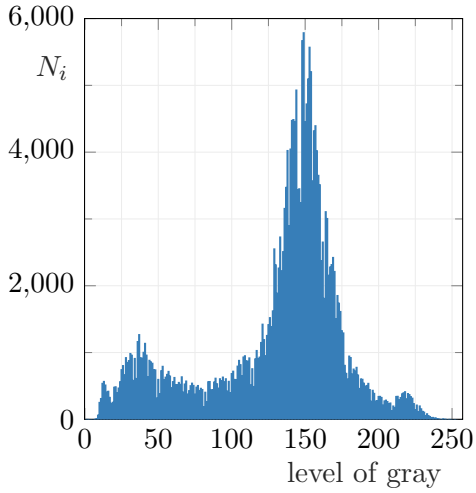


(e) Airplane: four thresholds.



(f) Airplane: five thresholds.

Figure 9.13: Thresholded airplane testing image.



(a) Histogram of boat testing image.

(b) Boat: one threshold.

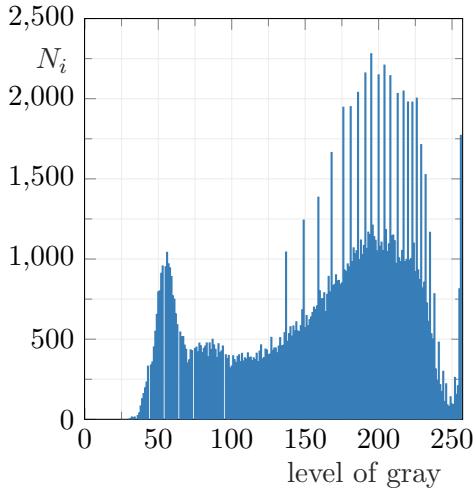
(c) Boat: two thresholds.

(d) Boat: three thresholds.

(e) Boat: four thresholds.

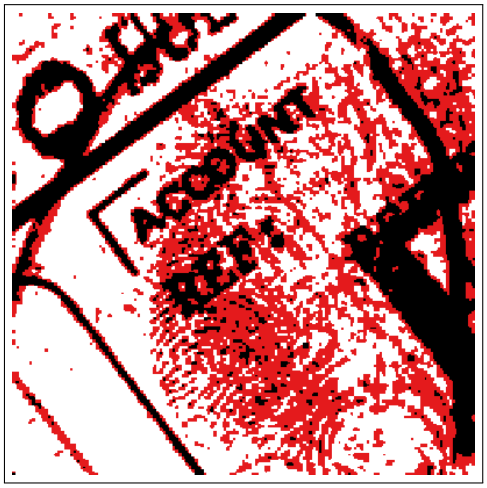
(f) Boat: five thresholds.

Figure 9.14: Thresholded boat testing image.

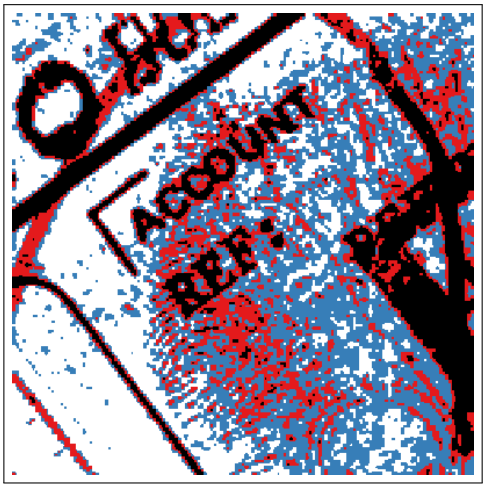


(a) Histogram of finger testing image.

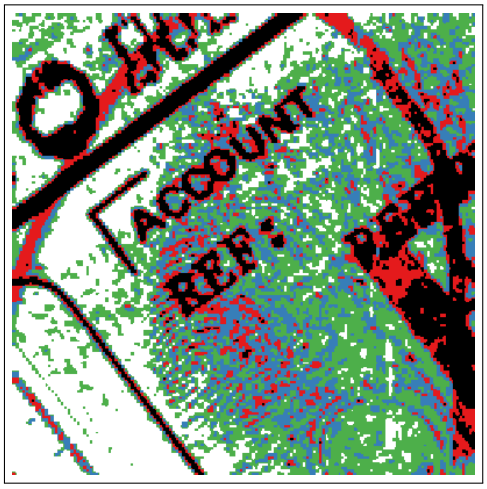
(b) Finger: one threshold.



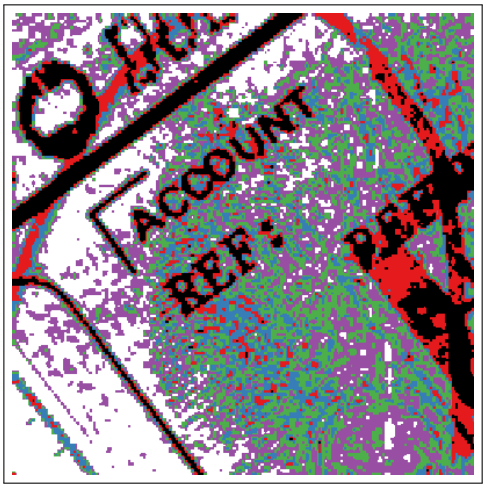
(c) Finger: two thresholds.



(d) Finger: three thresholds.

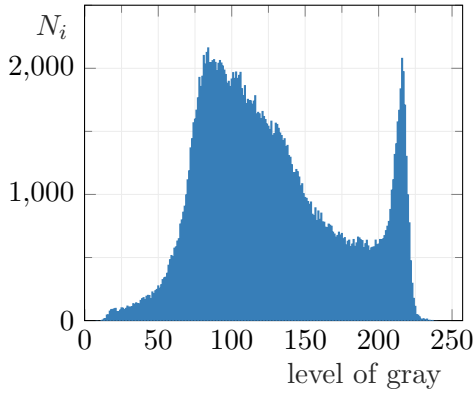


(e) Finger: four thresholds.



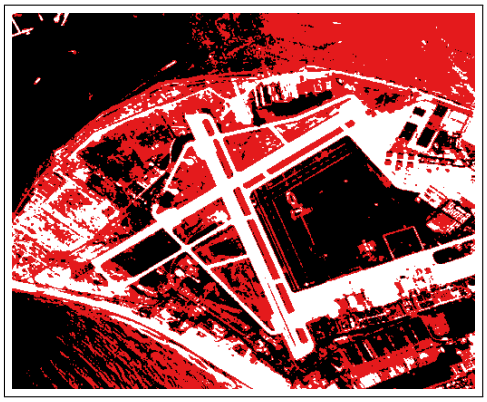
(f) Finger: five thresholds.

Figure 9.15: Thresholded finger testing image.

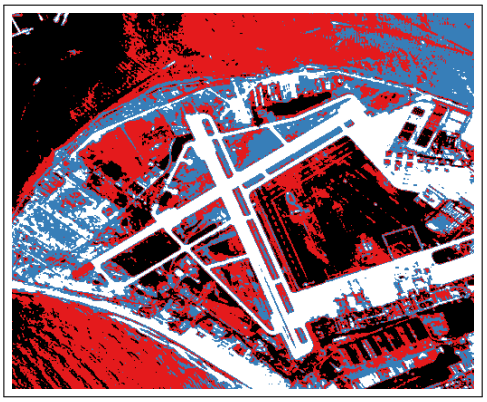


(a) Histogram of sandiego testing image.

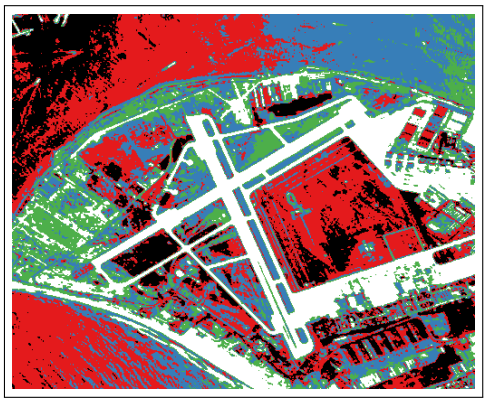
(b) Sandiego: one threshold.



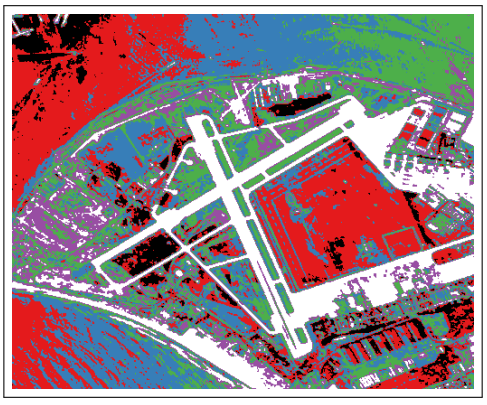
(c) Sandiego: two thresholds.



(d) Sandiego: three thresholds.

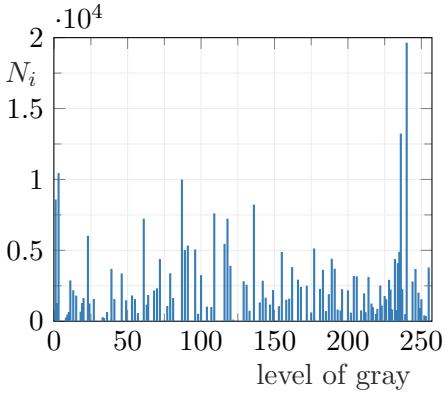


(e) Sandiego: four thresholds.

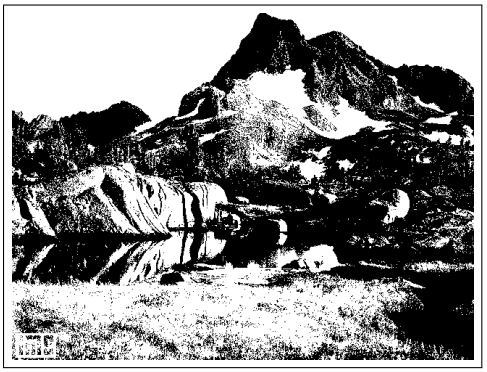


(f) Sandiego: five thresholds.

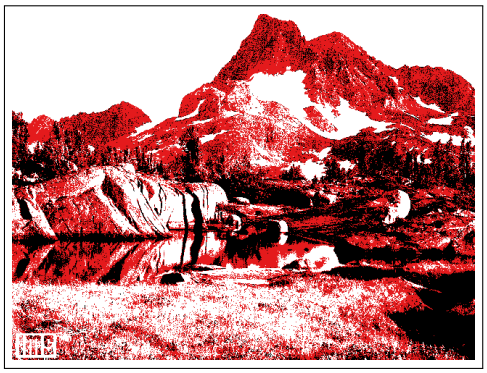
Figure 9.16: Thresholded sandiego testing image.



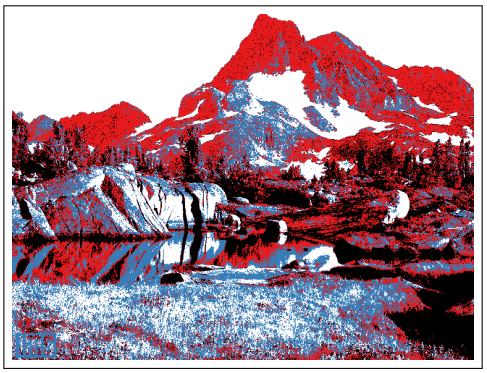
(a) Histogram of mountain testing image.



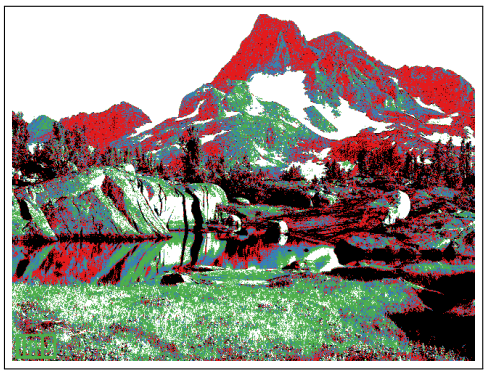
(b) Mountain: one threshold.



(c) Mountain: two thresholds.



(d) Mountain: three thresholds.



(e) Mountain: four thresholds.

(f) Mountain: five thresholds.

Figure 9.17: Thresholded mountain testing image.

9.5.3 License Plate Recognition by Using Thresholding

License plate recognition (LPR) or automatic number plate recognition (ANPR) is widely used in many real-life situations including parking lot attendance, traffic laws enforcement, etc. An image with a license plate is pre-processed before the optical character recognition of the license plate by image thresholding [107], [108]. However, the thresholding in the license plate recognition is not a trivial task [109]. The following figures in this subsection show that lighting conditions have a major impact on the performance of the thresholding technique. Note that image analysis in LPR consists of three parts: localization of the license plate in the image, thresholding of the license plate region, and optical character recognition of the characters. However, the localization and OCR steps are out of the scope of this thesis. Therefore, the image thresholding is performed on the whole testing image instead of the section with the license-plate. Such an approach makes the thresholding more challenging. Nonetheless, it better demonstrates the advantage of the variable number of dimensions approach.

The thresholding technique used for license-plate recognition is the same one from the previous subsection. It uses a variable number of dimensions representation with two objectives to find trade-off solutions with multiple threshold values. It will be shown that the number of thresholds needed for the proper image segmentation cannot be determined a priori.

Figures 9.18 – 9.21 show testing images for license-plate recognition. There are four subfigures for each figure where the top-left shows the original greyscale image and the remaining three subfigures show results of the thresholding method. Note that only a limited number of subfigures is shown in this section in order to keep a reasonable extent of this subsection. However, Appendix A in 10 contains more figures with results of thresholded LPR testing images.

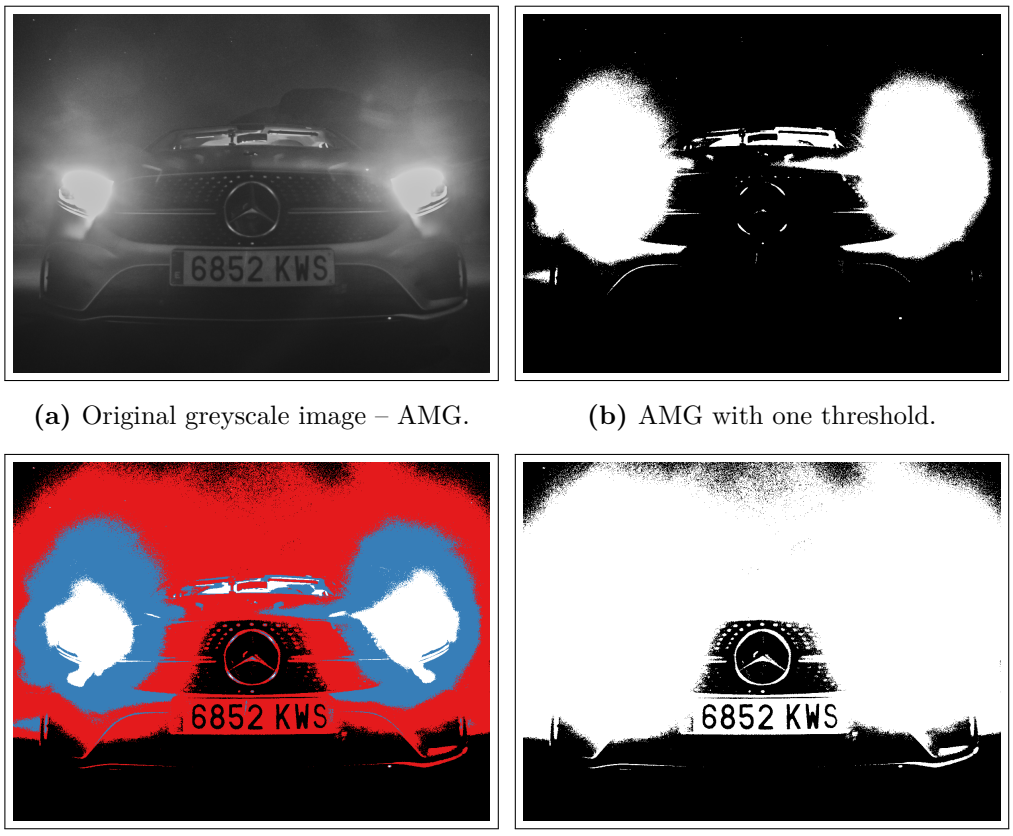
Figure 9.18 shows the AMG testing image. In this picture, the light from the headlamps is much brighter than the background of the license plate. Therefore, using just one threshold creates two layers, but the license plate is apparent in neither of them. Contrarily, if three thresholds are used, the license plate is clearly visible in the first layer (see Figure 9.18d). The subsequent OCR routine would most likely yield the desired "6852 KWS".

Figure 9.19 shows the BMW rear testing image. This picture shows very difficult light conditions for license-plate recognition. It is caused by the sun glare in the area of the license plate. Figure 9.19b shows the image with three thresholds, while Figures 9.19c and 9.19d. show the third and fourth layers, respectively. It can be seen that the "RAW" part of the license plate should be obtained correctly. Contrarily, the "PH 78" part of the license-plate is hardly legible because the characters are made only by contours.

Figure 9.20 shows the BMW front testing image. If two thresholds are used (see Figure 9.20b), the characters of the license plate can be seen, but they are rather noisy and the last "9" is melting into the edge of the license plate. Contrarily, the characters in the license plate are visible clearly in the case of three thresholds (Figures 9.20c and

9.20d).

Figure 9.21 shows the Taxi testing image. Light conditions in this image are very difficult. Moreover, the illumination from the license plate lamps makes the thresholding task even harder. It can be seen in Figure 9.21b that most of the characters in the license plate blends with the background. Figures 9.21c and 9.21d shows the thresholding with two thresholds. In this case all the characters in the license plate fell into the first layer. Contrarily, the whole background of the license plate is in the second or third layer. Therefore, the character recognition task is simple if two thresholds are used.



(a) Original greyscale image – AMG.

(b) AMG with one threshold.

(c) AMG with three thresholds.

(d) Three thresholds – first layer.

Figure 9.18: LPR testing image – AMG [110].

(a) Original greyscale image – BMW rear.

(b) BMW rear with three thresholds.

(c) Three thresholds – third layer.

(d) Three thresholds – fourth layer.

Figure 9.19: LPR testing image – BMW rear [111].

(a) Original greyscale image – BMW front.

(b) Two thresholds – first layer.

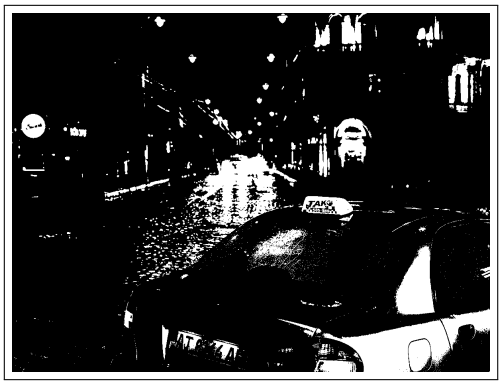
Figure 9.20: LPR testing image – BMW front [112].

(c) BMW front with three thresholds.

(d) Three thresholds – first layer.

Figure 9.20: LPR testing image – BMW front [112].

(a) Original greyscale image – Taxi.



(b) Taxi with one threshold.

(c) Taxi with two thresholds.

(d) Two thresholds – first layer.

Figure 9.21: LPR testing image – Taxi [113].

9.6 Clustering Problem

Clustering is a widely used unsupervised pattern classification technique. It separates the N_{DI} data-items (observations) into K clusters based on some similarity/dissimilarity metric [114]. The objective of the separation is to have data-items within one cluster to be similar to each other, while the data-items within different clusters to be dissimilar. Clustering techniques are used in a wide variety of applications: machine learning [115], image segmentation [116], data mining [43], psychology [117], economics [118], etc.

There are many clustering techniques to be found in the literature [119], such as the K-means clustering method [120], mean-shift clustering method [121], Density-based Spatial Clustering of Applications with Noise (DBSCAN) [122], etc. However, in this section, the evolutionary clustering approach will be compared to the K-means clustering method and the DBSCAN method.

The K-means clustering method is probably the most popular clustering algorithm for its simplicity. This algorithm minimizes the within-cluster sum of squares. In the beginning, the algorithm randomly selects K centroids. Afterward, each data-item is assigned to the cluster with the nearest mean (with the minimal Euclidean distance). The centroids are recalculated for data-items assigned to each cluster according to:

$$\mathbf{c}_q^{(g)} = \frac{1}{\left|C_q^{(g-1)}\right|} \sum_{\mathbf{x}_j \in C_q^{(g-1)}} \mathbf{x}_j, \tag{9.35}$$

where $\mathbf{c}_q^{(g)}$ is the mean of the q-th cluster (centroid) for the current iteration, $C_q^{(g-1)}$ is the set of data-items assigned to the q-th cluster in the previous iteration, and $\mathbf{x}_j$ denotes the j-th data-item in the cluster. When the cluster means are updated, all the observations are reassigned. These steps are repeated for a given number of iterations. It is evident that the number of clusters K must be given a priori. Nonetheless, the algorithm is widely used for its simplicity and small computational complexity.

On the other hand, the DBSCAN clustering method can identify the number of clusters K. However, it is computationally demanding, and it also performs poorly when the clusters are of varying density. In the beginning, the DBSCAN takes an arbitrary data point and searches its vicinity for the presence of neighbors. If there is not a sufficient number of points (user-defined parameter N_P), the data point is considered an outlier. The area of the neighborhood is controlled with the second user-defined parameter ϵ. If there are N_P or more neighbors within ϵ distance of the current data point, the neighbors are marked to belong to the same cluster. This search is recursively repeated over all the neighboring solutions until none of the new neighbors have enough data-items nearby. At this point, all the data points in the current cluster are determined. Afterward, a new unvisited point is retrieved, and the search process is repeated for another cluster. The DBSCAN process is finished when all the data points are marked as visited - in other words, all data points are either assigned to any cluster or marked as outliers.

9.6.1 Evolutionary Clustering

The evolutionary approach used in this section takes over the principle of the K-means clustering method and removes its fundamental disadvantage – K must be given a priori. The evolutionary algorithm proposes cluster means (centroids), and five iterations of the K-means method are performed for every agent. Using the GDE3 algorithm with a variable number of dimensions allows the method to test a various number of clusters as well as diverse initial centroids. When those five K-means iterations are executed, the updated centroid positions are injected back to the VND-GDE3 agent positions, and the fitness values are calculated for the updated centroids. Note that this is identical to the approach described in Section 9.2. In this case, the VND-GDE3 algorithm can be seen as a global optimizer, while the K-means method is a local optimizer with rapid convergence. Also note, that there is no need to perform many K-means iterations inside the fitness function evaluation because the K-means method starts with the centroids stored in the agent from previous iterations. Therefore, the K-means algorithm may perform up to $G^{(\mathrm{K-means})} = 5 \times G^{(\mathrm{VND-GDE3})}$ iterations, where G denotes the number of iterations.

The fitness function evaluation consists of five K-means iterations and then the assessment of the quality of clustering. The obvious approach is to calculate the number of data-items that were incorrectly assigned to the wrong cluster in comparison with the ground-truth (the ideal clustering solution) of the dataset. However, the ground-truth in a real-world clustering problem is unknown. Therefore, the quality of the solution is assessed by the Calinski-Harabasz index [123]. There are other measures to be found in the literature [124], but the Calinski-Harabasz index suits best to our needs. The index is fast to compute and favors clusters that are dense and well separated.

Calinski-Harabasz index, also known as the Variance Ratio Criterion, is the ratio of the sum of between-clusters dispersion and inter-cluster dispersion. The higher the score is, the better the clustering is. The index is calculated according to:

$$\mathrm{VRC} = \frac{\mathrm{SS}_B}{\mathrm{SS}_W} \frac{N_{\mathrm{DI}} - K}{K - 1}, \tag{9.36}$$

where SS_B is between cluster variance, SS_W is within-cluster variance, N_{DI} is the number of data-items, and K is the number of clusters. The between-cluster variance is obtained by:

$$\mathrm{SS}_B = \sum_{q=1}^{K} n_q \left(\mathbf{c}_q - \mathbf{c}_E\right) \left(\mathbf{c}_q - \mathbf{c}_E\right)^T, \tag{9.37}$$

where n_q is the number of data-items in the q-th cluster, $\mathbf{c}_q$ is the center of the cluster q, and $\mathbf{c}_E$ is the center of all data points. The within-cluster variance is calculated according to:

$$\mathrm{SS}_W = \sum_{q=1}^{K} \sum_{\mathbf{x}_j \in C_q} \left(\mathbf{x}_j - \mathbf{c}_q\right) \left(\mathbf{x}_j - \mathbf{c}_q\right)^T, \tag{9.38}$$

where $\mathbf{x}_j$ denotes the j-th data-item in the cluster and C_q is the set of data-items in the cluster q.

Finally, the fitness functions can be stated as:

$$f_1 = -\text{VRC}, \tag{9.39}$$

$$f_2 = K. \tag{9.40}$$

Note that the Calinski-Harabasz index is taken negatively because the VND-GDE3 expects a minimization problem. The second fitness function is the number of clusters.

9.6.2 Verification of the Method

This subsection deals with eight benchmark clustering datasets. Each of the datasets is solved by the standard K-means method, DBSCAN, and the evolutionary approach described in the previous subsection. Each simulation is repeated ten times because of the stochastic nature of the processes. The result of each simulation is compared to the ground-truth of the dataset. The number of errors is given in Tables 9.10, 9.11, and 9.12 for the VND-GDE3, K-means, and DBSCAN methods, respectively. Finally, the results of the clustering is visualized in Figures 9.22 – 9.28. Note that the Iris dataset is not visualized because the data-items have four dimensions.

The VND-GDE3 method used 100 agents over 50 iterations. Controlling parameters of VND-GDE3 algorithms were set as follows: the scaling factor $F = 0.2$, the probability of crossover $P_C = 0.2$, and the probability of dimension transition $P_{\text{DT}} = 0.35$. The minimal number of clusters for all datasets was two, and the maximal number of clusters was 35. The K-means method was performed for 250 iterations for each problem. The number of clusters for each dataset was pre-set to the ideal value, as shown in Table 9.9. The DBSCAN method uses two user-defined parameters – N_P and ϵ. They vary for each dataset, and the value shown in Table 9.9 was obtained empirically. Note that Table 9.9 also contains references to where the individual datasets were obtained.

In Tables 9.10, 9.11, and 9.12, it can be seen that the VND-GDE3 produced very consistent results compared to the K-means method. In the case of Flame and Aggregation

Table 9.9: Settings of clustering methods

	VND-GDE3		**K-means**	**DBSCAN**			
Dataset	$K_{\min}$	$K_{\max}$	K	N_P	ϵ	N_{DI}	**Reference**
Flame	2	35	2	4	1	240	[125]
Aggregation	2	35	7	3	1.2	788	[126]
Unbalanced	2	35	8	4	10000	6500	[127]
S1	2	35	15	4	20000	5000	[128]
S3	2	35	15	4	15000	5000	[128]
R15	2	35	15	4	0.4	600	[129]
D31	2	35	31	7	0.6	3100	[129]
Iris	2	35	3	5	0.25	150	[130]

Table 9.10: Number of errors in clusters proposed by VND-GDE3

Run	Flame	Aggregation	Unbalanced	S1	S3	R15	D31	Iris
1	39	107	0	4	91	2	119	16
2	39	108	0	4	101	2	119	16
3	39	107	0	4	108	2	170	16
4	39	107	0	4	118	2	156	16
5	39	107	0	3	84	2	185	16
6	39	107	0	4	109	2	104	16
7	39	107	0	3	94	2	180	16
8	39	107	0	3	96	2	135	16
9	39	107	0	4	81	2	105	16
10	39	107	0	4	99	2	240	16

Table 9.11: Number of errors in clusters proposed by K-means method

Run	Flame	Aggregation	Unbalanced	S1	S3	R15	D31	Iris
1	38	102	2042	437	1542	193	752	77
2	38	165	2042	506	984	116	867	77
3	35	95	4073	2156	103	221	678	17
4	39	107	0	416	676	140	1050	17
5	39	169	1102	1503	1808	224	947	75
6	41	106	4053	487	1229	273	1194	74
7	41	107	2045	928	754	113	1089	17
8	38	146	2040	4	689	246	753	17
9	38	169	2041	1230	649	213	969	17
10	38	169	0	1562	599	173	848	75

Table 9.12: Number of errors in clusters proposed by DBSCAN method

Run	Flame	Aggregation	Unbalanced	S1	S3	R15	D31	Iris
$\forall$	9	135	14	190	3128	57	1073	124

datasets, the K-means method was able to find a better solution than VND-GDE3. However, the improvement in best-case scenarios is very small compared to the deterioration in worst-case scenarios. It can be seen in Table 9.12 that the number of errors for any repetition of the DBSCAN method resulted in the same cluster geometry. Although the DBSCAN is stochastic, the only randomness is involved in the selection of the starting (unvisited) solution in a new cluster.

The following Figures 9.22 – 9.28 contain a set of subfigures for each dataset. The first subfigure shows the ground-truth. Below the ground-truth, there are pairs of subfigures next to each other for each clustering method. The visualization shows the best-case scenario. If the worst-case scenario for a corresponding method and dataset is diverse enough, the visualization of the worst-case scenario is shown as well. Each cluster in a figure is distinguished by a different color. However, there are only nine different colors

in the qualitative color scheme used. Therefore, there may be more clusters using the same color in the same figure. Therefore, it is necessary to distinguish the centroids of the clusters as well. They are marked with the cross sign with the same color as the data-items in the corresponding cluster.

The visualization of the Flame dataset is shown in Figure 9.22. It can be seen that VND-GDE3 and K-means methods performed rather poorly. However, this dataset is designed specifically so that clustering methods based on measurement of Euclidean distance from centroids would fail. Contrarily, the DBSCAN method excels in such problems. However, its result is strongly dependent on the settings of the N_p and ϵ controlling parameters. These parameters vary according to the clustering dataset under test, and the setting itself is not a trivial task.

The Aggregation dataset is a rather deceptive one. There are clusters with different sizes and a different number of data-items in them. Figures 9.23b and 9.23c show the best solution from the VND-GDE3 method. The VND-GDE3 method was unable to divide the three bottom left clusters with different sizes correctly. However, the K-means algorithm was also struggling, as can be seen in Figures 9.23d – 9.23g. On the other hand, the DBSCAN method managed well with the bottom left clusters. However, it got caught in another trap. There is a connection between the two clusters on the right. This is specifically designed to outwit the density-based algorithms such as the DBSCAN is.

Figure 9.24 shows the Unbalanced benchmark dataset. It can be seen from Tables 9.10, 9.11, and 9.12 that the VND-GDE3 algorithm found the ideal solution in all runs. On the other hand, the K-means algorithm found the ideal solution only twice. In the remaining attempts, the K-means method often failed to separate the three clusters on the left. The problem is that these clusters contain a large number of data-items compared to the clusters on the right. The DBSCAN method found a satisfactory solution, apart from the data-items it marked as outliers.

The S1 benchmark dataset contains 15 clusters that are rather separated. The best solution of VND-GDE3 and K-means methods are almost identical and not far from ground-truth. However, the VND-GDE3 maintained such a solution consistently in all runs. Contrarily, the K-means method found satisfactory solution just once. Figures 9.25f and 9.25g show the worst solution from the K-means method. The number of incorrectly assessed centroids is large. The DBSCAN method performed rather well. However, the number of clusters proposed by the DBSCAN is 17 because some outlying solution were incorrectly considered as a new cluster.

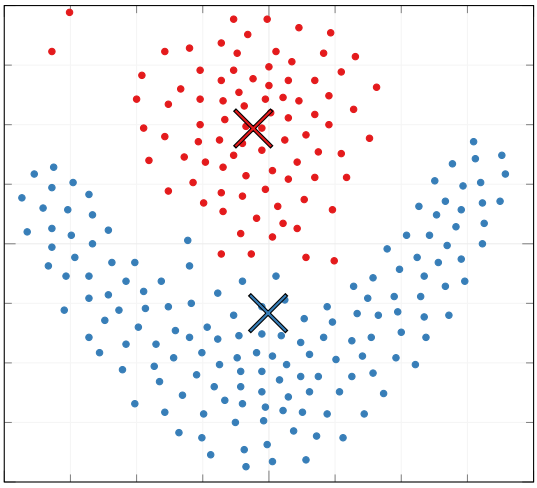
Figure 9.26 visualizes the S3 dataset. This dataset shows clusters that are poorly separated compared to the S1 dataset. As shown in Figures 9.26b – 9.26e, the VND-GDE3 performed rather well, and the difference between the best and worst solution is relatively small. On the other hand, the difference between the best and worst solution of the K-means method is extensive. The DBSCAN method on the S3 dataset completely fails because some of the clusters are not separated at all.

Visualization of the R15 dataset is shown in Figure 9.27. The VND-GDE3 found ten-

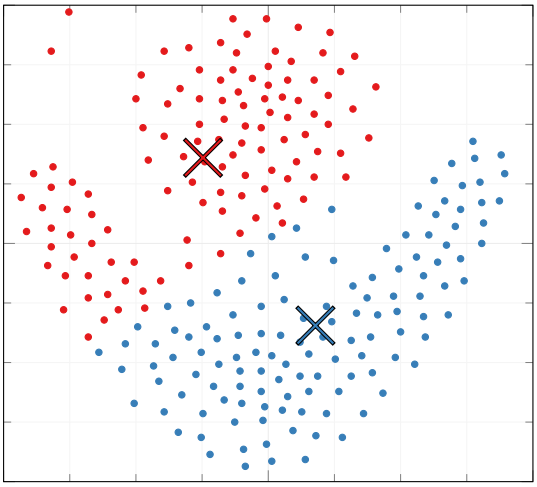
times the solution with only two errors. Contrarily, the K-means algorithm was not able to cluster the data satisfactorily. The DBSCAN method failed to recognize 15 clusters.

Figure 9.28 shows the visualization of the D31 dataset. As the name suggests, the number of clusters in this dataset is 31. That brings a considerable challenge to the clustering algorithm. Nonetheless, the VND-GDE3 found solutions with a minimum of 104 errors and a maximum of 240 errors. The reason that there were 240 incorrectly assigned data-items is that the algorithm proposed 35 clusters. However, in comparison with the K-means and the DBSCAN methods, the VND-GDE3 still has the best performance on the D31 dataset.

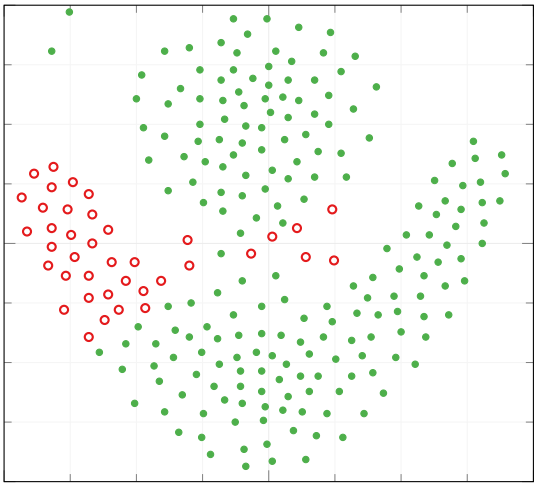
Finally, the Iris dataset is not visualized in figures because its data-items have four dimensions. Nonetheless, this dataset is the most used in publications related to clustering discipline. Tables 9.10, 9.11, and 9.12 show that the VND-GDE3 algorithm is most convenient because it found a solution with only 16 errors in all the repetitions.



(a) Ground-truth of Flame dataset.

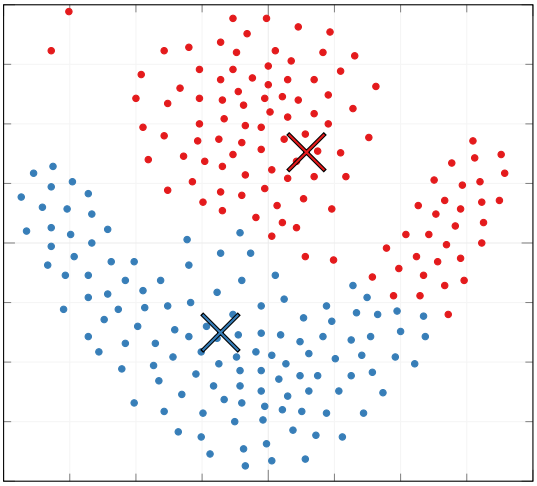


(b) Best clusters proposed by VND-GDE3.

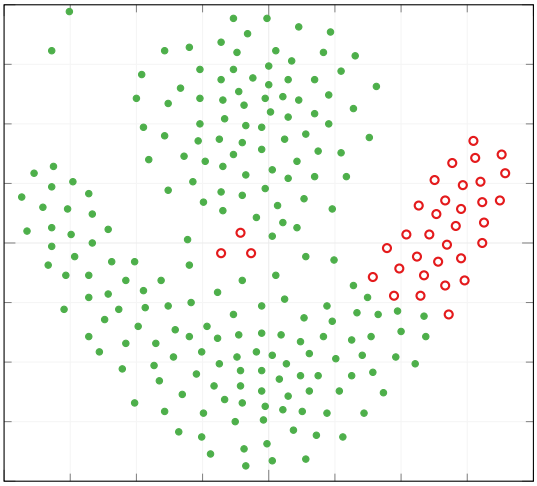


(c) Ground-truth vs. best VND-GDE3.

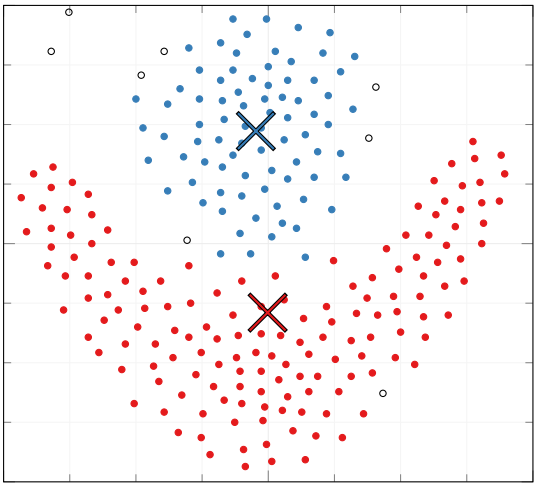
Figure 9.22: Visualization of Flame clustering dataset.



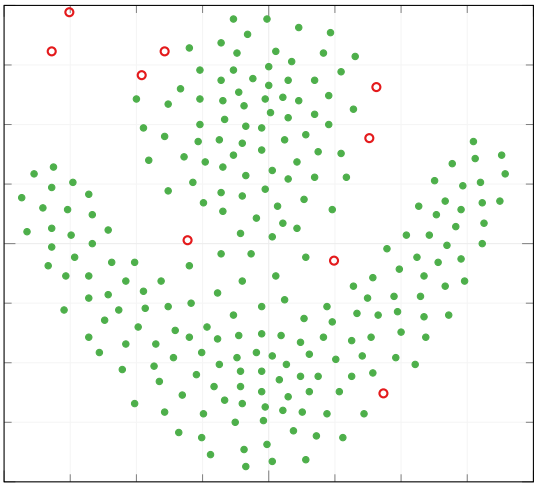
(d) Best clusters proposed by K-means.



(e) Ground-truth vs. best K-means.

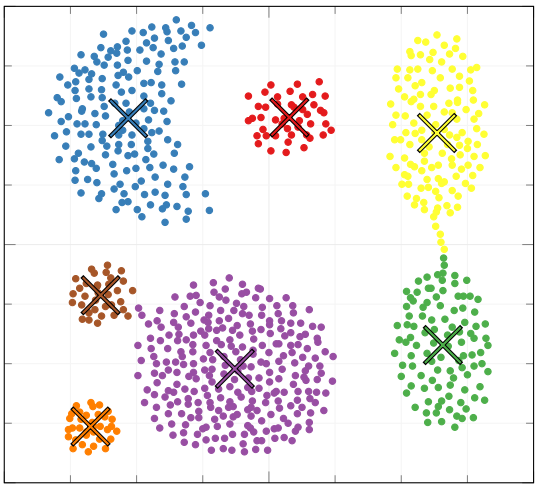


(f) Best clusters proposed by DBSCAN.



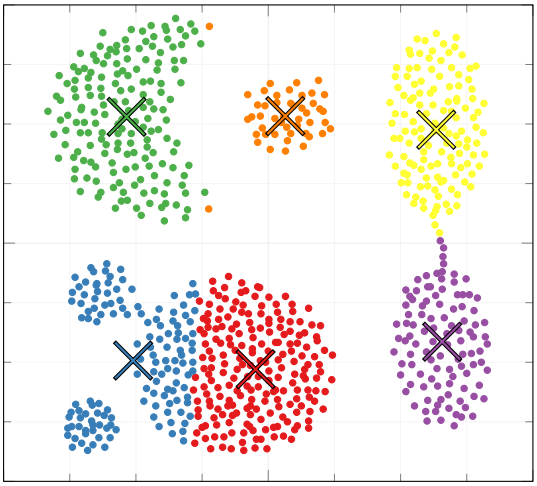
(g) Ground-truth vs. best DBSCAN.

Figure 9.22: Visualization of Flame clustering dataset.

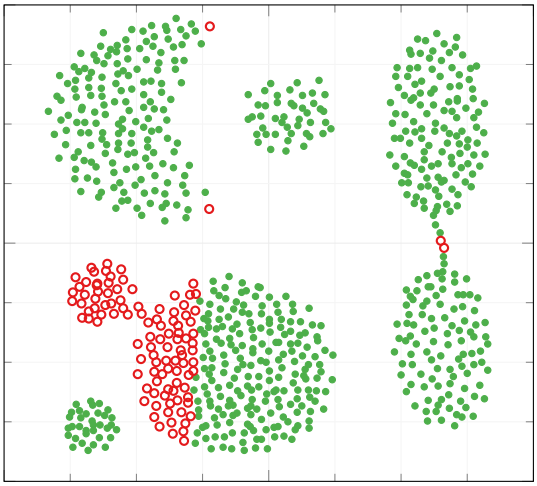


(a) Ground-truth of Aggregation dataset.

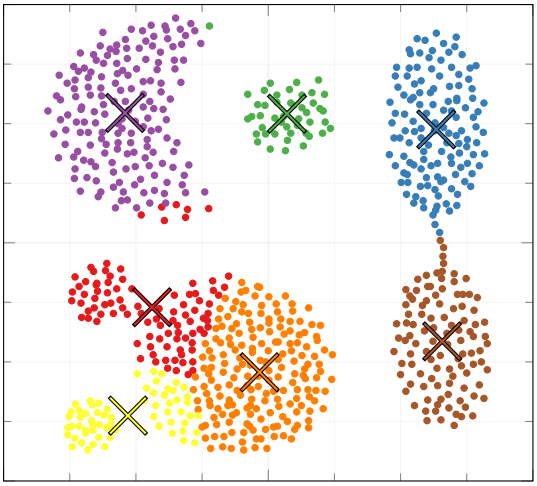
Figure 9.23: Visualization of Aggregation clustering dataset.



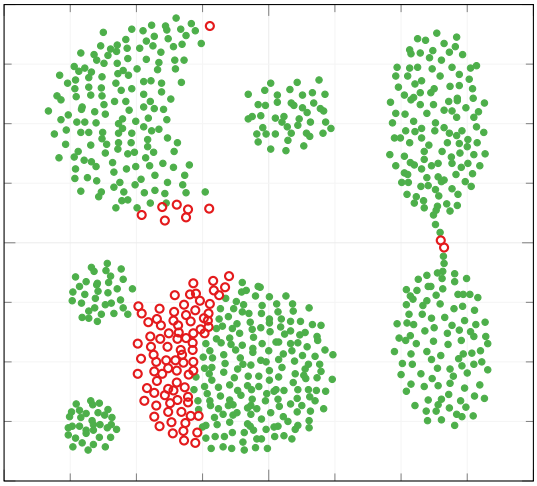
(b) Best clusters proposed by VND-GDE3.



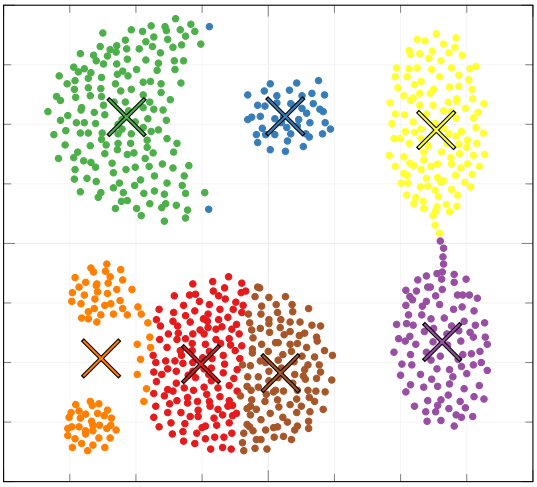
(c) Ground-truth vs. best VND-GDE3.



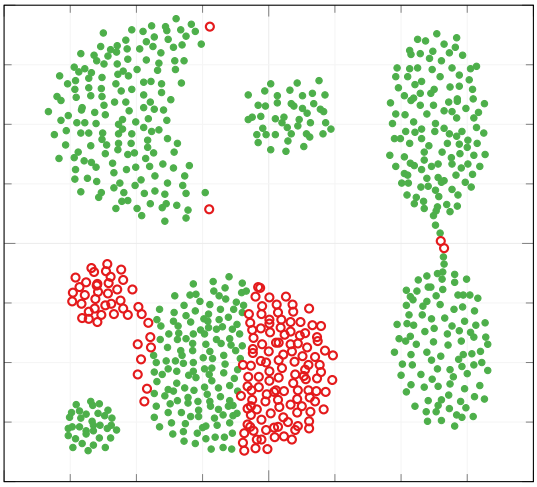
(d) Best clusters proposed by K-means.



(e) Ground-truth vs. best K-means.

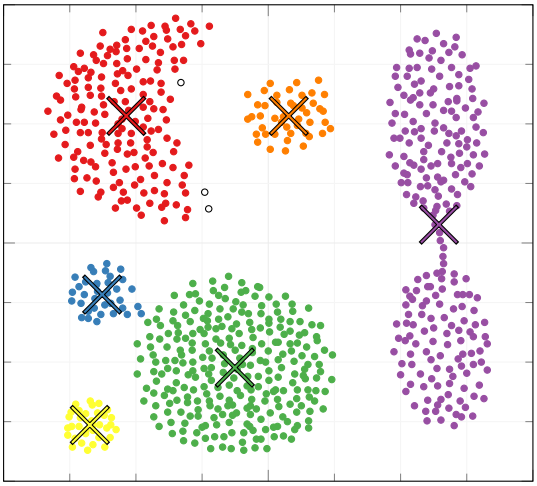


(f) Worst clusters proposed by K-means.

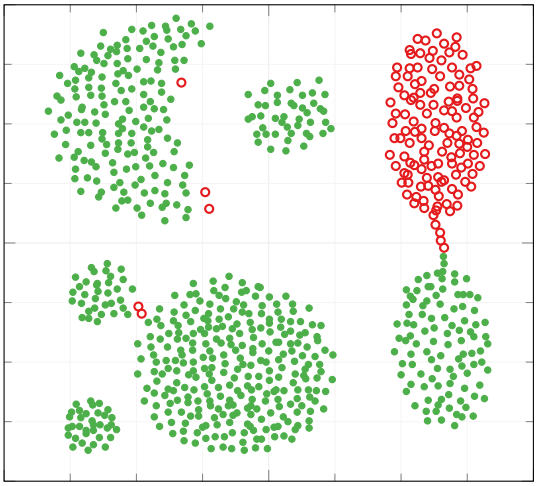


(g) Ground-truth vs. worst K-means.

Figure 9.23: Visualization of Aggregation clustering dataset.

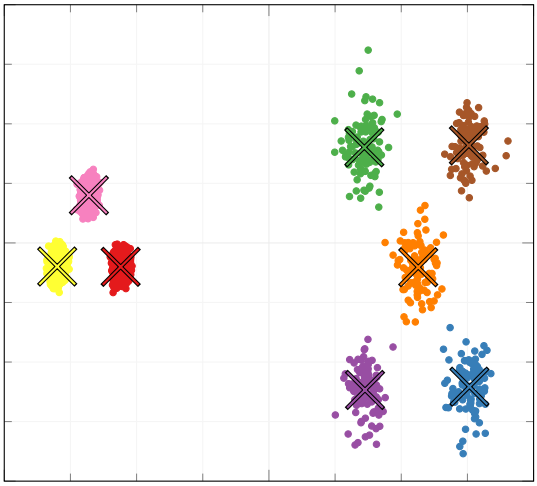


(h) Best clusters proposed by DBSCAN.

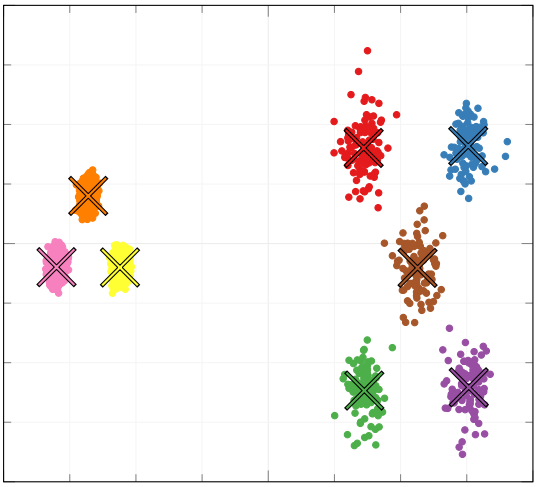


(i) Ground-truth vs. best DBSCAN.

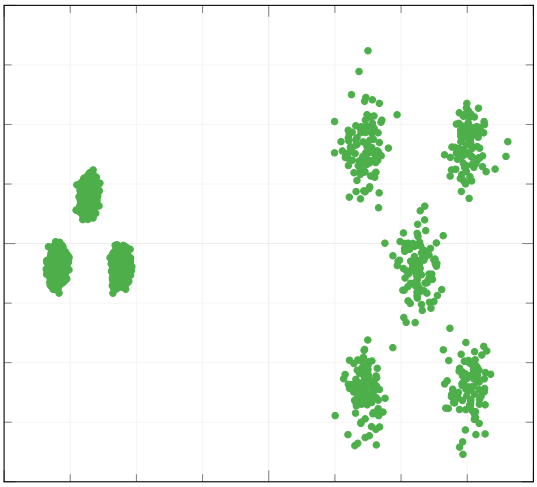
Figure 9.23: Visualization of Aggregation clustering dataset.



(a) Ground-truth of Unbalanced dataset.

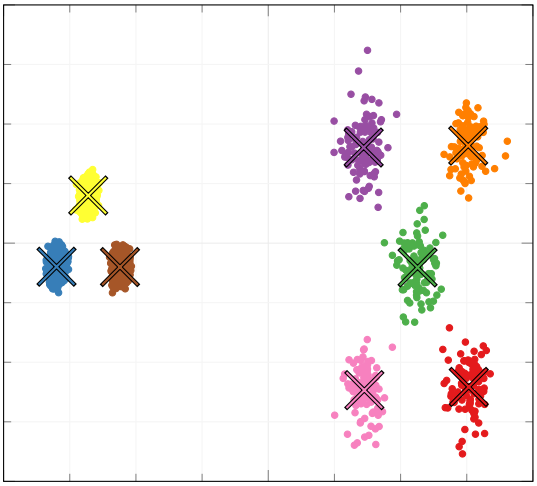


(b) Best clusters proposed by VND-GDE3.

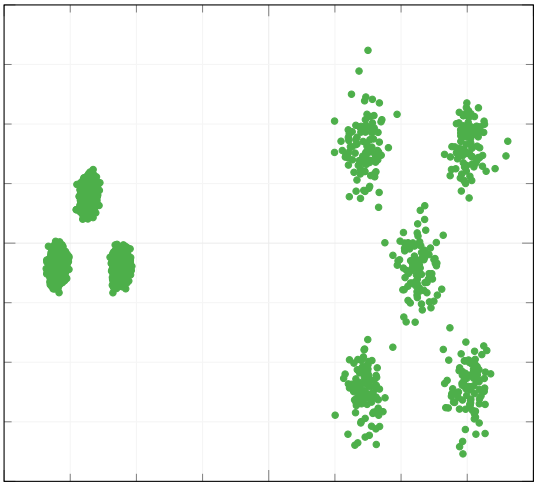


(c) Ground-truth vs. best VND-GDE3.

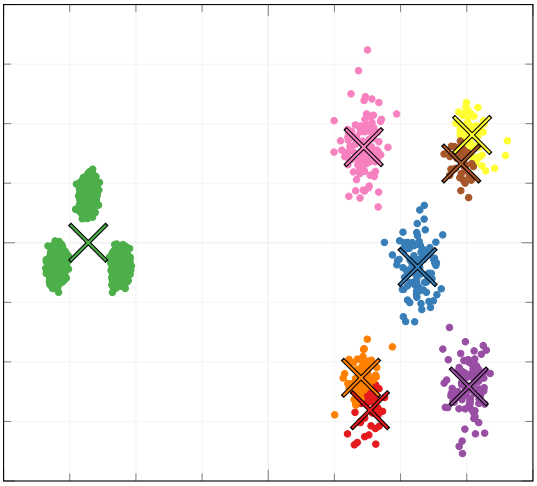
Figure 9.24: Visualization of Unbalanced clustering dataset.



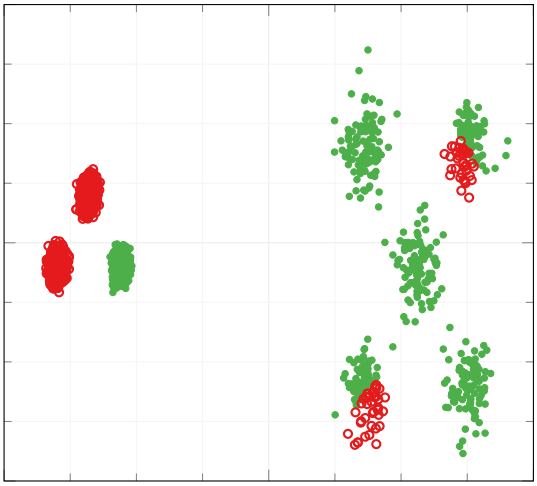
(d) Best clusters proposed by K-means.



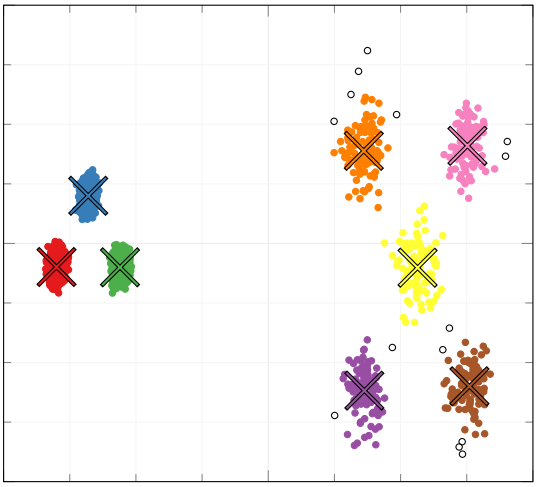
(e) Ground-truth vs. best K-means.



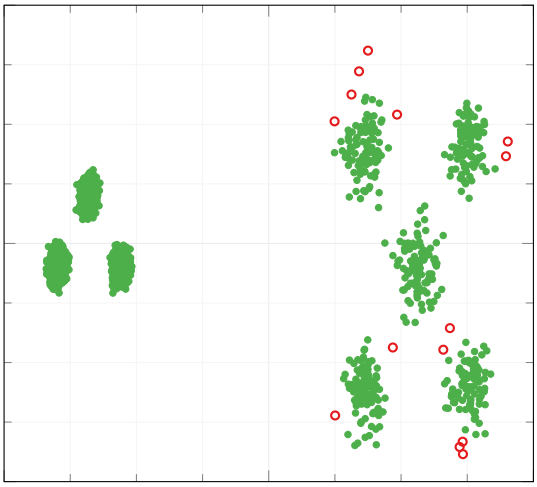
(f) Worst clusters proposed by K-means.



(g) Ground-truth vs. worst K-means.

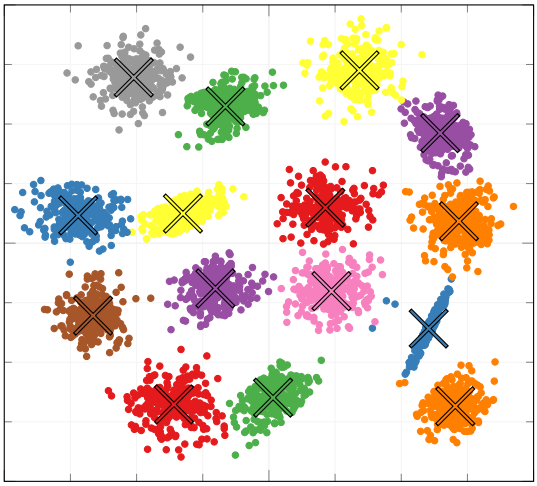


(h) Best clusters proposed by DBSCAN.

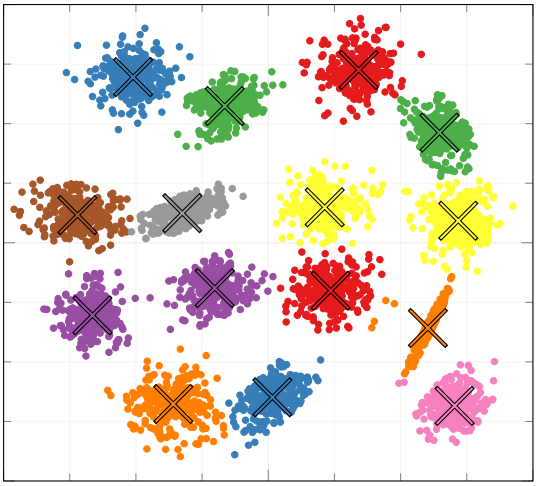


(i) Ground-truth vs. best DBSCAN.

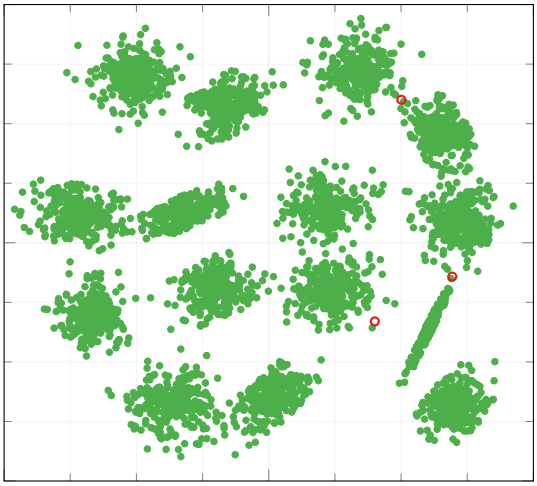
Figure 9.24: Visualization of Unbalanced clustering dataset.



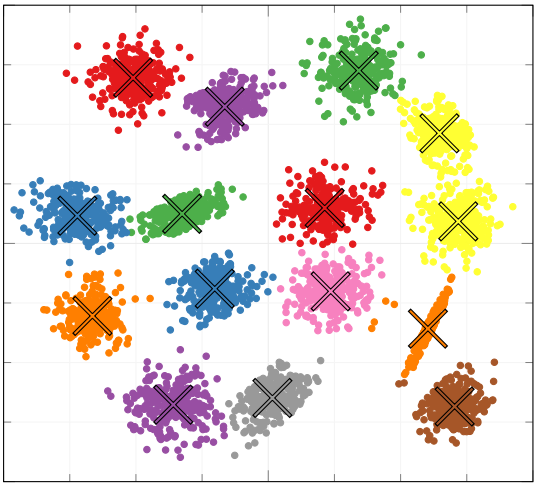
(a) Ground-truth of S1 dataset.



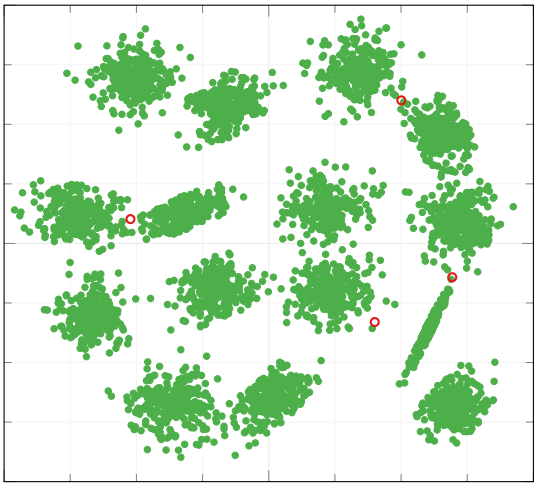
(b) Best clusters proposed by VND-GDE3.



(c) Ground-truth vs. best VND-GDE3.

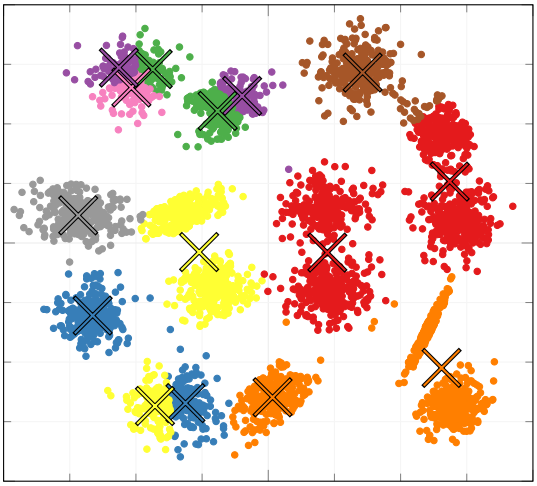


(d) Best clusters proposed by K-means.

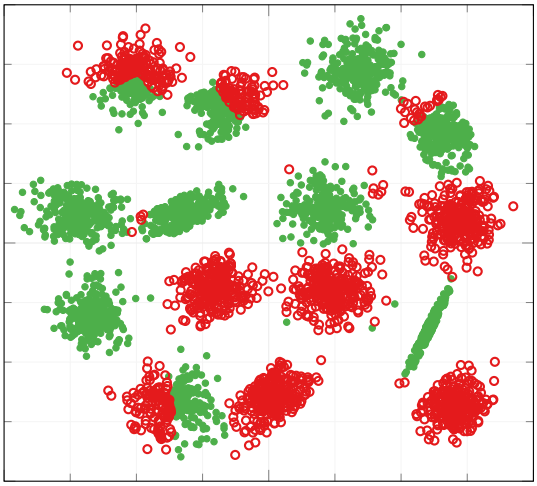


(e) Ground-truth vs. best K-means.

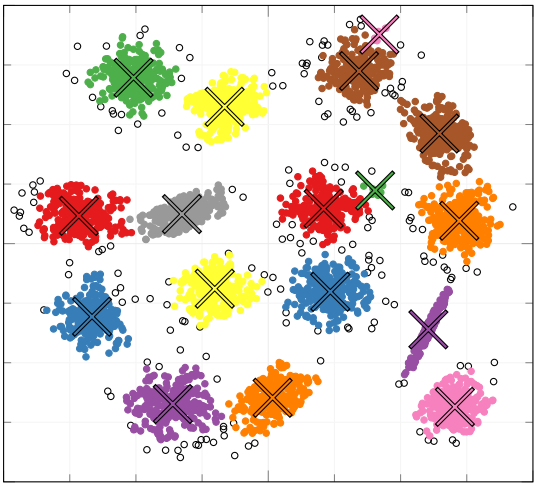
Figure 9.25: Visualization of S1 clustering dataset.



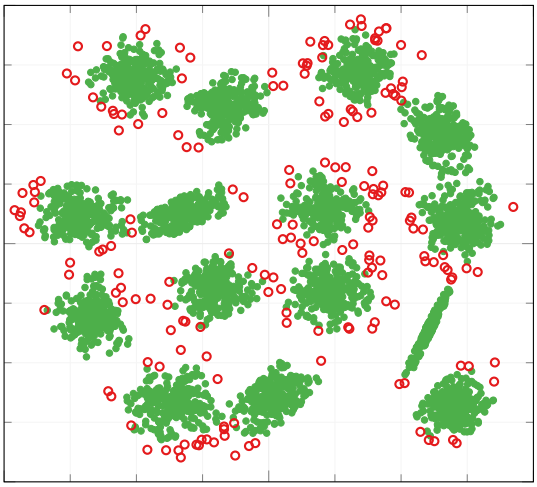
(f) Worst clusters proposed by K-means.



(g) Ground-truth vs. worst K-means.

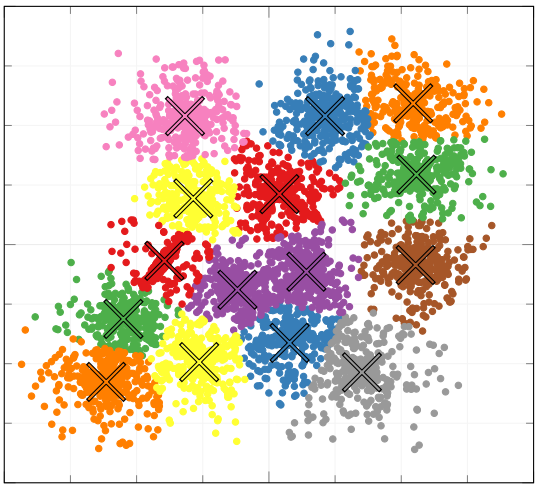


(h) Best clusters proposed by DBSCAN.



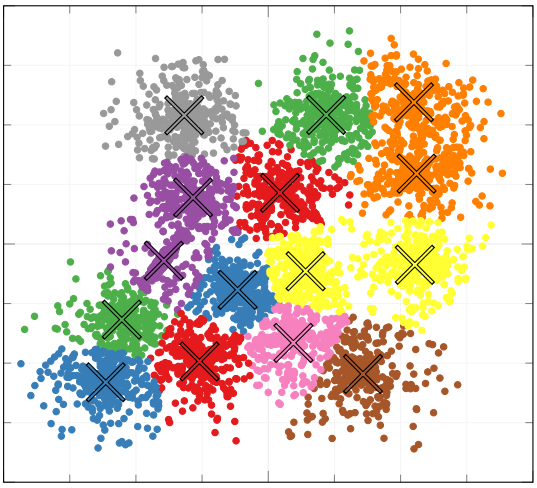
(i) Ground-truth vs. best DBSCAN.

Figure 9.25: Visualization of S1 clustering dataset.

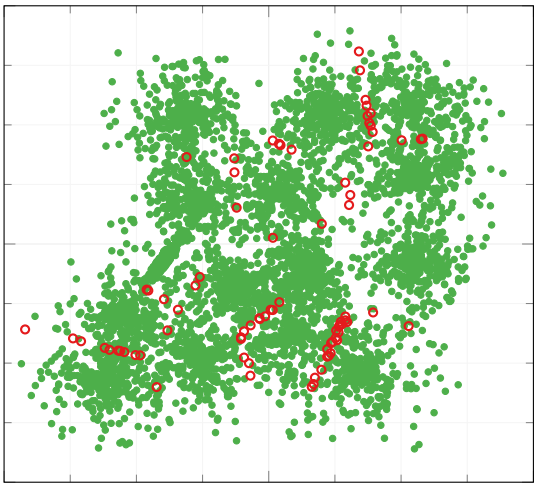


(a) Ground-truth of S3 dataset.

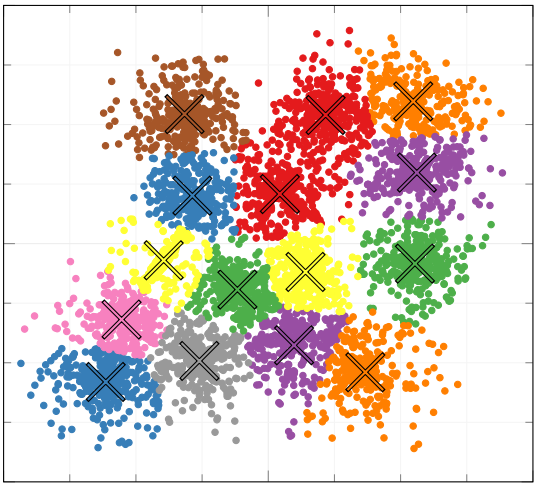
Figure 9.26: Visualization of S3 clustering dataset.



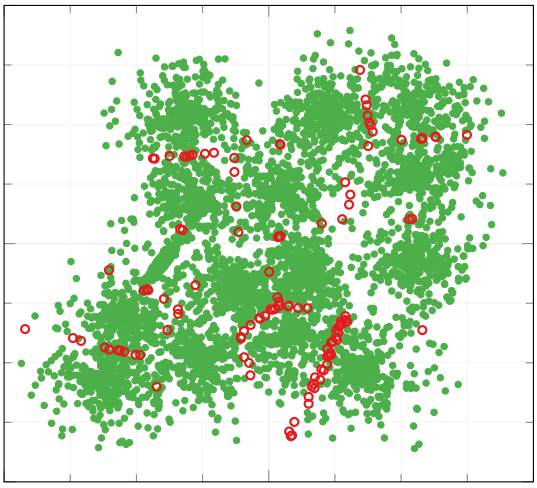
(b) Best clusters proposed by VND-GDE3.



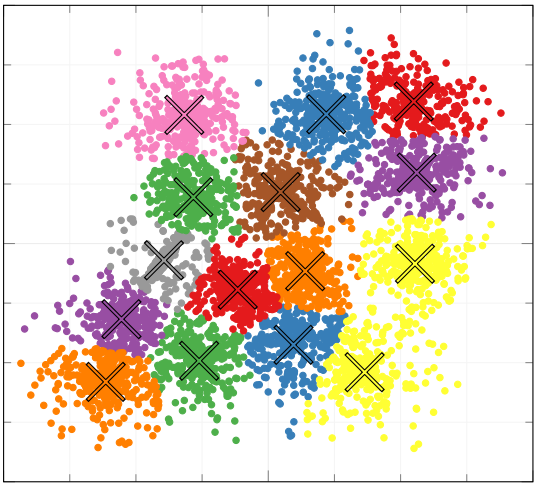
(c) Ground-truth vs. best VND-GDE3.



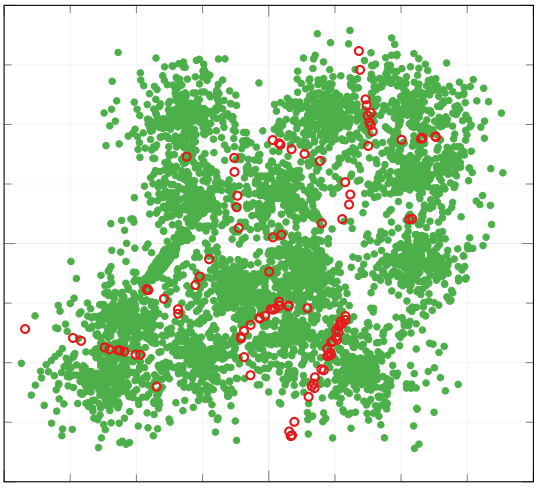
(d) Worst clusters proposed by VND-GDE3.



(e) Ground-truth vs. worst VND-GDE3.

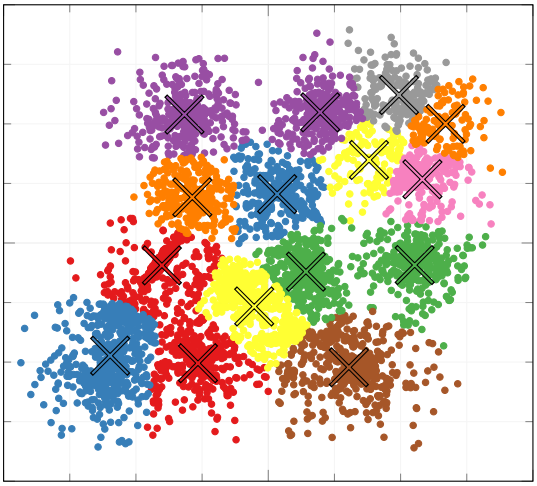


(f) Best clusters proposed by K-means.

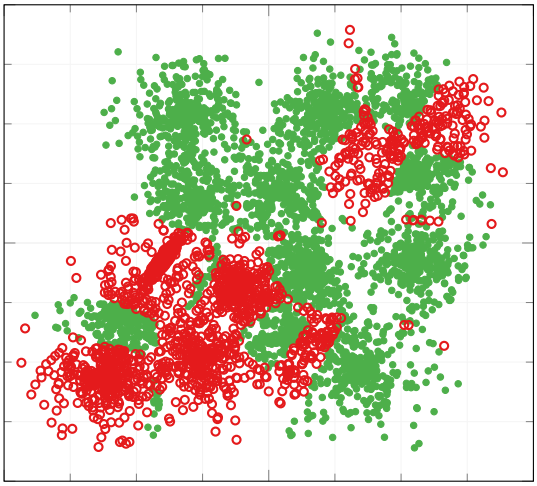


(g) Ground-truth vs. best K-means.

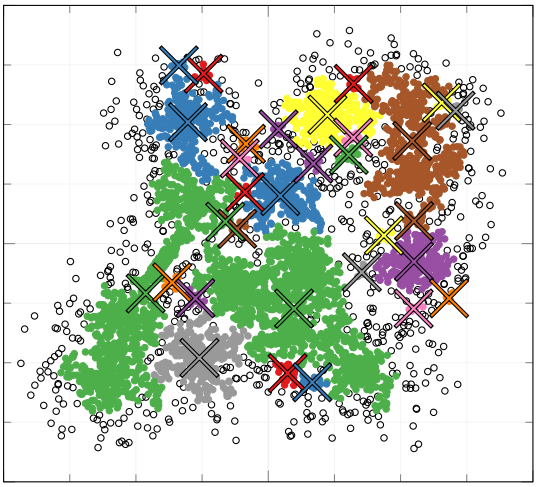
Figure 9.26: Visualization of S3 clustering dataset.



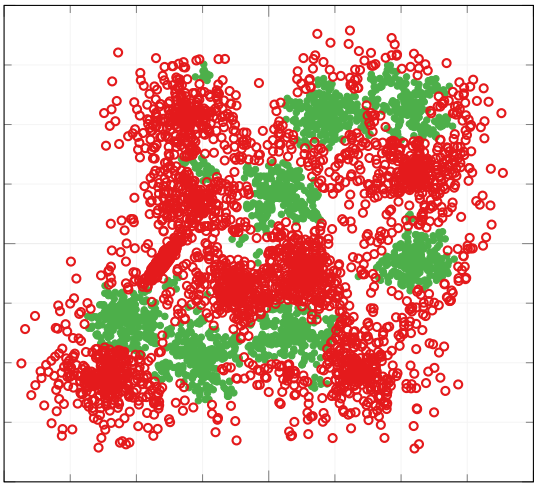
(h) Worst clusters proposed by K-means.



(i) Ground-truth vs. worst K-means.

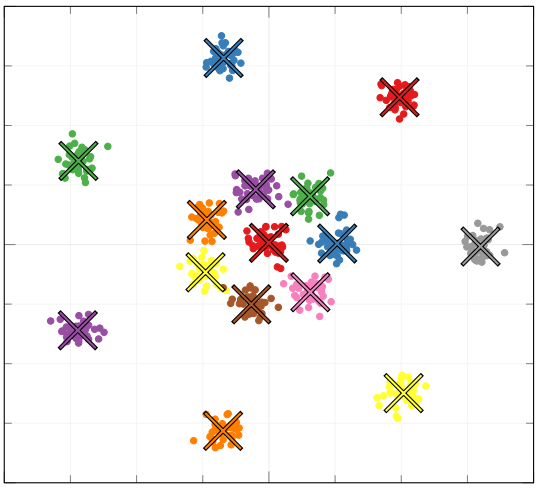


(j) Best clusters proposed by DBSCAN.



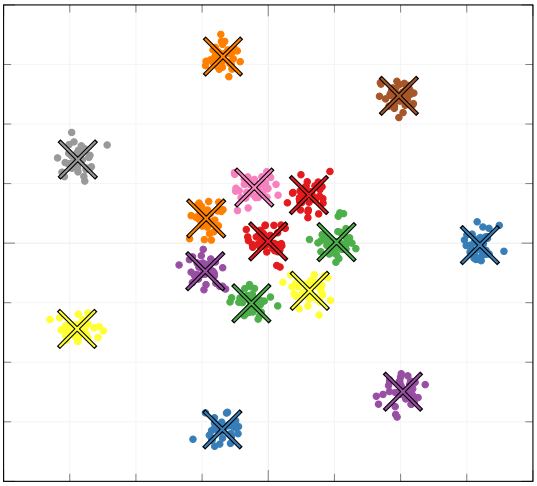
(k) Ground-truth vs. best DBSCAN.

Figure 9.26: Visualization of S3 clustering dataset.

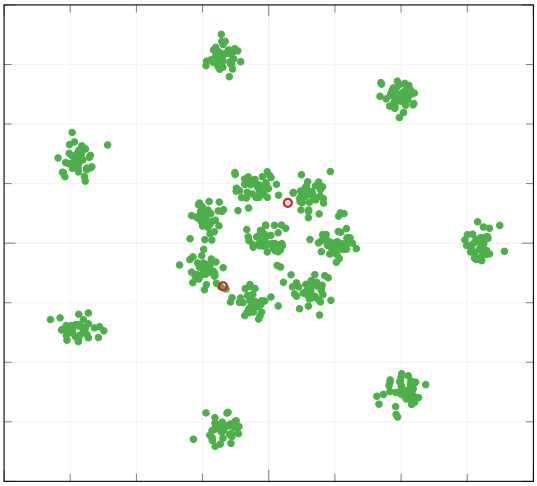


(a) Ground-truth of R15 dataset.

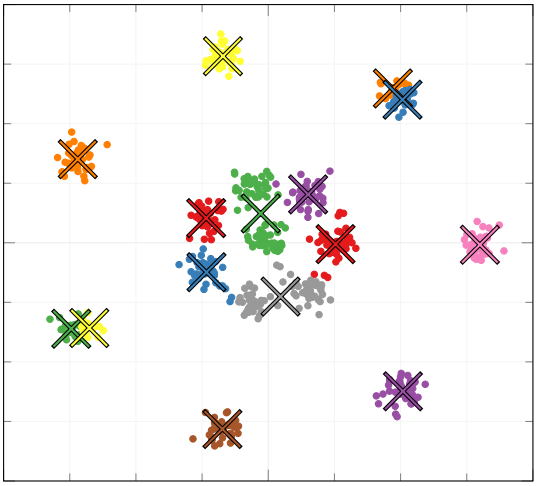
Figure 9.27: Visualization of R15 clustering dataset.



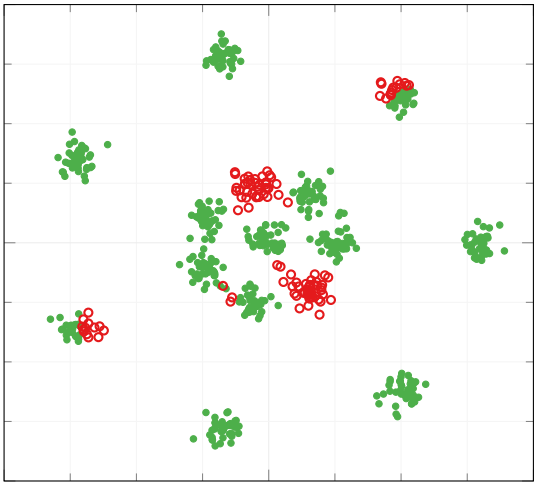
(b) Best clusters proposed by VND-GDE3.



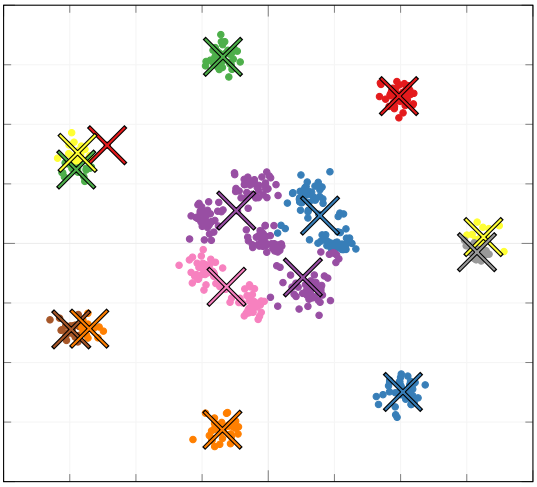
(c) Ground-truth vs. best VND-GDE3.



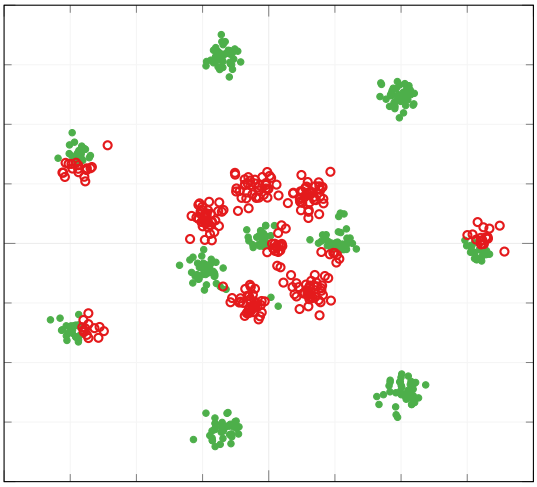
(d) Best clusters proposed by K-means.



(e) Ground-truth vs. best K-means.

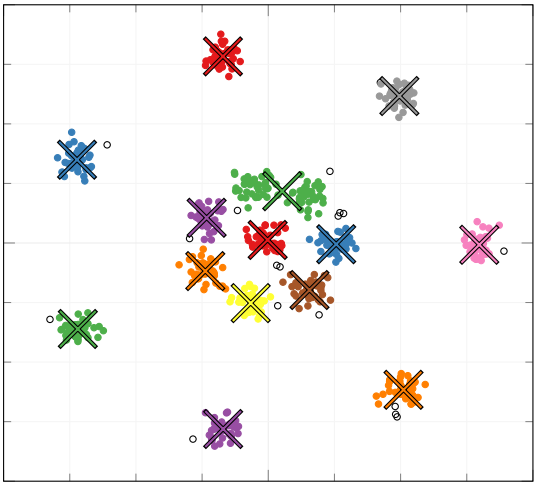


(f) Worst clusters proposed by K-means.

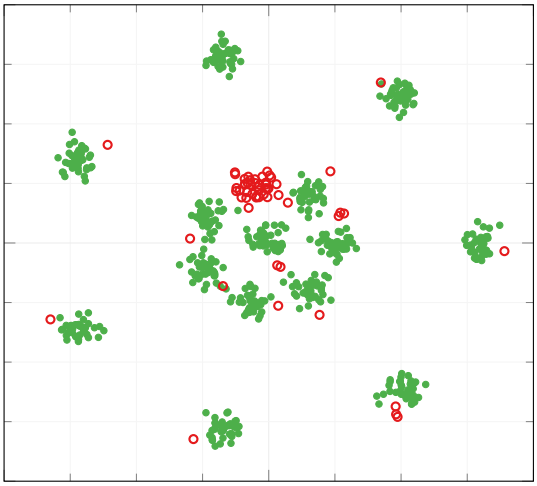


(g) Ground-truth vs. worst K-means.

Figure 9.27: Visualization of R15 clustering dataset.

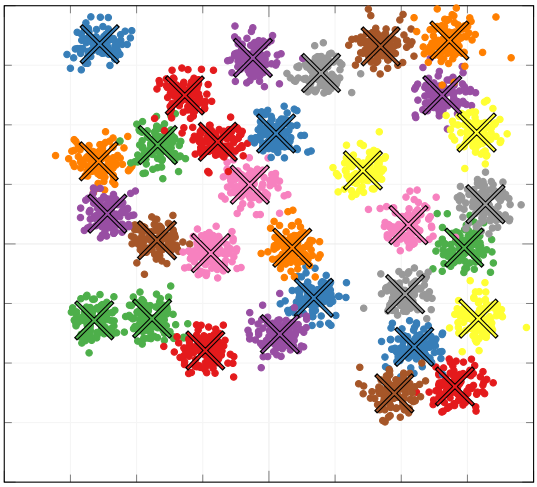


(h) Best clusters proposed by DBSCAN.

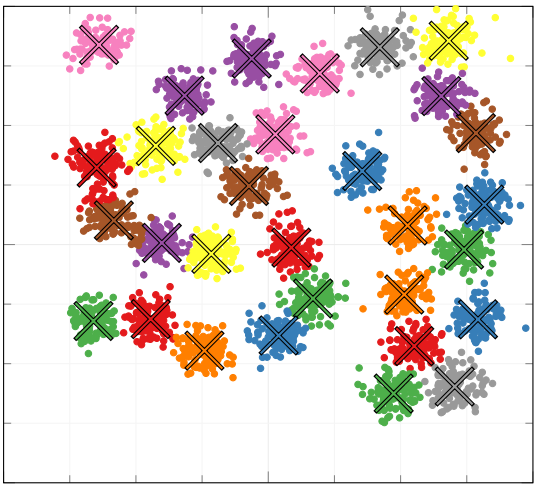


(i) Ground-truth vs. best DBSCAN.

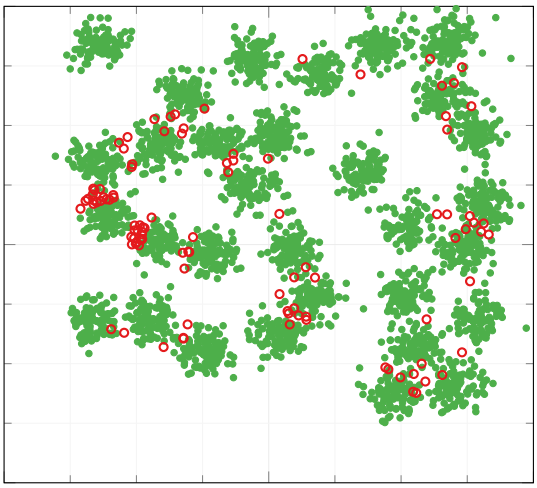
Figure 9.27: Visualization of R15 clustering dataset.



(a) Ground-truth of D31 dataset.

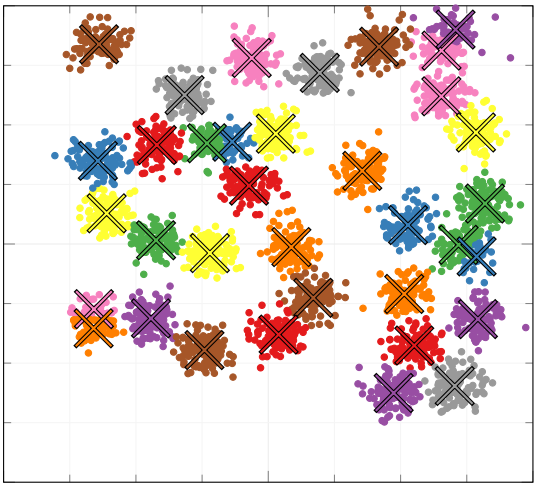


(b) Best clusters proposed by VND-GDE3.

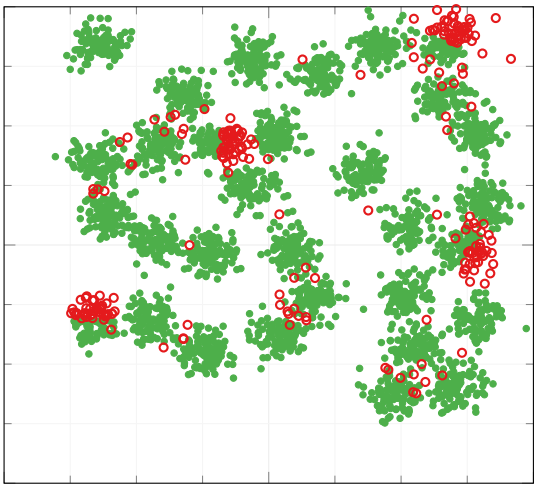


(c) Ground-truth vs. best VND-GDE3.

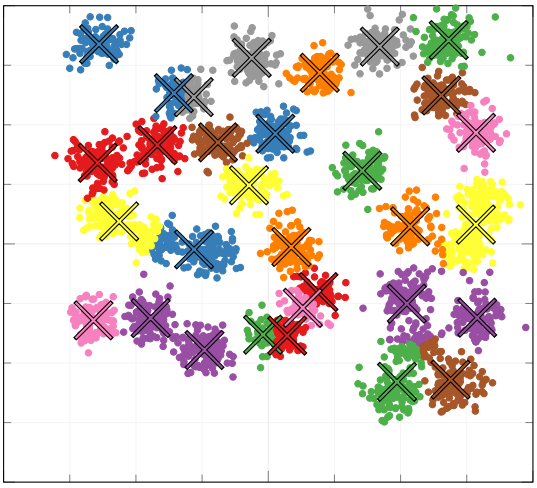
Figure 9.28: Visualization of D31 clustering dataset.



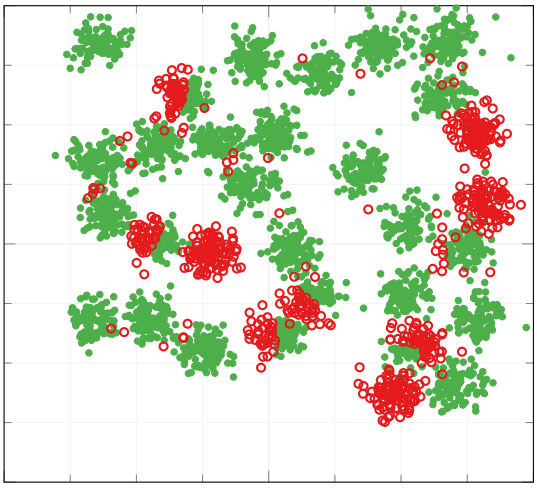
(d) Worst clusters proposed by VND-GDE3.



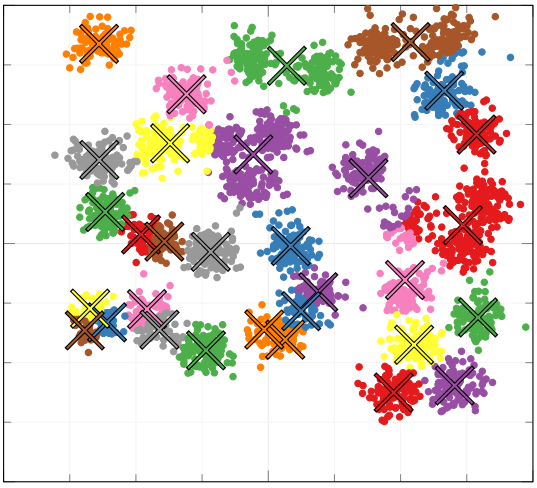
(e) Ground-truth vs. worst VND-GDE3.



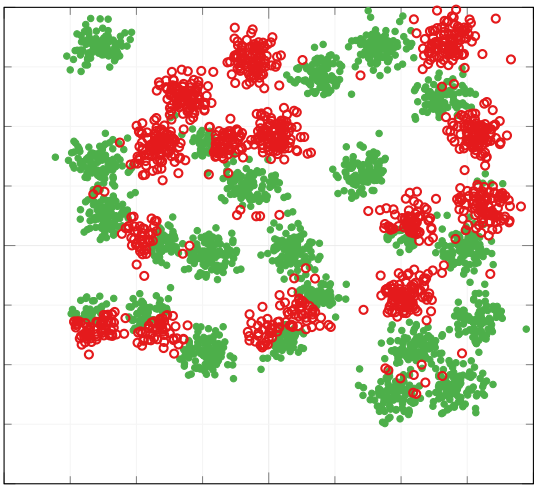
(f) Best clusters proposed by K-means.



(g) Ground-truth vs. best K-means.

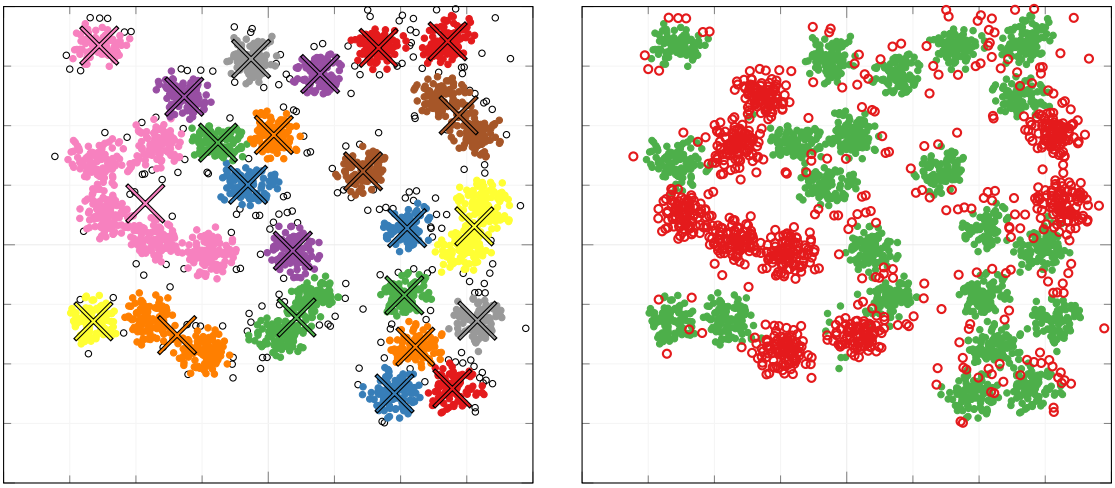


(h) Worst clusters proposed by K-means.



(i) Ground-truth vs. worst K-means.

Figure 9.28: Visualization of D31 clustering dataset.



(j) Best clusters proposed by DBSCAN. **(k)** Ground-truth vs. best DBSCAN.

Figure 9.28: Visualization of D31 clustering dataset.

10 CONCLUSION

This dissertation thesis deals with multi-objective evolutionary optimization with variable number of dimensions. Many real-life optimization tasks use decision vectors of variable length by nature. Although standard optimization algorithms with a fixed number of dimensions can solve such tasks, either the computational demands are much higher compared to the algorithms with a variable number of dimensions, or the representation of the problem has to be simplified. Therefore, the risk of losing decision space resolution emerges.

The idea of optimization algorithms with variable number of dimensions is probably as old as optimization algorithms itself. However, the research of optimization methods with variable number of dimensions is rather marginal compared to the fixed-length one. The survey of work in the field of optimization with a variable number of dimensions showed that there are many of them, but they are mostly only single-objective or can not work with the decision vectors of uneven lengths in the pure-VND nature.

Particle Swarm Optimization for Variable Number of Dimensions is one of the algorithms that is considered to be the pure-VND algorithm. However, it is a single-objective optimization algorithm. Nonetheless, the employed methodology for handling the vectors of different lengths was successfully applied in a multi-objective version of Particle Swarm Optimization. That gave birth to the VND-MOPSO algorithm. Similarly, a multi-objective Differential evolution-based algorithm with a variable number of dimensions was derived from GDE3 – the VND-GDE3 algorithm [62].

Both novel methods were verified by comparative studies against their impure-VND peers and also against the Clustered-GDE3 method. The Clustered-GDE3 method represents the standard algorithm with a fixed number of dimensions applied to problems with a variable number of dimensions. It was shown that a pure-VND algorithm outperforms both opponents, especially if the number of dimensionalities is large.

Before the comparison of the novel methods against others, the study of their setting parameters had to be carried out. Both these controlling parameters (the probability of dimensions transition in the VND-GDE3 and the probabilites to follow in the VND-MOPSO) control the behavior of the algorithm in search of the optimal dimensionality of the problem. Studies of other controlling parameters, common with non-VND peers, are unnecessary because the VND methodology does not change the basic principles of the corresponding predecessor. Therefore, the setting of parameters from various studies in the literature still applies.

The verification of the methods utilizes a library of testing problems with a variable number of dimensions. This library was created by modifying the well-known libraries for multi-objective optimization (namely: DTLZ, LZ, UF, and ZDT libraries). Therefore, the convergence properties of such testing problems are ensured. The methodology is well described and is easily applicable to any scalable multi-objective problem.

All the methods and testing problems are included in the FOPS optimization frame-

work [MM2, MM4]. The development of the framework is an important part of this thesis, although its development began before my doctoral studies. The reason for the creation of FOPS is that there did not exist any framework where optimization methods with a variable number of dimensions could be implemented. The use of a framework is essential if various and numerous comparative studies are to be composed, executed, and visualized.

FOPS is a unique tool for the optimization of all kinds. It has already been exploited in various papers [MM8, MM9, MM1, MM10, MM11, MM12, MM13, MM14, MM15]. The last chapter of this thesis presents several real-life applications published in [MM7, 63, 64]. All of them were carried out in the FOPS framework. This demonstrates the versatility of the FOPS. Moreover, most of the applications are problems with a variable number of dimensions from the field of electrical engineering.

The first VND application is the Optimal placement of transmitters. It was published in [MM2] This problem is a perfect demonstration of the VND problem that can not be tackled with a non-VND algorithm without considerable limitations. Either the number of transmitters is defined a priori, or the decision space is sampled so the transmitters at predefined positions can be enabled or disabled. The sampling of the decision space is shown in the next application – the linear antenna array problem.

The linear antenna array problem presents a synthesis of dipole array where the side-lobe level and the number of active dipoles are optimized. The problem shows two different representations of the problem – representation with a variable number of dimensions tackled by the VND-GDE3 algorithm and representation with a fixed number of dimensions using a uniform-grid. Uniform-grid representation is tackled by the standard GDE3 algorithm. It is shown that better values of Side-lobe Level with fewer antennas used are achieved with the VND-GDE3 algorithm. The problems was published in [62] and [MM12].

Another application is the synthesis of digital circuits. In this application, the 3-input, 6-input, and 11-input multiplexers were synthesized by the VND-GDE3 algorithm. The number of product terms in the SOP expression was arbitrary. Therefore, the digital circuits were synthesized without the use of Karnaugh maps or other time-consuming methods.

The next-to-last application is the automated image thresholding. The standard, exhaustive approach is compared to the evolutionary approach in the first part. Afterward, the optimization algorithm with a variable number of dimensions is used to segment the testing images by multiple thresholds, and the last part utilizes the multiple threshold approach in license-plate recognition.

Finally, the clustering problem is solved by a multi-objective evolutionary algorithm with a variable number of dimensions. The evolutionary approach eliminates the main disadvantage of the widely used K-means clustering method. Several clustering benchmark datasets were tackled by two standard clustering methods and one exploiting the VND-GDE3 algorithm. It was shown that for most clustering datasets, the VND-GDE3 approach was the most successful method.

BIBLIOGRAPHY

[1] V. Pareto, *The mind and society*, vol. 1. Jonathan Cape, London, 1935.

[2] D. E. Goldberg and J. H. Holland, "Genetic algorithms and machine learning," *Machine learning*, vol. 3, no. 2, pp. 95–99, 1988.

[3] R. J. Kennedy and Eberhart, "Particle swarm optimization," in *Proceedings of IEEE International Conference on Neural Networks IV*, vol. 1000, 1995.

[4] R. Storn and K. Price, "Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces," *Journal of global optimization*, vol. 11, no. 4, pp. 341–359, 1997.

[5] K. Deb, *Multi-objective optimization using evolutionary algorithms*, vol. 16. John Wiley & Sons, 2001.

[6] K. Miettinen, *Nonlinear multiobjective optimization*, vol. 12. Springer Science & Business Media, 2012.

[7] Y. Y. Haimes, "On a bicriterion formulation of the problems of integrated system identification and system optimization," *IEEE transactions on systems, man, and cybernetics*, vol. 1, no. 3, pp. 296–297, 1971.

[8] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

[9] M. Reyes-Sierra and C. C. Coello, "Multi-objective particle swarm optimizers: A survey of the state-of-the-art," *International journal of computational intelligence research*, vol. 2, no. 3, pp. 287–308, 2006.

[10] S. Kukkonen and J. Lampinen, "GDE3: The third evolution step of generalized differential evolution," in *Evolutionary Computation, 2005. The 2005 IEEE Congress on*, vol. 1, pp. 443–450, IEEE, 2005.

[11] A. Zhou, B.-Y. Qu, H. Li, S.-Z. Zhao, P. N. Suganthan, and Q. Zhang, "Multiobjective evolutionary algorithms: A survey of the state of the art," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 32–49, 2011.

[12] H.-T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *Journal of the ACM (JACM)*, vol. 22, no. 4, pp. 469–476, 1975.

[13] S. Kukkonen and K. Deb, "A fast and effective method for pruning of non-dominated solutions in many-objective problems," in *Parallel Problem Solving from Nature-PPSN IX*, pp. 553–562, Springer, 2006.

[14] D. A. Van Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a Pareto front," in *Late breaking papers at the genetic programming 1998 conference*, pp. 221–228, Citeseer, 1998.

[15] E. Zitzler, *Evolutionary algorithms for multiobjective optimization: Methods and applications*, vol. 63. Citeseer, 1999.

[16] J. D. Knowles, L. Thiele, and E. Zitzler, "A tutorial on the performance assessment of stochastic multiobjective optimizers," *TIK-Report*, vol. 214, 2006.

[17] J. Derrac, S. García, D. Molina, and F. Herrera, "A practical tutorial on the use of nonparametric statistical tests as a methodology for comparing evolutionary and swarm intelligence algorithms," *Swarm and Evolutionary Computation*, vol. 1, no. 1, pp. 3–18, 2011.

[18] J. H. Zar, *Biostatistical analysis*. Pearson Education India, 1999.

[19] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[20] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial intelligence through simulated evolution.* John Wiley & Sons, 1966.

[21] D. E. Goldberg, B. Korb, K. Deb, *et al.*, "Messy genetic algorithms: Motivation, analysis, and first results," *Complex systems*, vol. 3, no. 5, pp. 493–530, 1989.

[22] I. Harvey, "Species adaptation genetic algorithms: A basis for a continuing saga," in *Toward a practice of autonomous systems: Proceedings of the first european conference on artificial life*, pp. 346–354, 1992.

[23] D. Dasgupta and D. R. McGregor, "Nonstationary function optimization using the structured genetic algorithm.," in *PPSN*, vol. 2, pp. 145–154, Citeseer, 1992.

[24] J. R. Koza, "Evolution of subsumption using genetic programming," in *Proceedings of the First European Conference on Artificial Life*, pp. 110–119, 1992.

[25] E. Falkenauer and A. Delchambre, "A genetic algorithm for bin packing and line balancing," in *Proceedings 1992 IEEE International Conference on Robotics and Automation*, pp. 1186–1192, IEEE, 1992.

[26] D. S. Burke, K. A. De Jong, J. J. Grefenstette, C. L. Ramsey, and A. S. Wu, "Putting more genetics into genetic algorithms," *Evolutionary Computation*, vol. 6, no. 4, pp. 387–410, 1998.

[27] B. Hutt and K. Warwick, "Synapsing variable-length crossover: Meaningful crossover for variable-length genomes," *IEEE transactions on evolutionary computation*, vol. 11, no. 1, pp. 118–131, 2007.

[28] D. A. Van Veldhuizen and G. B. Lamont, "Multiobjective optimization with messy genetic algorithms," in *Symposium on Applied Computing: Proceedings of the 2000 ACM symposium on Applied computing-*, vol. 1, pp. 470–476, Citeseer, 2000.

[29] K. N. Ripon, C.-H. Tsang, S. Kwong, and M.-K. Ip, "Multi-objective evolutionary clustering using variable-length real jumping genes genetic algorithm," in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 1, pp. 1200–1203, IEEE, 2006.

[30] C.-K. Ting, C.-N. Lee, H.-C. Chang, and J.-S. Wu, "Wireless heterogeneous transmitter placement using multiobjective variable-length genetic algorithm," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 4, pp. 945–958, 2009.

[31] K. E. Mathias and L. D. Whitley, "Initial performance comparisons for the delta coding algorithm," in *Proceedings of the First IEEE Conference on Evolutionary Computation. IEEE World Congress on Computational Intelligence*, pp. 433–438, IEEE, 1994.

[32] M. Schütz, "Other operators: Gene duplication and deletion," *The Handbook of Evolutionary Computation*, 1995.

[33] C. Lee and E. Antonsson, "Variable length genomes for evolutionary algorithms.," in *GECCO*, vol. 2000, p. 806, 2000.

[34] R. S. Zebulum, M. Vellasco, and M. A. Pacheco, "Variable length representation in evolutionary electronics," *Evolutionary Computation*, vol. 8, no. 1, pp. 93–120, 2000.

[35] D. Chu and J. E. Rowe, "Crossover operators to control size growth in linear GP and variable length GAs," in *2008 IEEE Congress on Evolutionary Computation (IEEE World Congress on Computational Intelligence)*, pp. 336–343, IEEE, 2008.

[36] J. Decraene, M. Chandramohan, F. Zeng, M. Y. H. Low, and W. Cai, "Evolving agent-based model structures using variable-length genomes," in *Proceedings of the 4th International Workshop on Optimisation in Multi-Agent Systems at AAMAS*, vol. 11, 2011.

[37] A. Ghaffarizadeh, K. Ahmadi, and N. S. Flann, "Sorting unsigned permutations by reversals using multi-objective evolutionary algorithms with variable size individuals," in *2011 IEEE Congress of Evolutionary Computation (CEC)*, pp. 292–295, IEEE, 2011.

[38] Z. Qiongbing and D. Lixin, "A new crossover mechanism for genetic algorithms with variable-length chromosomes for path optimization problems," *Expert Systems with Applications*, vol. 60, pp. 183–189, 2016.

[39] J. Kennedy and R. C. Eberhart, "A discrete binary version of the particle swarm algorithm," in *1997 IEEE International conference on systems, man, and cybernetics. Computational cybernetics and simulation*, vol. 5, pp. 4104–4108, IEEE, 1997.

[40] A. Mukhopadhyay and M. Mandal, "Identifying non-redundant gene markers from microarray data: a multiobjective variable length pso-based approach," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 11, no. 6, pp. 1170–1183, 2014.

[41] Z. Yangyang, J. Chunlin, Y. Ping, L. Manlin, W. Chaojin, and W. Guangxing, "Particle swarm optimization for base station placement in mobile communication," in *IEEE International Conference on Networking, Sensing and Control, 2004*, vol. 1, pp. 428–432, IEEE, 2004.

[42] M. Omran, A. Salman, and A. Engelbrecht, "Dynamic clustering using particle swarm optimization with application in unsupervised image classification," in *Fifth World Enformatika Conference (ICCI 2005), Prague, Czech Republic*, pp. 199–204, 2005.

[43] A. Abraham, S. Das, and S. Roy, "Swarm intelligence algorithms for data clustering," in *Soft computing for knowledge discovery and data mining*, pp. 279–313, Springer, 2008.

[44] Y. Yan and L. A. Osadciw, "Density estimation using a new dimension adaptive particle swarm optimization algorithm," *Swarm Intelligence*, vol. 3, no. 4, p. 275, 2009.

[45] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Fractional particle swarm optimization in multidimensional search space," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 40, no. 2, pp. 298–319, 2009.

[46] F. Van Den Bergh *et al.*, *An analysis of particle swarm optimizers*. PhD thesis, University of Pretoria South Africa, 2001.

[47] J. Riget and J. S. Vesterstrøm, "A diversity-guided particle swarm optimizer-the ARPSO," *Dept. Comput. Sci., Univ. of Aarhus, Aarhus, Denmark, Tech. Rep*, vol. 2, p. 2002, 2002.

[48] P. Kadlec and V. Šeděnka, "Particle swarm optimization for problems with variable number of dimensions," *Engineering Optimization*, vol. 50, no. 3, pp. 382–399, 2017.

[49] M. O'Neill and A. Brabazon, "Grammatical differential evolution.," in *IC-AI*, pp. 231–236, 2006.

[50] Y. Chen, V. Mahalec, Y. Chen, X. Liu, R. He, and K. Sun, "Reconfiguration of satellite orbit for cooperative observation using variable-size multi-objective differential evolution," *European Journal of Operational Research*, vol. 242, no. 1, pp. 10–20, 2015.

[51] B. Wang, Y. Sun, B. Xue, and M. Zhang, "A hybrid differential evolution approach to designing deep convolutional neural networks for image classification," in *Australasian Joint Conference on Artificial Intelligence*, pp. 237–250, Springer, 2018.

[52] I. Landa-Torres, S. Gil-Lopez, S. Salcedo-Sanz, J. Del Ser, and J. A. Portilla-Figueras, "A novel grouping harmony search algorithm for the multiple-type access node location problem," *Expert Systems with Applications*, vol. 39, no. 5, pp. 5262–5270, 2012.

[53] I. Landa-Torres, D. Manjarres, S. Salcedo-Sanz, J. Del Ser, and S. Gil-Lopez, "A multi-objective grouping harmony search algorithm for the optimal distribution of 24-hour medical emergency units," *Expert systems with applications*, vol. 40, no. 6, pp. 2343–2349, 2013.

[54] S. Salcedo-Sanz, P. García-Díaz, J. Del Ser, M. N. Bilbao, and J. A. Portilla-Figueras, "A novel grouping coral reefs optimization algorithm for optimal mobile network deployment problems under electromagnetic pollution and capacity control criteria," *Expert Systems with Applications*, vol. 55, pp. 388–402, 2016.

[55] C. H. Antunes, P. Lima, E. Oliveira, and D. F. Pires, "A multi-objective simulated annealing approach to reactive power compensation," *Engineering Optimization*, vol. 43, no. 10, pp. 1063–1077, 2011.

[56] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," in *Micro Machine and Human Science, 1995. MHS'95., Proceedings of the Sixth International Symposium on*, pp. 39–43, IEEE, 1995.

[57] M. Gheith, A. B. Eltawil, and N. A. Harraz, "Solving the container pre-marshalling problem using variable length genetic algorithms," *Engineering Optimization*, vol. 48, no. 4, pp. 687–705, 2016.

[58] A. J. Nebro, E. Alba, G. Molina, F. Chicano, F. Luna, and J. J. Durillo, "Optimal antenna placement using a new multi-objective chc algorithm," in *Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pp. 876–883, ACM, 2007.

[59] B. Rekiek, P. De Lit, F. Pellichero, T. L'Eglise, P. Fouda, E. Falkenauer, and A. Delchambre, "A multiple objective grouping genetic algorithm for assembly line design," *Journal of intelligent manufacturing*, vol. 12, no. 5-6, pp. 467–485, 2001.

[60] J. Handl and J. Knowles, "Evolutionary multiobjective clustering," in *International Conference on Parallel Problem Solving from Nature*, pp. 1081–1091, Springer, 2004.

[61] J. S. Arora, *Introduction to optimum design*. Elsevier, 2004.

[62] M. Marek and P. Kadlec, "Another evolution of generalized differential evolution," *Engineering Optimization*, vol. 53, to be published, accepted on 6th November 2020.

[63] M. Marek and P. Kadlec, "Discretization of decision variables in optimization algorithms," in *Proceedings of the 24th Conference STUDENT EEICT 2018*, pp. 320–324, april 2018.

[64] M. Marek and P. Kadlec, "Using a tolerance-based surrogate method for computer resources saving in optimization," *Radioengineering*, vol. 28, no. 1, pp. 9–18, 2019.

[65] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997.

[66] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler, "PISA: A platform and programming language independent interface for search algorithms," in *Evolutionary Multi-Criterion Optimization* (C. M. Fonseca, P. J. Fleming, E. Zitzler, L. Thiele, and K. Deb, eds.), (Berlin, Heidelberg), pp. 494–508, Springer Berlin Heidelberg, 2003.

[67] J. J. Durillo and A. J. Nebro, "jMetal: A java framework for multi-objective optimization," *Advances in Engineering Software*, vol. 42, no. 10, pp. 760–771, 2011.

[68] A. Liefooghe, M. Basseur, L. Jourdan, and E.-G. Talbi, "ParadisEO-MOEO: A framework for evolutionary multi-objective optimization," in *International Conference on Evolutionary Multi-criterion Optimization*, pp. 386–400, Springer, 2007.

[69] R. Shen, J. Zheng, and M. Li, "A hybrid development platform for evolutionary multi-objective optimization," in *Evolutionary Computation (CEC), 2015 IEEE Congress on*, pp. 1885–1892, IEEE, 2015.

[70] Y. Tian, R. Cheng, X. Zhang, and Y. Jin, "PlatEMO: A MATLAB platform for evolutionary multi-objective optimization," *IEEE Computational Intelligence Magazine*, vol. 12, no. 4, pp. 73–87, 2017.

[71] D. Gorissen, I. Couckuyt, P. Demeester, T. Dhaene, and K. Crombecq, "A surrogate modeling and adaptive sampling toolbox for computer based design," *Journal of Machine Learning Research*, vol. 11, no. Jul, pp. 2051–2055, 2010.

[72] A. HyperStudy, "v7. 0," *Altair Inc., Troy, MI, USA*, 2018.

[73] J. A. Nelder and R. Mead, "A simplex method for function minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[74] M. Gen and R. Cheng, *Genetic algorithms and engineering optimization*, vol. 7. John Wiley & Sons, 2000.

[75] I. Zelinka, "SOMA: self-organizing migrating algorithm," in *New Optimization Techniques in Engineering*, pp. 167–217, Springer, 2004.

[76] N. Hansen, S. D. Müller, and P. Koumoutsakos, "Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES)," *Evolutionary computation*, vol. 11, no. 1, pp. 1–18, 2003.

[77] P. Kadlec and Z. Raida, "A novel multi-objective self-organizing migrating algorithm.," *Radioengineering*, vol. 20, no. 4, 2011.

[78] M. Molga and C. Smutnicki, "Test functions for optimization needs," *Test Functions For Optimization Needs*, vol. 101, 2005.

[79] J. Momin and X.-S. Yang, "A literature survey of benchmark functions for global optimization problems," *Journal of Mathematical Modelling and Numerical Optimisation*, vol. 4, no. 2, pp. 150–194, 2013.

[80] S. Kiranyaz, T. Ince, A. Yildirim, and M. Gabbouj, "Fractional particle swarm optimization in multidimensional search space," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 40, no. 2, pp. 298–319, 2010.

[81] K. Deb, L. Thiele, M. Laumanns, and E. Zitzler, "Scalable multi-objective optimization test problems," in *Evolutionary Computation, 2002. CEC'02. Proceedings of the 2002 Congress on*, vol. 1, pp. 825–830, IEEE, 2002).

[82] H. Li and Q. Zhang, "Multiobjective optimization problems with complicated Pareto sets, MOEA/D and NSGA-II," *IEEE Transactions on evolutionary computation*, vol. 13, no. 2, pp. 284–302, 2009.

[83] Q. Zhang, A. Zhou, S. Zhao, P. N. Suganthan, W. Liu, and S. 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*, vol. 264, 2008.

[84] S. Huband, P. Hingston, L. Barone, and L. While, "A review of multiobjective test problems and a scalable test problem toolkit," *IEEE Transactions on Evolutionary Computation*, vol. 10, no. 5, pp. 477–506, 2006.

[85] E. Zitzler, K. Deb, and L. Thiele, "Comparison of multiobjective evolutionary algorithms: Empirical results," *Evolutionary Computation*, vol. 8, no. 2, pp. 173–195, 2000.

[86] H. Li and K. Deb, "Challenges for evolutionary multiobjective optimization algorithms for solving variable-length problems," in *Evolutionary Computation (CEC), 2017 IEEE Congress on*, pp. 2217–2224, IEEE, 2017.

[87] V. Šeděnka and Z. Raida, "Critical comparison of multi-objective optimization methods: Genetic algorithms versus swarm intelligence.," *Radioengineering*, vol. 19, no. 3, 2010.

[88] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems," *Journal of global optimization*, vol. 31, no. 4, pp. 635–672, 2005.

[89] C. Caloz and T. Itoh, "Multilayer and anisotropic planar compact PBG structures for microstrip applications," *IEEE Transactions on Microwave Theory and Techniques*, vol. 50, no. 9, pp. 2206–2208, 2002.

[90] S. Orfanidis, *Electromagnetic Waves and Antennas*. Rutgers University, 2002.

[91] A. Hussein, H. Abdullah, A. Salem, S. Khamis, and M. Nasr, "Optimum design of linear antenna arrays using a hybrid MoM/GA algorithm," *IEEE Antennas and Wireless Propagation Letters*, vol. 10, pp. 1232–1235, 2011.

[92] Z. Wang, Y. Sun, X. Yang, and S. Li, "Hybrid optimisation method of improved genetic algorithm and IFT for linear thinned array," *The Journal of Engineering*, vol. 2019, no. 20, pp. 6457–6460, 2019.

[93] J. F. Miller, P. Thomson, and T. Fogarty, "Designing electronic circuits using evolutionary algorithms. arithmetic circuits: A case study," 1997.

[94] M. Karnaugh, "The MAP method for synthesis of combinational logic circuits," *Transactions of the American Institute of Electrical Engineers, Part I: Communication and Electronics*, vol. 72, no. 5, pp. 593–599, 1953.

[95] W. V. Quine, "The problem of simplifying truth functions," *The American mathematical monthly*, vol. 59, no. 8, pp. 521–531, 1952.

[96] W. J. Masek, "Some NP-complete set covering problems," *Unpublished manuscript*, 1979.

[97] M. Sezgin and B. Sankur, "Survey over image thresholding techniques and quantitative performance evaluation," *Journal of Electronic imaging*, vol. 13, no. 1, pp. 146–166, 2004.

[98] F. Petitcolas, *Public-Domain Test Images for Homeworks and Projects*. Digital Image Processing at University of Wisconsin-Madison, 2012. [Online; accessed November 19, 2020].

[99] M. Kamel and A. Zhao, "Extraction of binary character/graphics images from grayscale document images," *CVGIP: Graphical Models and Image Processing*, vol. 55, no. 3, pp. 203–217, 1993.

[100] J.-S. Chang, H.-Y. M. Liao, M.-K. Hor, J.-W. Hsieh, and M.-Y. Chern, "New automatic multi-level thresholding technique for segmentation of thermal images," *Image and vision computing*, vol. 15, no. 1, pp. 23–34, 1997.

[101] W. Oh and B. Lindquist, "Image thresholding by indicator kriging," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 7, pp. 590–602, 1999.

[102] S. Abdullah, F. PirahanSiah, N. Z. Abidin, and S. Sahran, "Multi-threshold approach for license plate recognition system," in *International Conference on Signal and Image Processing WASET Singapore August*, pp. 25–27, 2010.

[103] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[104] A. Weber, *SIPI Image Database*. Signal and Image Processing Institute at University of Southern California, 1977. [Online; accessed November 19, 2020].

[105] W. Burger and M. J. Burge, *Digital image processing: an algorithmic introduction using Java*. Springer Science & Business Media, 2009.

[106] J. Qin, C. Wang, and G. Qin, "A multilevel image thresholding method based on subspace elimination optimization," *Mathematical Problems in Engineering*, vol. 2019, 2019.

[107] K. Omar, A. S. Zaini, M. Petrou, and M. Khalid, "Determining adaptive thresholds for image segmentation for a license plate recognition system," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 6, pp. 510–523, 2016.

[108] K. Aboura, "Automatic thresholding of license plate image data," in *The Eighth International Conference on Intelligent Technologies*, pp. 332–339, 2007.

[109] J. Parker and P. Federl, "An approach to license plate recognition," *University of Calgary*, 1996.

[110] P. Plashkin, *Car parked to contemplate the night views of sky full of stars*. unsplash.com, October 2019. [Online; accessed November 19, 2020].

[111] F. Albert, *Black BMW M3 on road*. unsplash.com, May 2020. [Online; accessed November 19, 2020].

[112] S. Patel, *Middle of the road*. unsplash.com, May 2020. [Online; accessed November 19, 2020].

[113] E. O'Donnell, *Grey taxi during nighttime*. unsplash.com, May 2019. [Online; accessed November 19, 2020].

[114] A. K. Jain and R. C. Dubes, *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[115] C. Carpineto and G. Romano, "A lattice conceptual clustering system and its application to browsing retrieval," *Machine learning*, vol. 24, no. 2, pp. 95–122, 1996.

[116] G. B. Coleman and H. C. Andrews, "Image segmentation by clustering," *Proceedings of the IEEE*, vol. 67, no. 5, pp. 773–785, 1979.

[117] M. J. Brusco, E. Shireman, and D. Steinley, "A comparison of latent class, K-means, and K-median methods for clustering dichotomous data.," *Psychological methods*, vol. 22, no. 3, p. 563, 2017.

[118] Z. Feng and J. Zhang, "Nonparametric K-means algorithm with applications in economic and functional data," *Communications in Statistics-Theory and Methods*, pp. 1–15, 2020.

[119] R. Xu and D. Wunsch, "Survey of clustering algorithms," *IEEE Transactions on neural networks*, vol. 16, no. 3, pp. 645–678, 2005.

[120] S. Lloyd, "Least squares quantization in PCM," *IEEE transactions on information theory*, vol. 28, no. 2, pp. 129–137, 1982.

[121] K. Fukunaga and L. Hostetler, "The estimation of the gradient of a density function, with applications in pattern recognition," *IEEE Transactions on information theory*, vol. 21, no. 1, pp. 32–40, 1975.

[122] M. Ester, H.-P. Kriegel, J. Sander, X. Xu, *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise.," in *KDD*, pp. 226–231, 1996.

[123] T. Caliński and J. Harabasz, "A dendrite method for cluster analysis," *Communications in Statistics-theory and Methods*, vol. 3, no. 1, pp. 1–27, 1974.

[124] J. Baarsch and M. E. Celebi, "Investigation of internal validity measures for K-means clustering," in *Proceedings of the international multiconference of engineers and computer scientists*, vol. 1, pp. 14–16, sn, 2012.

[125] L. Fu and E. Medico, "Flame, a novel fuzzy clustering method for the analysis of DNA microarray data," *BMC bioinformatics*, vol. 8, no. 1, p. 3, 2007.

[126] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering aggregation," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, pp. 4–es, 2007.

[127] M. Rezaei and P. Fränti, "Set-matching methods for external cluster validity," *IEEE Trans. on Knowledge and Data Engineering*, vol. 28, no. 8, pp. 2173–2186, 2016.

[128] P. Fränti and O. Virmajoki, "Iterative shrinking method for clustering problems," *Pattern Recognition*, vol. 39, no. 5, pp. 761–765, 2006.

[129] C. J. Veenman, M. J. T. Reinders, and E. Backer, "A maximum variance cluster algorithm," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 24, no. 9, pp. 1273–1280, 2002.

[130] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[131] D. Lyusov, *Silver Caddy on green grass field.* unsplash.com, June 2020. [Online; accessed November 19, 2020].

[132] R. Laroca, E. Severo, L. A. Zanlorensi, L. S. Oliveira, G. R. Gonçalves, W. R. Schwartz, and D. Menotti, "A robust real-time automatic license plate recognition based on the YOLO detector," in *2018 international joint conference on neural networks (IJCNN)*, pp. 1–10, IEEE, 2018.

[133] I. Tanaskovic, *Nissan GTR R35*. unsplash.com, May 2020. [Online; accessed November 19, 2020].

[134] P. Tran, *Greyscale photo of Mercedes-Benz*. unsplash.com, August 2020. [Online; accessed November 19, 2020].

[135] J. Rhyner, *Silver Porsche 911 parked on street during daytime*. unsplash.com, August 2020. [Online; accessed November 19, 2020].

AUTHOR'S BIBLIOGRAPHY

[MM1] P. Kadlec, M. Marek, M. Štumpf, and V. Šeděnka, "PCB decoupling optimization with variable number of capacitors," *IEEE Transactions on Electromagnetic Compatibility*, 2018.

[MM2] M. Marek, P. Kadlec, and M. Čapek, "FOPS: A new framework for the optimization with variable number of dimensions," *International Journal of RF and Microwave Computer-Aided Engineering*, vol. 30, no. 9, p. e22335, 2020.

[MM3] M. Marek and P. Kadlec, "Another evolution of generalized differential evolution," *Engineering Optimization*, vol. 53, to be published, accepted on 6th November 2020.

[MM4] M. Marek, P. Kadlec, and M. Cupal, "FOPS Documentation." `http://antennatoolbox.com/download_file?file=FOPS_documentation.pdf`, 2017. [Online; accessed 24-November-2020].

[MM5] M. Marek and P. Kadlec, "Discretization of decision variables in optimization algorithms," in *Proceedings of the 24th Conference STUDENT EEICT 2018*, pp. 320–324, april 2018.

[MM6] M. Marek and P. Kadlec, "Using a tolerance-based surrogate method for computer resources saving in optimization," *Radioengineering*, vol. 28, no. 1, pp. 9–18, 2019.

[MM7] M. Marek, Z. Raida, and P. Kadlec, "Synthesis of electromagnetic equivalents of composite sheets by multi-objective optimization of anisotropic band-stop filters," in *2018 12th European Conference on Antennas and Propagation (EuCAP 2018) (EuCAP 2018)*, (London, United Kingdom (Great Britain)), apr 2018.

[MM8] D. Warmowska, M. Marek, and Z. Raida, "Matlab-based multi-objective optimization of broadband circularly polarized antennas," in *Proceedings of LAPC 2017*, pp. 1–5, october 2017.

[MM9] P. Kadlec, V. Šeděnka, M. Štumpf, and M. Marek, "Solution of an inverse scattering problem using optimization with a variable number of dimensions," in *2017 International Conference on Electromagnetics in Advanced Applications (ICEAA)*, pp. 976–979, IEEE, 2017.

[MM10] P. Kadlec, V. Šeděnka, M. Marek, and M. Štumpf, "Optimizing a decoupling capacitor on a PCB: A fully time-domain approach based on PSO and TD-CIM," in *2018 International Symposium on Electromagnetic Compatibility (EMC EUROPE)*, pp. 585–589, IEEE, 2018.

[MM11] P. Kadlec, M. Marek, and M. Štumpf, "Fast lightning stroke localization in the time domain," in *Proceedings of the 2019 International Symposium on Electromagnetic Compatibility (EMC Europe 2019)*, no. 1, (IEEE), pp. 54–58, IEEE, september 2019.

[MM12] P. Kadlec, M. Čapek, J. Rýmus, M. Marek, M. Štumpf, L. Jelínek, M. Mašek, and P. Kotalík, "Design of a linear antenna array: Variable number of dimensions approach," in *2020 30th International Conference Radioelektronika (RADIOELEKTRONIKA)*, pp. 1–6, IEEE, 2020.

[MM13] P. Kadlec, M. Marek, and M. Štumpf, "Lightning stroke localization–a time-domain approach based on evolutionary optimization," *IEEE Transactions on Electromagnetic Compatibility*, 2020.

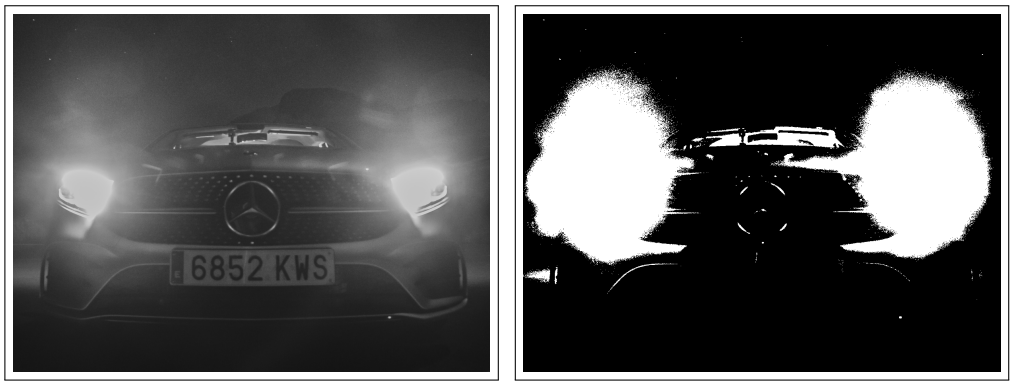
[MM14] P. Kadlec, M. Marek, and M. Štumpf, "Thin-film screen time-domain shielding effectiveness: Multi-objective optimization of the testing pulse," in *2020 International Symposium on Electromagnetic Compatibility-EMC EUROPE*, pp. 1–5, IEEE, 2020.

[MM15] M. Čapek, P. Hazdra, V. Adler, P. Kadlec, V. Šeděnka, M. Marek, M. Mašek, V. Losenický, M. Štrambach, M. Mazánek, and J. Rýmus, "AToM: A versatile MATLAB tool for antenna synthesis," in *2018 12th European Conference on Antennas and Propagation (EuCAP 2018) (EuCAP 2018)*, (London, United Kingdom (Great Britain)), apr 2018.

APPENDIX A

This appendix contains an extended library of images from thresholding problem for license-plate recognition. There are eight subfigures for each testing image noted from a) to h):

a) original testing image,
b) testing image segmented by one threshold,
c) testing image segmented by two thresholds,
d) the best of three layers where the license plate is most clear
e) testing image segmented by three thresholds,
f) the best of four layers where the license plate is most clear
g) testing image segmented by four thresholds,
h) the best of five layers where the license plate is most clear.

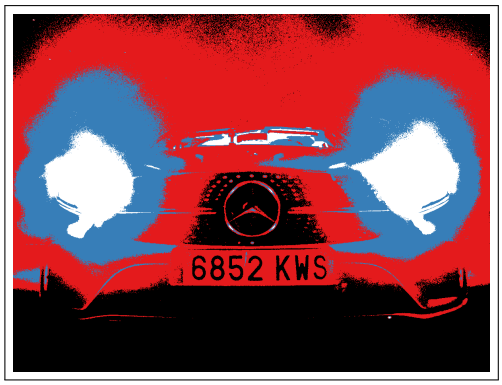


(a) Original greyscale image – AMG.

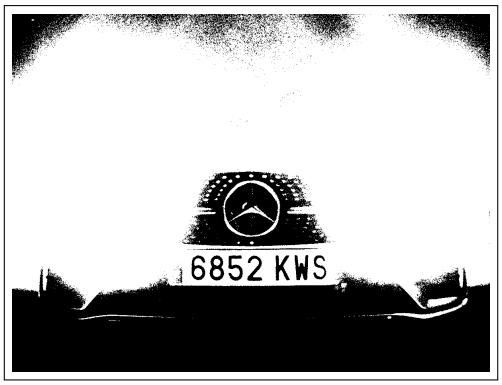
(b) AMG with one threshold.

(c) AMG with two thresholds.

(d) Two thresholds – first layer.



(e) AMG with three thresholds.



(f) Three thresholds – first layer.

(g) AMG with four thresholds.

(h) Four thresholds – first layer.

Figure 10.1: LPR testing image – AMG [110].

(a) Original greyscale image – BMW rear. **(b)** BMW rear with one threshold.

(c) BMW rear with two thresholds. **(d)** Two thresholds – second layer.

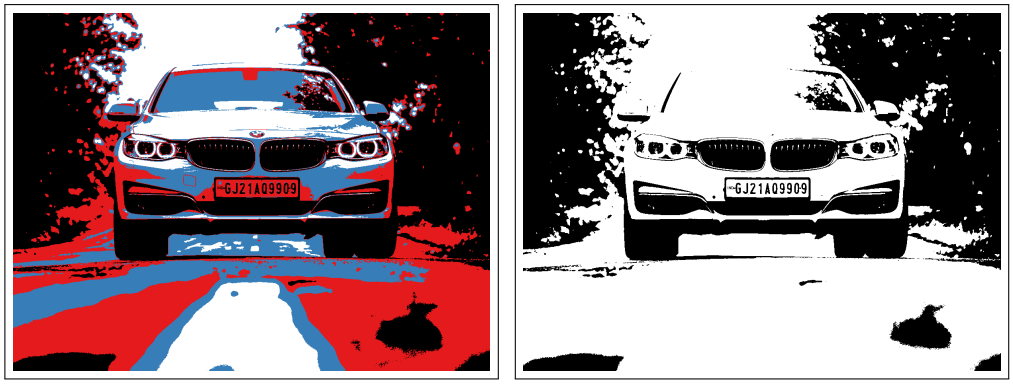
(e) BMW rear with three thresholds. **(f)** Three thresholds – third layer.

(g) BMW rear with four thresholds. **(h)** Four thresholds – fifth layer.

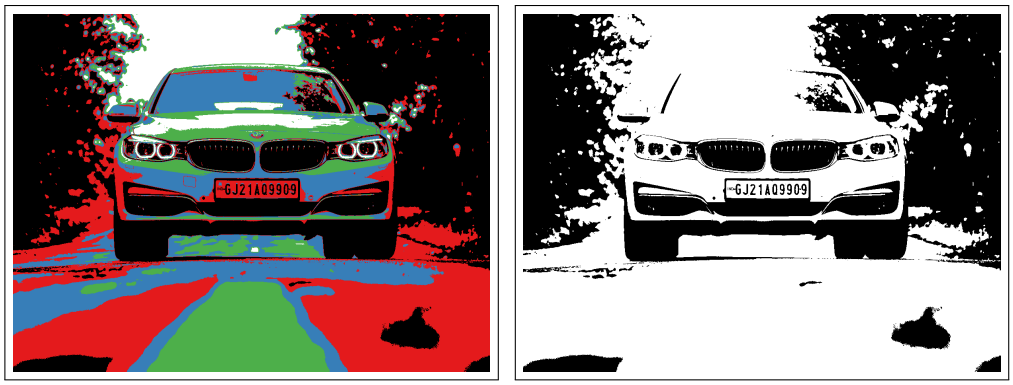
Figure 10.2: LPR testing image – BMW rear [111].

(a) Original greyscale image – BMW front. **(b)** BMW front with one threshold.

(c) BMW front with two thresholds. **(d)** Two thresholds – first layer.



(e) BMW front with three thresholds. **(f)** Three thresholds – first layer.



(g) BMW front with four thresholds. **(h)** Four thresholds – first layer.

Figure 10.3: LPR testing image – BMW front [112].

(a) Original greyscale image – Taxi.

(b) Taxi with one threshold.

(c) Taxi with two thresholds.

(d) Two thresholds – first layer.

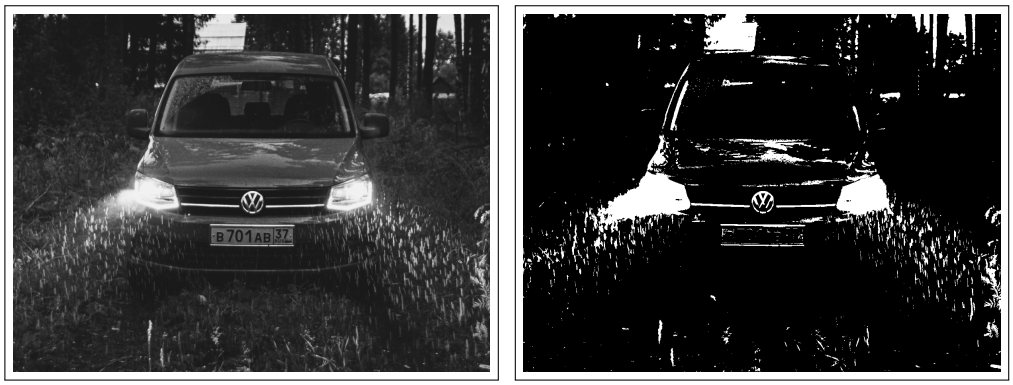
(e) Taxi with three thresholds.

(f) Three thresholds – first layer.

(g) Taxi with four thresholds.

(h) Four thresholds – first layer.

Figure 10.4: LPR testing image – Taxi [113].

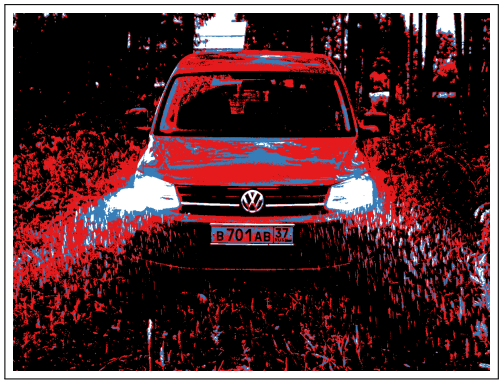


(a) Original greyscale image – Caddy.

(b) Caddy with one threshold.

(c) Caddy with two thresholds.

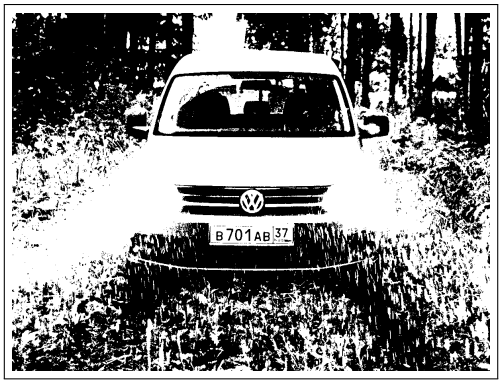
(d) Two thresholds – first layer.



(e) Caddy with three thresholds.

(f) Three thresholds – first layer.

(g) Caddy with four thresholds.



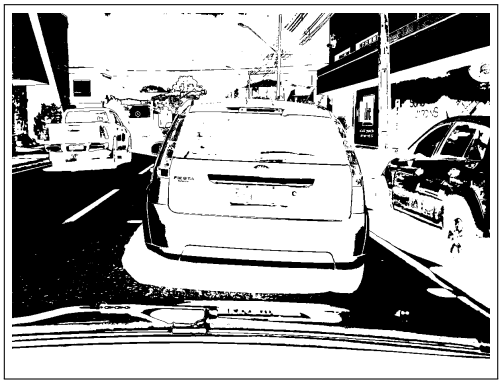
(h) Four thresholds – first layer.

Figure 10.5: LPR testing image – Caddy [131].

(a) Original greyscale image – Fiesta.

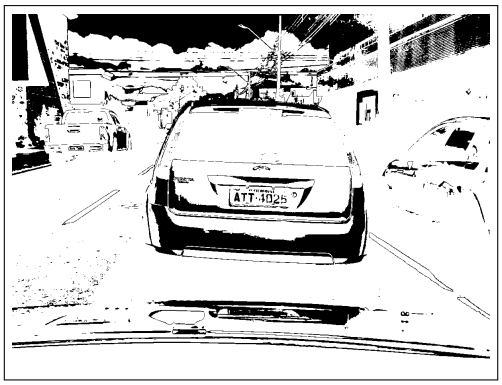
(b) Fiesta with one threshold.

(c) Fiesta with two thresholds.



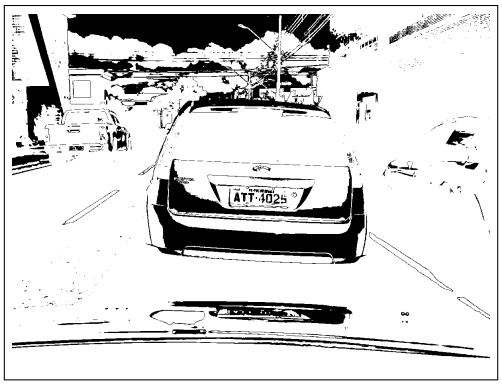
(d) Two thresholds – second layer.

(e) Fiesta with three thresholds.



(f) Three thresholds – third layer.

(g) Fiesta with four thresholds.



(h) Four thresholds – fourth layer.

Figure 10.6: LPR testing image – Fiesta [132].

(a) Original greyscale image – Polo.

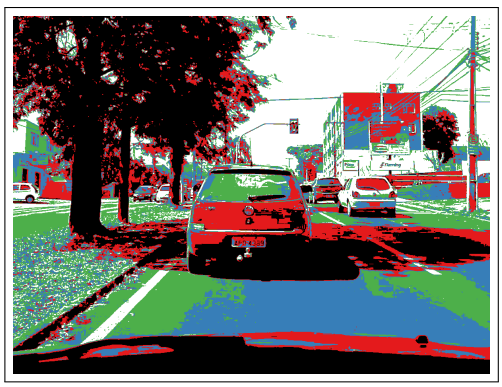
(b) Polo with one threshold.

(c) Polo with two thresholds.

(d) Two thresholds – first layer.

(e) Polo with three thresholds.

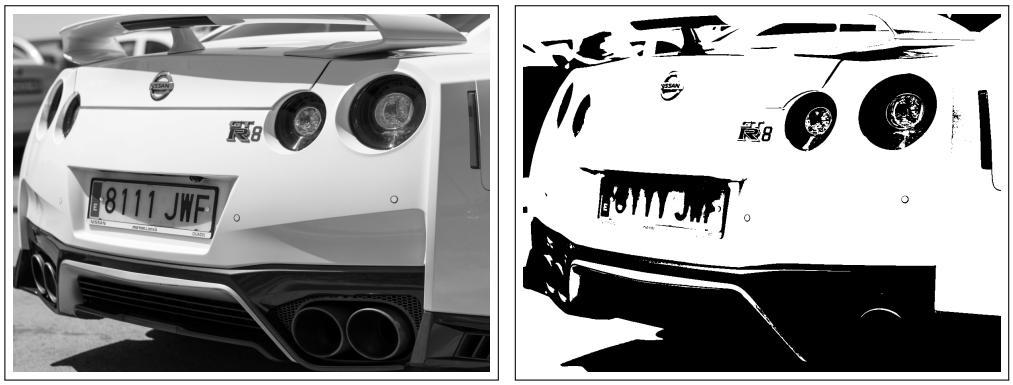
(f) Three thresholds – first layer.



(g) Polo with four thresholds.

(h) Four thresholds – first layer.

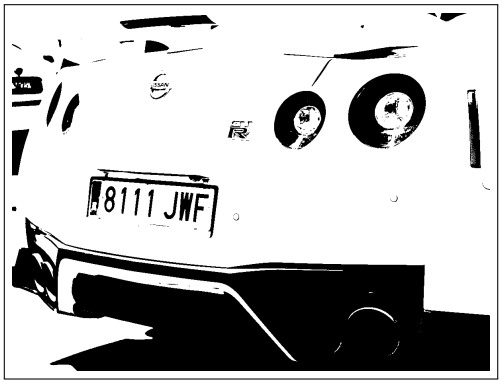
Figure 10.7: LPR testing image – Polo [132].



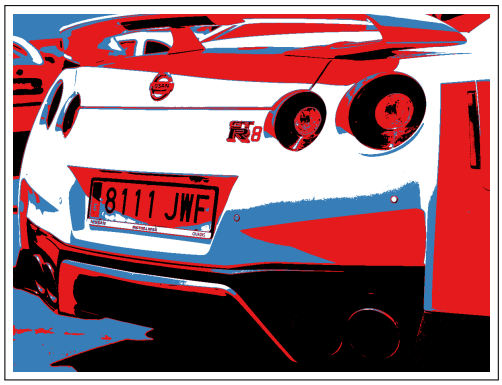
(a) Original greyscale image – GTR.

(b) GTR with one threshold.

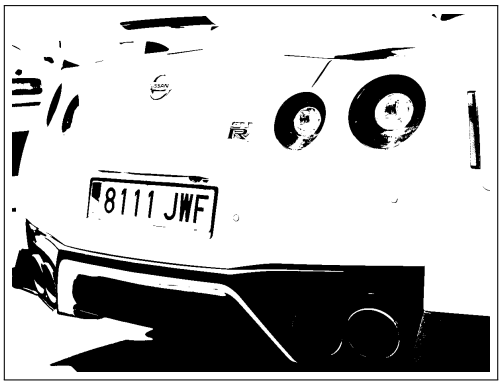
(c) GTR with two thresholds.



(d) Two thresholds – first layer.

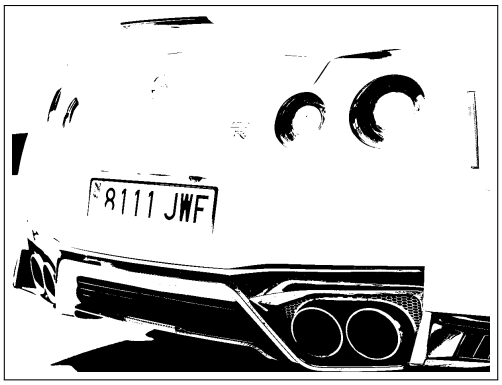


(e) GTR with three thresholds.



(f) Three thresholds – first layer.

(g) GTR with four thresholds.



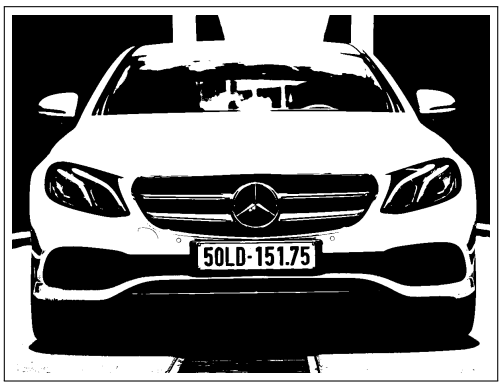
(h) Four thresholds – first layer.

Figure 10.8: LPR testing image – GTR [133].

(a) Original greyscale image – MB.

(b) MB with one threshold.

(c) MB with two thresholds.

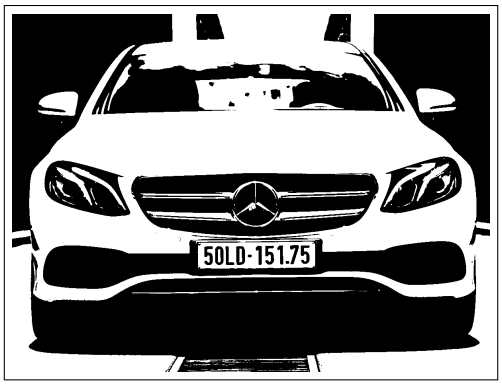


(d) Two thresholds – first layer.

(e) MB with three thresholds.

(f) Three thresholds – first layer.

(g) MB with four thresholds.



(h) Four thresholds – first layer.

Figure 10.9: LPR testing image – MB [134].

(a) Original greyscale image – Porsche. **(b)** Porsche with one threshold.

(c) Porsche with two thresholds. **(d)** Two thresholds – first layer.

(e) Porsche with three thresholds. **(f)** Three thresholds – first layer.

(g) Porsche with four thresholds. **(h)** Four thresholds – first layer.

Figure 10.10: LPR testing image – Porsche [135].