Aisha Azeez Younus

New Algorithms for Multi-objective Optimization

PhD dissertation

Supervisor:
Prof. Dr. hab. Marcin Studniarski

Faculty of Mathematics and Computer Science

University of Łódź

Łódź 2024

Author’s declaration:

I hereby declare that this dissertation is my own work.

……..

Aisha Younus

Supervisor’s declaration:

The dissertation is ready to be reviewed

……..

Prof. Dr. hab. Marcin Studniarski

- **Keywords:** Genetic algorithm, single-objective optimization, multi objective optimization, similarity of chromosomes, dissimilarity of chromosomes, Pareto, dynamic schema, Non-dominated Sorting, initial population.

ACM classification 2012

1- Computing methodologies - Machine learning - Machine learning approaches - Bio-inspired approaches - Genetic algorithms.
2- Computing methodologies - Artificial intelligence - Search methodologies - Discrete space search.

Abstract

In this work, five new multi-objective evolutionary algorithms are constructed and programmed by using Matlab. They are designed to search for a Pareto optimum of a multi-objective optimization problem. These algorithms are based on modified NSGA-II, hybrid algorithms, big population and multi-tasking. We use the DSC algorithm that is exploring similarities and dissimilarities between solutions (chromosomes represented as binary strings). Then, a special way to discover a schema of a binary string (the DSDSC algorithm) is used. Also, the effect of a big initial population is applied.

To prove the efficiency of these algorithms, fourteen test functions were used. We used nine test functions without constraint, and five test functions with constraint. One of them is with one variable, five functions with two variables, one function with three variables, one with four variables, one function with six variables, two functions with ten variables, and three functions with 30 variables. The results showed, in most cases, the superiority of the algorithms proposed in this thesis in run time.

This thesis consists of six chapters. Chapter one is a general introduction to optimization. Chapter two contains a literature review and the discussion of performance measures for multi-objective optimization algorithms. In the third chapter, an algorithm called NSDSC is described, which combines using the dissimilarity operator and the similarity operator with random generation of a part of each new population, and building Pareto fronts. Also, a modification of NSDSC, called NS-DSDSC, is presented. The fourth chapter introduces two new algorithms,

first Hybrid NSDSC with NSGA-II algorithm, second the First Big population Hybrid NSDSC with NSGA-II algorithm. The fifth chapter contains evolutionary multi-tasking algorithm, finally, chapter six contains some conclusions of this work.

Streszczenie

W tej pracy skonstruowano i zaprogramowano pięć nowych wielokryterialnych algorytmów ewolucyjnych przy użyciu Matlaba. Zostały one zaprojektowane w celu poszukiwania optimum Pareto wielokryterialnego problemu optymalizacji. Algorytmy te są oparte na zmodyfikowanym NSGA-II, algorytmach hybrydowych, dużej populacji i wielozadaniowości. Używamy algorytmu DSC, który bada podobieństwa i różnice między rozwiązaniami (chromosomy reprezentowane jako ciągi binarne). Następnie stosuje się specjalny sposób odkrywania schematu ciągu binarnego (algorytm DSDSC). Zastosowano również efekt dużej początkowej populacji. Aby udowodnić wydajność tych algorytmów, użyto czternastu funkcji testowych. Użyliśmy dziewięciu funkcji testowych bez ograniczeń i pięciu funkcji testowych z ograniczeniami. Jedna z nich jest z jedną zmienną, pięć funkcji z dwiema zmiennymi, jedna funkcja z trzema zmiennymi, jedna z czterema zmiennymi, jedna funkcja z sześcioma zmiennymi, dwie funkcje z dziesięcioma zmiennymi i trzy funkcje z 30 zmiennymi. Wyniki wykazały, w większości przypadków, wyższość algorytmów zaproponowanych w tej rozprawie jeśli chodzi o czas wykonywania.

Rozprawa składa się z sześciu rozdziałów. Rozdział pierwszy jest ogólnym wprowadzeniem do optymalizacji. Rozdział drugi zawiera przegląd literatury i dyskusję na temat miar wydajności dla wielokryterialnych algorytmów optymalizacji. W rozdziale trzecim opisano algorytm o nazwie NSDSC, który łączy użycie operatora odmienności i operatora podobieństwa z losowym generowaniem części każdej nowej populacji i budowaniem frontów Pareto. Przedstawiono również modyfikację NSDSC o nazwie NS-DSDSC. Czwarty rozdział wprowadza dwa nowe algorytmy, pierwszy Hybrid NSDSC z algorytmem NSGA-II, drugi First Big population Hybrid NSDSC z algorytmem NSGA-II. Piąty rozdział zawiera ewolucyjny algorytm wykorzystujący wielozadaniowość, a rozdział szósty zawiera pewne wnioski z tej pracy.

ACKNOWLEDGEMENTS

First of all, I am grateful to my country, Iraq, I appreciate The Republic of Poland and all staff of the University of Lodz, especially my supervisor Prof. Dr. hab. Marcin Studniarski for his valuable insights and wise guidance during the study period, because he showed me valuable guidance in writing the doctoral dissertation and published papers.

I would like to thank my Husband Radhwan Al-jawadi for his support during the PhD study, as well as also my thanks to my children for their patience with me and their patience of difficulties of alienation.

Eventually, my sincere gratitude goes to my beloved parents (God's mercy on them), my sister and my brothers, for their patience and encouragement throughout the study period. I want to thank all those who helped me out even by a word. I would like to thank all my friends here in Poland and in Iraq.

Aisha Younus

Abbreviations:

ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
BBO	Biogeography Based Optimization
BFO	Bacterial Foraging Optimization
CD	Crowding Distance
CMOPs	Constrained Multi-objective Optimization Problems
CNS	Constrained Non-dominated Sorting
DCD	Dynamic Crowding Distance
DE	Differential Evolution
DHCS	Distributed Heterogeneous Computing Systems
DPEA	Distance-based Pareto Evolutionary Algorithm
DPGA	Distance-based Pareto GA
DSC	Dissimilarity and Similarity of Chromosomes
DSDSC	Dynamic Schema with Dissimilarity and Similarity of Chromosomes
EAs	Evolutionary Algorithms
EMCMO	Evolutionary Multitasking-based Constrained Multi-objective Optimization
EMO	Evolutionary Multi-objective Optimization
EO	Evolutionary Optimization
GA	Genetic Algorithm
GD	Generational Distance
HABC-DE	Hybrid Artificial Bee Colony and Differential Evolution
HBBABC	Hybrid Biogeography Based and Artificial Bee Colony
HMCP	Hub Maximal Covering Problem
HSCP	Hub Set Covering Problem
HV	Hyper Volume

IGD	Inverted Generational Distance
MID	Mean ideal distance
MOGA	Multi-Objective GA
NP	Nested Partitions
NPGA	Niched Pareto GA
NSDSC	Non-dominated Sorting with Dissimilarity and Similarity of Chromosomes
NS-DSDSC	Non-dominated Sorting with Dynamic Schema Dissimilarity and Similarity of Chromosomes
NSGA	Non-dominated Sorting Genetic Algorithm
NSPSO	Non-dominated Sorting Particle Swarm Optimization
NTVPSO	Nonlinear Time-Varying PSO
PAES	Pareto-Archived Evolution Strategy
PFA	Pareto-Front-Arithmetic
POF	Pareto Optimal Front
PSO	Particle Swarm Optimization
QM	Quality metric
SDVRP	Split Delivery Vehicle Routing Problem
SGA	Simple Genetic Algorithm
SM	Spacing metric
SOO	Single Objective Optimization
SPEA	Strength Pareto Evolutionary Algorithm
TDGA	Thermo Dynamical GA
TSP	Traveling Salesman Problem
VEGA	Vector-Evaluated GA
VOEA	Vector Optimized EA
WBGA	Weight Based GA
WS	Weighted-Sum

WSN	Wireless Sensor Networks
MFEA	Multi Factorial Evolutionary Algorithm
MFO	Multi Factorial Optimization
QIs	Quality Indicators

Contents

List of Tables

List of Figures

List of Algorithms

CHAPTER ONE: Introduction

1.1 Overview

In this chapter we provide an overview of the main optimization methods for Multi-Objective Optimization (MOO). We mention some classical methods, and focus on Non-dominated Sorting Genetic Algorithms (NSGA), also principles of MOO and the scalarization method.

1.2 Principles of Optimization

In any problem involving decision making, be it in engineering or in economics, optimization plays a crucial role. The task of decision making entails choosing between various alternatives. Our desire to make the "best" decision stands behind the choice. The goodness of the alternatives is measured by an objective function or performance index.

Optimization theory and techniques deal with selecting the best alternative in the sense of a given objective function. The area of optimization has received enormous attention in recent years, primarily because of the rapid progress in computer technology, including the development and availability of user-friendly software, high-speed and parallel processors, and artificial neural networks [1].

Search algorithms, also known as optimization approaches, are one possible way to assist a decision maker in making a good solution choice. Real-time applications can benefit from the use of optimization algorithms [2].

Optimization is a task of searching for a set of decision variables which would minimize or maximize objective function subjected to satisfying constraints [3].

There is no universal method, but a set of tools which requires a lot of experience to be used properly [4].

1.3 Classification of Optimization Problems

There are many ways to classify optimization problems, and different classification schemes are appropriate for different purposes. Some common classification schemes are the following.

Linear vs nonlinear optimization: This classification is based on the type of constraints and objective function that the optimization problem has. Linear optimization problems have linear constraints and a linear objective function, while nonlinear optimization problems may have nonlinear constraints and/or a nonlinear objective function [5]**,** [6], [7].

Convex vs nonconvex optimization: This classification is based on the geometry of the objective function and constraints. Convex optimization problems have an objective function and constraints that are both convex, which means that they have a "bowl" shaped geometry. Nonconvex optimization problems may have a more complicated geometry [8], [9].

Continuous vs discrete optimization: This classification is based on the type of decision variables that the optimization problem has. Continuous optimization problems have decision variables that can take on any value in a continuous range, while discrete optimization problems have decision variables that can only take on certain discrete values [9], [10].

Single-objective vs multi-objective optimization: This classification is based on the number of objective functions that the optimization problem has. Single-objective optimization problems have a single objective function to be minimized or maximized, while multi-objective optimization problems have multiple objective functions that must be balanced in some way [11], [12], [13].

Deterministic vs stochastic optimization: This classification is based on the presence or absence of randomness in the optimization problem. Deterministic optimization problems have no randomness, while stochastic optimization problems have some element of randomness that must be taken into account in the optimization process [14].

Static vs Dynamic Optimization Problems. Static Optimization Problem: This kind optimization happens when variables don’t depend on each other, Dynamic Optimization Problem: This happens when some design variables depend on some other design variables [15].

1.4 Introduction to Multi-Objective Optimization

As the name suggests, multi-objective optimization involves optimizing a number of objectives simultaneously. The problem becomes challenging when the objectives are in conflict with each other, that is, the optimal solution of one objective function is different from that of the other. In solving such problems, with or without the presence of constraints, these problems give rise to a set of trade-off-optimal solutions, popularly known as Pareto-optimal solutions. Due to the multiplicity in solutions, these problems were proposed to be solved suitably using Evolutionary Multi-Objective Optimization algorithms (EMO) which use a population approach in its search procedure [16].

Modern optimization has seen a tremendous increase in the field of multi-objective optimization. There are numerous techniques and algorithms available to solve multi-objective optimization issues. The techniques fall into two types:

(i) Traditional approaches, which rely on direct or gradient-based techniques and follow some mathematical principles, Classical methods mostly attempt to scalarize multiple objectives and perform repeated applications (multiple runs) to find a set of Pareto-optimal solutions.

(ii) Non-traditional approaches that follow some natural or physical principles, EMO methods attempt to find multiple Pareto-optimal solutions in a single simulation run [17].

1.4.1 Some of EMO and Non-elitist Methodologies

Since the beginning of the 1990s, multi-objective optimization (MOO) approaches have adequately demonstrated their expertise in identifying a collection of well-converged and well-diversified non-dominated solutions to various two- and three-objective optimization problems. However, there are a lot of optimization problems with four or more objectives in real-world applications [18], [19].

The Evolutionary Algorithms (EA) researchers realized that they had to solve multi-objective optimization problems. In 1967, Rosenberg proposed a Weighted-Sum (WS) approach to convert multiple objectives into a single goal [20], [21].

The Weighted-Sum method is called generated method, because we generate a point every time, also it applies a single objective method many times and not every Pareto optimal point can be found by this method, because of inability to find some Pareto optimal solutions, those in non-convex region, also some difficulties in the WS method arise because one needs to know weights of points. However, a solution generated by this approach is Pareto-optimal.

The most famous method for solving MOO problems is Vector-Evaluated GA (VEGA), that was suggested by David Schaffer in 1984, the VEGA is a very simple method, so suppose we have two objectives, the method divides population into two halves, the first half evaluates f_1, the second half evaluates f_2, and then recombination of the whole population takes place [20], [22].

In 1989, Goldberg's suggestion was to use the concept of dominance to allocate more copies to the non-dominated individuals in population, since diversity is another concern, it has also been proposed to use a niching strategy, where niching operator controls the selection pressure on population members to prevent the population from being dominated by a single solution [20]. To implement this idea, at least three independent groups of researchers developed different versions of multi-objective evolutionary algorithms during 1993-1994 [20].

In 1993, Fonseca and Fleming suggested a Multi-Objective GA (MOGA), in which all non-dominated population members are assigned a rank one, other individuals are evaluated by counting the number of solutions (denoted by k) that dominate a given solution, then a rank (k+1) is assigned to this solution. The selection procedure then selects the lowest-rank solutions to form the mating pool [20], [23].

There are many studies that talk about of EMO and Non-elitist Techniques: Vector Optimized EA (VOEA) [24], [5], Weight Based GA (WBGA) [21], Multiple Objective GA (MOGA) [25], [26], Non-Dominated Sorting GA (NSGA) [27], Niched Pareto GA (NPGA) [28] and Predator-Prey ES [29].

1.4.1.1 Difficulties with Non-Elitist Approaches

Non-elitist strategies have the following three defects: (1) the Pareto-optimal individuals that have been found are not preserved over time, (2) they have difficulty to maintain variety on the Pareto frontier, (3) the convergence of solutions towards the Pareto frontier is slow [30].

New approaches have been used to overcome the aforementioned challenges, such as the application of niching, clustering, and grid-based techniques to efficiently distribute solutions on the Pareto front [30], [31].

1.4. 2 Some of EMO and Elitist Methodologies

Non-elitist EMO methodologies mentioned above, give a good head start to the research and application of EMO, but they have a common problem that they don't have an elite-preserving mechanism [20].

However, as the second generation of algorithms, plenty of EMO methods were proposed which implemented elite-preserving.

In 1995, Osychka and Kundu suggested a Distance-based Pareto GA as new GA-based multicriteria optimization method [32], while in 2016 the authors of [33] proposed Distance-based Pareto Evolutionary Algorithm (DPEA), as a new algorithm, which strongly depends on the ideas behind Strength Pareto Evolutionary Algorithm (SPEA and SPEA2) [34]. SPEA and SPEA2 are not able to handle infeasible solutions satisfactorily, while DPEA keeps the infeasible solutions in population, so that the evolutionary process will not be delayed [33].

In 1996 Kite et al. have presented Thermo Dynamical GA (TDGA) [35], a GA using the concepts of the entropy and the temperature in the selection operation, for multi-objective optimization.

In 1999, the author of [36], proposed the Strength Pareto EA (SPEA), using an external population (an archive) where all non-dominated solutions found are stored. The archive updates in each iteration, all solutions from the archive are non-dominated [25], [36], [37].

In 2002, Deb presented “A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II” [38], also he presented this algorithm in 2001 in the book [39], this NSGA-II was faster than

NSGA in time complexity of computations: $O(M N^2)$ where NSGA was $O(M N^3)$. Moreover, NSGA-II has a principle of elitism where NSGA does not have it. The NSGA-II procedure is one of the popularly used EMO procedures which attempts to find multiple Pareto-optimal solutions in a multi-objective optimization problem and has the following three features:

1) it uses an elitist principle,

2) it uses an explicit diversity preserving mechanism, and

3) it emphasizes non-dominated solutions [40], [16].

The first and second features of NSGA-II are included in our previous optimization algorithm "A New Genetic Algorithm Based on Dissimilarities and Similarities, 2017" [41], for Single Objective Optimization (SOO), therefore we have decided to use it combined with NSGA-II. But instead of using GA operators for optimization algorithm, we use Dissimilarities and Similarities of Chromosomes operators (DSC).

In 1999, Knowles and Corne presented a Pareto-Archived ES (PAES) [42], the idea was to maintain a finite-sized archive of non-dominated solutions, it was used instead of a local search tool, the global search procedure based on crossover operators was applied to the set of candidate solutions, which undergo a local search and replace their origin in Lamarck's spirit. Where, the Lamarckian spirit of Baldwinian criteria can be used to incorporate learning into an EA. According to the Lamarckian philosophy, an individual genetic structure is affected by or reflects their learning outcomes. In actuality, this back-coding means that the population has better genotype and fitness values replace the initial ones, making every successful behavior inheritable [43].

1.5 The Concept of Domination

The goal of multi-objective optimization is to locate nondominated solutions, and the domination idea offers a useful tool to compare solutions for multi-objective optimization [44].

The MOO problem using can be written as follows [54]:

$$min/\,max\ f_1(x), f_2(x), \ldots . f_m(x) \qquad (1.1)$$

$$\text{subject to} : x \in \Omega$$

where x is a solution, m is the number of objective functions, Ω is a feasible set, $f_i(x)$ is i-th objective function, and min/max means that the functions are minimized or maximized simultaneously .

A solution is said to *dominate* another solution if it is better or equal in all objectives and strictly better in at least one objective [45], as the following mathematical formula shows.

Solution x_1 dominates x_2 (denoted by $x_1 \prec x_2$) if:

$f_i(x_1) \leq f_i(x_2)$ for all $i \in \{1,2,..,m\}$ and $f_i(x_1) < f_i(x_2)$ for at least one i, where m is the number of objective functions [22].

A solution $x^*$is called a *Pareto optimal solution* if there is no other solution $s \in \Omega$ such that $s \prec x^*$. A Pareto optimal solution is also called a *nondominated solution* [29] or an *efficient solution*. The set of all the Pareto solutions in Ω is called the *Pareto set* and its image in the objective space (see Figure 1.1) is called the *Pareto front*.

The Pareto front represents the trade-offs between the different objectives, and any solution that is not on the Pareto front can be improved in at least one objective without making any of the other objectives worse [46].

The concept of an efficient solution for multiobjective optimization is not simple. It has a close connection to the decision-makers' attitudes and preferences. The idea is that an efficient solution (nondominated, or noninferior) must respect the domination structure of the decision maker, to help the decision maker in tradeoff analysis, the multiobjective optimization must select an acceptable solution [47].

1.5.1 What is Pareto front? [20]

In multi-objective optimization, a Pareto front is the image of the Pareto optimal set under the objective functions that represents the trade-offs between different objectives, where no solution can be improved in one objective without degrading another objective [48], [49].

In other words, a point on the Pareto front is the best possible solution given the trade-offs between the different objectives. The Pareto front is named after Vilfredo Pareto, an Italian economist who first described the concept of Pareto efficiency.

The concept of Pareto front can be visualized in a Pareto chart, which is a scatter plot of the solutions in the objective space. Each point on the chart represents a different solution, and the Pareto front is the set of points that are not dominated by any other point.

In a Pareto optimization problem, the goal is to find the Pareto front, or a good approximation of it. There are various methods to find Pareto front like evolutionary algorithms and gradient descent.

Pareto front is used in different fields like engineering, finance, game theory and so on, when we have multiple objectives that are in conflict with each other and we need to find trade-offs between them [20].

In practice, when working with multi-objective optimization problems, the Pareto front is often approximated using a set of solutions obtained through numerical methods [30], [50].

The true Pareto front is not always known, and the approximate Pareto front can be used instead. The approximate Pareto front can be used as a benchmark to evaluate the performance of different optimization algorithms and to compare different solutions to a problem [51], [52].

1.6 Methods of MOO

The solution methods of MOO problems can essentially be divided into two groups: (1) the Pareto methods and (2) the scalarization methods [53]. The Pareto and scalarization methods are different. In the Pareto methods the performance indicators are treated separately. These methods produce a set of compromise solutions (trade-offs) that can be displayed in the form of Pareto Optimal Front (POF). On the other hand, the scalarization methods use a performance indicators component that forms a scalar function which is incorporated in the fitness function [54], [8]. The following section will explain both methods.

1.6.1 Pareto Method

In the MOO, there is a multi-variable space of the objective function vectors and the decision variable space of the solution vectors. For every solution x in the decision variable space there is a corresponding point in the objective function space [39].

The idea of dominance is used to distinguish between dominated and non-dominated solutions, and the Pareto technique maintains the solution vectors independent of one another throughout optimization. The dominating solutions and best values on the Pareto front in MOO are then obtained to the minimization or maximization problem. This scenario is known as the Pareto optimization . The mapping between the solution vector and the objective function vector can be seen in Figure 1.1 [54].

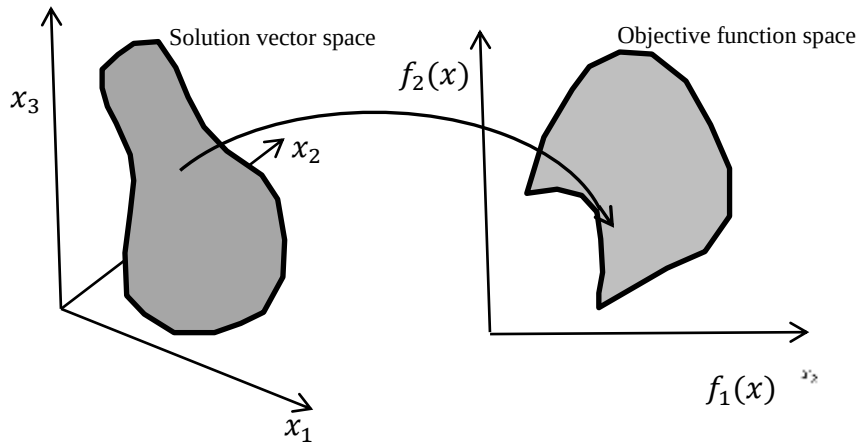

Figure 1.1 Mapping solution space with objective space [54].

1.6.2 Scalarization Method

The scalarization method makes the multi-objective function create a single solution and the weight is determined before the optimization process. The scalarization method incorporates multi-objective functions into one scalar fitness function.

$$F(x) = W_1 f_1(x) + W_2 f_2(x) + \cdots + W_n f_n(x) \qquad (1.2)$$

1.7 Diversity and Convergence

One of the objectives of MOO is to find solutions that are as close to the Pareto-optimal front as possible, and the other is to find solutions that are as diverse as possible within the achieved non-dominated front. These two objectives are at odds with one another in several

respects. Referring to Figure 1.2, the first aim needs searching towards the Pareto-optimal region, whereas the second goal needs searching along the Pareto-optimal front [55].

In our work we have used these algorithms: DSC [41], [12] and DSDSC [56], because they generate random solutions in each iteration [12].

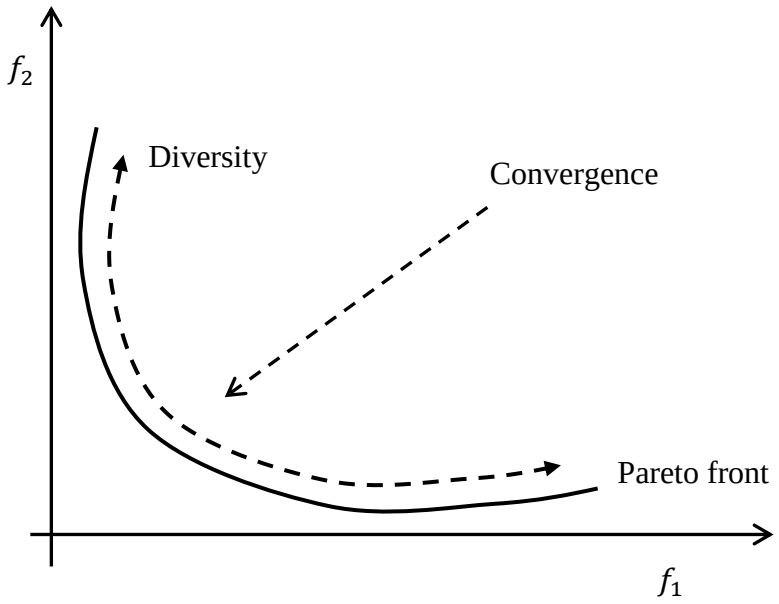

Figure 1. 2 The aspect of MOEA performance metrics: diversity and convergence [55]

1.8 Advanced Optimization Techniques

The Nested Partitions NP technique, another metaheuristic for combinatorial optimization that is easily applied to simulation optimization issues, is introduced in [57]. The primary concept behind this approach is to carefully divide the feasible region into sub-regions, assess each region's potential, and then concentrate the computing effort on the most promising region. Each partition in this procedure is nested within the last partition in an iterative way [57], [12]. The NP method's computing efficiency primarily depends on the partitioning, which, if done in a way that groups fitting solutions together, can quickly arrive at a near-optimal solution [58].

Deterministic and metaheuristic algorithms can be used to perform global optimization; for further information, see [59]. For a vast class of optimization problems where deterministic algorithms are not appropriate, metaheuristic approaches are helpful (for example, functions with a large number of local extrema).

Numerous function evaluations are needed for metaheuristics. They are often characterized as population-based stochastic search routines, that ensure a high possibility of escaping the local optimal solutions when compared to gradient-based and direct search algorithms [60].

The following are the population-based metaheuristics' techniques [15], [61]:

1. Evolutionary computation, including genetic algorithms, evolutionary programming, and evolutionary strategies.
2. Swarm intelligence: artificial immune systems, Bacterial Foraging Optimization (BFO), Artificial Bee Colony (ABC), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), and biogeography-based optimization.
3. Evolutionary Algorithms (EA), including scatter search, path relinking, Differential Evolution (DE), coevolutionary algorithms, and cultural algorithms.

1.9 Single Objective Optimization

A Single-Objective Optimization (SOO) problem's objective function could have many global optimal points. For instance, there are 18 optimum solutions to the Shubert problem, but any of the solutions that the algorithm finds will satisfy the decision maker. Single-objective global optimization is the name given to this kind of optimization [12].

For instance, in the Traveling Salesman Problem (TSP), the objective is to determine the shortest route that allows us to visit each city just once before returning to the starting point. The goal of this task is to reduce the tour's duration [39], [62].

1.10 Genetic Algorithm

The heuristic search methods known as Genetic Algorithms (GA) are based on the process of natural evolution. They have found applications in generating useful solutions for

problems involving optimization and search. The foundation of GA is natural selection modeling, which does not require the computation of any secondary functions like derivatives. The following are some benefits of GA that make it more effective in solving optimization problems:

(a) The likelihood of local minimum trapping is reduced.

(b) Moving from one point to another requires less computational effort until reaching to the solution.

(c) Evaluation of the fitness of each string guides the search.

One advantage of applying GA techniques is that they typically result in globally optimal solutions [2], [12].

In 1960s, "Evolutionary computing" was introduced by I. Rechenberg in his work “Evolution strategies”, and was further developed by other researchers. Genetic Algorithms (GAs) were discovered by John Holland who suggested this idea in his book “Adaptation in natural and artificial systems” in 1975 [63]. Holland suggested GA as a heuristic method based on “survival of the fittest”. GA proved to be a useful tool for search and optimization problems [64].

Genetic algorithms have long been used to solve problems. J. H. Holland's innovative work in the 1970s made a substantial contribution to applications in science and engineering [65].

1.10.1 Genetic Operators

In a typical GA, there are typically three operators [44]. The first is the selection operator, which creates a single copy or multiple copies of individuals of the population. Individuals that are physically fit are more likely to be picked; otherwise, they are removed from the solution pool. The next operator is recombination operator, also referred to as the "crossover" operator, where from the generation two individuals are chosen for a crossover. There are two complimentary uses for the crossover operator. First, it provides new points for farther testing within the hyperplanes already existent in the population [12], [66]. Furthermore, crossover adds members of new hyperplanes to the population that neither of the parent structures had previously included. As a result, there is a far higher chance of having offspring who perform better. "Mutation" is the name of the third operator. By randomly flipping a bit in a population of strings, this operator serves as a background operator and is utilized to explore some of the

previously unexplored points in the search space. Because frequent usage of this operator would result in a totally random search, it typically has a very low likelihood of activation [66].

A genetic search begins with an initial population that is created at random and in which each member is examined using a fitness function. Individuals in this generation and the ones after it are replicated or removed through selection based on how fit they are. Applying GA operators results in subsequent generations. This procedure is planned to produce a generation of high-achieving individuals [66].

1.11 Thesis Contributions and Overview

This thesis contributes to give new multi-objective optimization algorithms, where the main idea is to use new optimization algorithms (DSC and DSDSC) in multi-objective optimization instead of GA, and comparing with NSGA-II.

Then we use the other two new hybrid multi-objective optimization algorithms that combine Non-dominated Sorting Dissimilarity and Similarity of Chromosomes (NSDSC) with NSGA-II, also apply a big population on a hybrid algorithm, Then we compare the results with NSGA-II.

Finally, a new multi-objective algorithm using multi-tasking with NSGA-II is proposed.

1.12 Computing running time

Briefly, the run time was computed and compared between our new algorithms with NSGA-II on 14 tested multi-objective functions, our algorithms have shown superiority over NSGA-II, except for some functions, for which our algorithms have not found a solution, especially for the problems that have involved constraints.

1.13 Structure of the Thesis

The thesis is organized as follows:

Chapter 2: The literature review of multi-objective optimization approaches, NSGA-II, and initial population effects.

Chapter 3: Presentation of two new multi-objective algorithms: NSDSC and NS-DSDSC.

Chapter 4: Presentation of two new hybrid multi-objective algorithms: NSDSC with NSGA-II and the application of a big population on NSDSC with NSGA-II.

Chapter 5: A new multi-objective algorithm using multi-tasking with NSGA-II.

Chapter 6: Conclusions.

Appendix A: Presentation of all test functions.

CHAPTER TWO: Literature Review

2.1 Introduction

Numerous methods based on Multi-Objective Optimization (MOO) techniques have been put forth in recent years to process MOO problems.

In this literature review, we focus on the following topics: NSGA-II, Non-dominated Sorting Particle Swarm Optimization (NSPSO), crowded distance, multi-objective hybrid algorithm, and initial population.

2.2 Literature review of NSGA

The authors of [38] present "A Fast and Elitist Multiobjective Genetic Algorithm (NSGA-II)", where they mention the main criticisms of the NSGA approach have been as follows:

1) Nondominated sorting has a high computational complexity, with the currently employed approach having an $O(MN^3)$ complexity (where M is the number of objectives and N is the population size). For large population sizes, this renders NSGA computationally expensive. This significant complexity results from the difficulty of the nondominated sorting process in each generation.

2) Lack of elitism: Recent findings indicate that elitism can greatly speed up the GA's performance, which also can assist to prevent the loss of good solutions once they are discovered.

3) Specification of the sharing parameter is required: The idea of sharing has been the mainstay of traditional systems for ensuring diversity in a population in order to produce a wide variety of similar solutions. The biggest issue with sharing is that it needs a sharing parameter (σ_{share}) to be specified. Despite some efforts to dynamically size the sharing parameter [24] [14], it would be preferable to have a parameter-free diversity-preservation system.

The authors of [38] have found that NSGA-II outperforms two other contemporary MOEAs: Pareto-Archived Evolution Strategy (PAES) and strength-Pareto EA (SPEA) in terms of finding a diverse set of solutions and in converging near the true Pareto-optimal set.

In [67] the authors have used NSGA-II and Non-dominated Sorting Particle Swarm Optimization Algorithm (NSPSO) for efficient utilization of resources and execution of the tasks, these algorithms were implemented intending to schedule independent tasks in a Distributed Heterogeneous Computing Systems (DHCS) environment of optimizing makespan and flowtime simultaneously, where "makespan and flowtime refers to the response time in execution of the user's request" [67], from the results obtained, it is seen that NSGA-II provides a set of good quality solutions that offer more flexibility to users to estimate their preferences and choose a desired schedule.

The results also verify that genetic operators used in NSGA-II help to evolve better solutions rather than controlling the movement of the particles in NSPSO for evolving better solutions. Though PSO is simpler to implement it is found that NSGA-II works well in the case of multi-objective problems. A sample of their work of [67] is shown in the Figure 2.1.

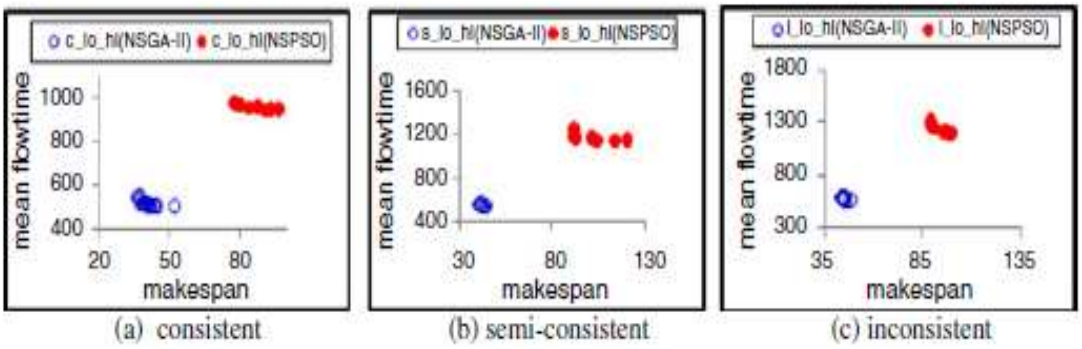

Figure 2. 1 Comparison NSGA-II and NSPSO [67]

2.3 Hybrid algorithms

2. 3. 1 Hybrid algorithms for single-objective optimization

A hybrid evolutionary algorithm made up of Genetic Algorithm (GA), heuristics, and Ant Colony Optimization (ACO) has been presented by G. P. Rajappa in [68]. The Split Delivery Vehicle Routing Problem (SDVRP) was tested and a solution was proposed. In a brief, the hybrid algorithm creates and evaluates a large initial population (1000) using ACO, then selects 500 of the best routes and inserts them into the modified genetic algorithm to create an initial generation. For the datasets included in the author's analysis, the hybrid GA demonstrates the ability to deliver better outcomes while requiring less computing time.

In Chapter 4 of our thesis, we use a similar concept of a large initial population. The initial large population is evaluated by NSDSC; then, the best solutions are taken from it and inserted as the first generation to the NSGA-II algorithm. For each function that is tested, the initial population starts with 1000 elements, then we take best 200 of them.

According to the authors of [69], the global path planning is a problem in the area of mobile robots and it is difficult due to its complexity and nature as a nondeterministic polynomial-time hard problem (NP-hard). They proposed a novel hybrid optimization technique to address this issue by first inventing the PSO and DE algorithms, then combining them. The hybrid algorithm's evolved PSO, known as Nonlinear Time-Varying PSO (NTVPSO), updates the positions of the particles' velocities in an effort to prevent stagnation.

A Hybrid Biogeography Based and Artificial Bee Colony algorithm (HBBABC) is created in [70] by combining the two well-known algorithms: Biogeography Based Optimization (BBO) and Artificial Bee Colony (ABC). It makes use of the ABC exploration and BBO exploitation functionalities. The performance of this hybrid approach, which takes into consideration discrete design variables and five engineering design optimization issues, was tested on 14 benchmark problems. The Mean Solution, Best Solution, T-test, Success Rate, and other criteria are also taken into consideration.

Using the same criterion as above, experimental results show that HBBABC performs better overall than BBO and ABC.

The author of [71] suggests a new real-coded evolutionary algorithm to use for a four-bar linkage's path synthesis. Differential Evolution (DE) and Real-valued Genetic Algorithm (RGA) are combined in the suggested novel evolutionary algorithm. The "GA-DE hybrid algorithm" is the name of this hybrid algorithm. The author replaces the crossover operation in the RGA with differential vector perturbation, using the best individual or a few good individuals as the base vectors. This is the sole difference between the proposed technique and RGA. Four cases were used to evaluate the procedure, and the results showed that three of the cases had more precise solutions than those found using previous evolutionary techniques.

2. 3. 2 Hybrid algorithms for multi-objective optimization

In [72] a Hybrid Algorithm combining the multi-objective Artificial Bee Colony and Differential Evolution (HABC-DE) is tested on two data sets. For finding the best collection of criteria to increase the value of software release while keeping expenses within the budget is the goal of the process of selecting software requirements. It is categorized as a non-deterministic polynomial (NP) hard problem and is known as the next release problem (NRP) with constrained multi-objective version.

In [73] the authors suggest a multi-objective optimization process coupled with the NSGA-II algorithm and entropy weighted method called Technique for Order Preference by Similarity to Ideal Solution (TOPSIS) to lightweight design of the dump truck carriage. A multi-objective lightweight optimization of the dump truck carriage using the NSGA-II method and the Kriging surrogate model was carried out. The best dump truck design was then chosen from among Pareto options using the entropy weight TOPSIS approach. The findings demonstrate that the optimized dump truck carriage achieves a noteworthy mass reduction of 81 kg, as much as 3.7% compared with the original carriage.

In [74] the authors suggest a hybrid, multi-objective optimization technique for designing Permanent Magnet (PM) traction motors that allows for global optimal tracking. The new methodology relies on the Artificial Bee Colony (ABC) technique, Strength Pareto Evolutionary Algorithm (SPEA2), and Differential Evolution (DE) strategy, which are used to ensure quick and consistent convergence to the ideal Pareto front. Through both relevant test functions and an application case involving an unequal teeth surface mounted permanent magnet wheel motor design, the effectiveness of the generated approach is compared to other well-established and potent algorithms from the literature.

In order to balance network lifetime and coverage, in [75] the authors take the coverage optimization problem in Wireless Sensor Networks (WSN) into consideration. These include decreasing energy use, increasing coverage rate, and increasing energy consumption equilibrium. Hybrid-MOEA/D-I and Hybrid-MOEA/D-II, two upgraded hybrid multi-objective evolutionary algorithms, have been suggested. In order to efficiently optimize sub-problems of the multi-objective optimization problem in WSN, Hybrid-MOEA/D-I, which is based on the well-known multi-objective evolutionary algorithm based on decomposition (MOEA/D), combines a Genetic Algorithm (GA) and a Differential Evolution (DE) algorithm.

2.4 Evolutionary Algorithms

Including traditional GAs, evolutionary algorithms (EAs) are a big class of optimization techniques that draw their inspiration from the process of natural evolution. According to Eiben and Smith [76], “there are many different variants of evolutionary algorithms. The common underlying idea behind all these techniques is the same: given a population of individuals within some environment that has limited resources, competition for those resources causes natural selection (survival of the fittest)”.

The following steps essentially summarize how different EA implementations (such as genetic algorithms, genetic programming, and evolutionary strategies) work:

1. Initialize a population randomly and evaluate each candidate;

2. Select parents;
3. Recombine pairs of parents;
4. Mutate the resulting offspring;
5. Evaluate each new candidate;
6. Select individuals for the next generation;
7. Repeat from Step 2 until a stopping criterion is satisfied.

Our algorithms presented in this thesis can be considered as evolutionary algorithms.

2.5 Performance measures of MOO [77]

In multi-objective optimization, there are two goals: convergence to the Pareto-optimal set and maintenance of diversity in solutions of the Pareto-optimal set [38].

There are many metrics that can be used to evaluate the quality of a set of solutions in multi-objective optimization. These metrics are often used to compare different Pareto fronts or to compare a Pareto front to an ideal solution. Some of the metrics are:

1. *Generational Distance (GD).* The GD indicator, introduced by Van Veldhuizen and Lamont in 1998 [78], measures the quadratic mean of the Euclidean distances of solutions in the given set to the closest point on the Pareto front. For a given solution set $A = \{a_1, a_2, \dots, a_N\}$, the authors define

$$GD(A) := \frac{1}{N}\left(\sum_{i=1}^{N}\left(d_2(a_i, PF)\right)^2\right)^{1/2},$$

where $d_2(a_i, PF)$ is the Euclidean distance of a_i to the true Pareto front *PF*. To compute $d_2(a_i, PF)$ in practice, usually, a finite reference set *R* is used that well represents the Pareto front; then

$$d_2(a_i, PF) = \min_{r \in R} d_2(a_i, r).$$

In [79], GD measures the quadratic mean of the Euclidean distances of solutions in the given set to the closest point on the Pareto front, while in [80], GD is defined as the arithmetic mean of these distances.

2. *Error Ratio (ER).* The ER indicator, introduced by Van Veldhuizen in 1999 [81] considers the proportion of solutions in some set *A* that are not Pareto optimal. It is defined by the formula

$$ER(A) \coloneqq \frac{\sum_{a \in A} e(a)}{N},$$

where *N* is the number of elements of the solution set *A*, and

$$e(a) \coloneqq \begin{cases} 0, & \text{if } a \in PF, \\ 1, & \text{otherwise.} \end{cases}$$

3. *Inverted Generational Distance (IGD).* The IGD, first used by Coello and Sierra (2004) [82] is amongst the most commonly used indicators. It is an inversion of the GD indicator because it measures the distance from the Pareto front to the solution set. Given a solution set *A* and a reference set $R = \{r_1, r_2, \dots, r_M\} \subseteq PF$, we have

$$IGD(A, R) \coloneqq \frac{1}{M} \sum_{i=1}^{M} \min_{a \in A} d_2(r_i, a).$$

In [83], the IGD metric was used as a measure of convergence in evolutionary algorithms for multi-objective optimization.

4. *Hyper Volume (HV).* This metric measures the volume of the objective space that is dominated by the solutions on a Pareto front. It is often used as a measure of the quality of a Pareto front, with larger values indicating a better quality Pareto front [84], [85].

Let's denote the reference point as $Z = (z_1, z_2, \dots, z_m)$ where m is the number of objectives, and z_i is the *i*-th component of the reference point. $A = \{a_1, a_2, \dots, a_n\}$ is a set of non-dominated solutions, where n is the number of solutions. The HV metric is calculated as:

$$HV(A, Z) = \int_{-\infty}^{z_1} \int_{-\infty}^{z_2} \dots \int_{-\infty}^{z_m} g(A, x_1, x_2, \dots, x_m) dx_1 dx_2 \dots dx_m$$

where $g(A, x_1, x_2, \dots, x_m)$ is a function that calculates the contribution of the set of solutions *A* to the hyper volume at points $(x_1, x_2, \dots, x_m)$.

5. *Spacing metric.* This metric measures the average distance between the solutions on a Pareto front. It is often used as a measure of diversity in evolutionary

algorithms for multi-objective optimization [86]. The following formula explains the spacing metric.

$$SP(S)=\sqrt{\frac{1}{|S|-1}\sum_{i=1}^{|S|}(\bar{d}-d_i)^2}$$

where $d_i = min_{s_i,s_j \in S, s_i \neq s_j}\left\|F(s_i) - F(s_j)\right\|_1$ is the l_1 distance between a point $s_i \in S$ and the closest point of the Pareto front approximation, and $\bar{d}$ the mean of the d_i.

where F is a vector objective function and |*S*| is the number of points in a Pareto set approximation *S*.

6. *Spread metric.* This metric measures the spread of the solutions on a Pareto front in the objective space. It is often used as a measure of the quality of a Pareto front, with larger values indicating a better quality Pareto front [87].

$$spread = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N-1}\left\|d_i - \bar{d}\right\|^2}$$

where *spread* is the spread metric, N is the number of solutions on the Pareto front, d_i is the Eculidean distance between solution i and its nearest neighbor on the Pareto front, $\bar{d}$ is the average of all the d_i.

7. DeltaP metric. This metric measures the difference between solutions on the Pareto front and solutions obtained by the algorithm. It is often used as a measure of convergence in evolutionary algorithms for multi-objective optimization [88]. We define new indicator Δ_p as follows.

Let $X = \{x_1, \dots, x_n\}, Y = \{y_1, \dots, y_m\} \subset R^k$ be finite and non-empty sets. Then we define $\Delta_p(X,Y)$ by

$$\Delta_p(X,Y) \coloneqq \max(GD_p(X,Y), IGD_p(X,Y))$$

$$= \max\left(\left(\frac{1}{N}\sum_{i=1}^{N} dist(x_i,Y)^p\right)^{1/p}, \left(\frac{1}{M}\sum_{i=1}^{M} dist(y_i,X)^p\right)^{1/p}\right)$$ [89]

In general, a smaller value of the DeltaP metric is better. The DeltaP metric is a measure of the difference in performance between two algorithms or methods and a

smaller value indicates that the difference is smaller and the algorithms or methods are more similar in performance. However, the specific meaning of "better" can vary depending on the context in which the metric is used. It is always depending on the problem to be solved and the setting of a threshold [90].

These are just a few of the many metrics that can be used to evaluate the quality of a set of solutions in multi-objective optimization. The choice of metric will depend on the specific goals and characteristics of the optimization problem.

2.7 Initial population effects

The results of preliminary studies show that a well-distributed initial population really speeds up the convergence to the correct Pareto front [91], [92], [93].

In our algorithms, a portion of the population is regenerated randomly after each cycle. So, this feature speeds up the convergence to the correct Pareto front, as we observed in the results in chapter four and five .

Most of Multi-Objective Evolutionary Algorithms (MOEA) are based on random generation of initial population [94]. This random approach frequently results in an initial population that only contains impractical solutions. Consequently, a MOEA's job is to direct the search to the feasible region as well as to converge towards the Pareto-optimal front. The authors of [94] present a novel approach called Pareto-Front-Arithmetics (PFA), The main idea of PFA is: The MOP is divided into multiple sub-problems each of which is independently optimized using a common optimization technique, where the results of the sub-optimizations are merged in the fast PFA step. The obtained non-dominant solutions are used as initial solutions to the global optimization problem. The proposed methodology has the benefit of being adaptable to any MOEA currently in use.

In our work, we have used a big initial population in one algorithm (first big population with Hybrid NSDSC with NSGA-II) in Chapter 4. It shows better results comparing with other our algorithms and also with NSGA-II on tested functions.

2.8 Modified NSGA-II

In [95] the authors suggested to modify NSGA-II to solve hub covering problems. Such models are applied in martial transportation networks, message delivery in telecommunication systems and air transport systems. To solve the proposed model, a modified version of NSGA-II was developed in which new crossover and mutation operators were introduced to adapt with structure of the considered problem. Hub covering problems, as location-allocation problems, consist of two sub-problems namely Hub Set Covering Problem (HSCP) and Hub Maximal Covering Problem (HMCP). The modified NSGA-II deviates from the standard one in three ways:

(1) Enhanced operators are created to adapt to the problem.

(2) An immigration operator is added for a better search in the solution space.

(3) A new mechanism is created for increasing the population.

The results of modified NSGA-II were better than original ones for two problems in terms of Quality Metric (QM), Spacing Metric (SM), Mean Ideal Distance (MID), see [95].

In [96] the authors suggested a Modified Non-Dominated Sorting Genetic Algorithm-II (MNSGA-II). It was applied to the multi-objective reactive power planning problem by incorporating the concept of Dynamic Crowding Distance (DCD) in NSGA-II algorithm. It was found from the simulation results that MNSGA-II performed better than NSGA-II in general.

CHAPTER THREE: The NSDSC and NS-DSDSC Algorithms

3.1 Introduction

In this chapter we present a description of NSGA-II, also, we propose two modifications of NSGA-II: the first one, by using the Dissimilarity and Similarity of Chromosomes (DSC) algorithm as optimization algorithm instead of GA, then, the second one by using the Dynamic Schema Dissimilarity and Similarity of Chromosomes (DSDSC) algorithm as optimization algorithm instead of GA. These two algorithms: Non-dominated Sorting Dissimilarity and Similarity of Chromosomes (NSDSC) and Non-dominated Sorting Dynamic Schema Dissimilarity and Similarity of Chromosomes (NS-DSDSC) were examined on 14 multi-objective test problems.

3.2 Description of NSGA-II

In [16] the author describes the NSGA-II algorithm in details, this algorithm is a new version of NSGA, where the main difference is that NSGA uses two populations: one for parents and one for offspring [22]. NSGA-II combines these populations into one, see Figure 3.1. Also, NSGA-II includes a crowding distance mechanism for selecting solutions in crowded regions of the Pareto front.

3. 2. 1 Non-dominated Sorting Genetic Algorithm (NSGA)

Many old techniques scalarize the objective vector into a single objective in an attempt to handle multiobjective optimization issues. The achieved solution in those circumstances is very dependent on the weight vector employed during the scalarization process and necessitates that the user be aware of the problem. In addition, designers may be more interested in a group of Pareto-optimal points rather than a single point when handling multiobjective issues. Considering that GAs operate on populations of points, it makes sense to use GAs to multiobjective optimization problems in order to simultaneously capture several solutions [22].

The NSGA differs from the Simple Genetic Algorithm (SGA) mainly in the way the selection operator works [22]. The crossover and mutation operators remain as usual. Before the selection is performed, the population is ranked on the basis of an individual's nondomination. The non-dominated individuals present in the population are first identified from the current population. The idea behind the nondominated sorting procedure is that a ranking selection method is used to emphasize good points and a niche method is used to maintain stable subpopulations of good points, where, the niche method in Genetic Algorithm maintains stable subpopulations of good points by sharing populations, enhancing diversity, and optimizing multimodal functions for faster and more optimal solutions [97].

NSGA originally focused on binary-coded genetic representations, but it can be adapted for real-valued representations and other encoding schemes. The NSGA uses a unique non-dominated sorting approach to classify solutions into multiple Pareto fronts. This sorting technique enables the algorithm to distinguish between solutions that are superior and inferior concerning multiple objectives. Also, NSGA was invented by Srinivas and Deb (1995) to overcome of the weakness of a Vector Evaluated GA (VEGA) (Schaffer, 1984) [22].

3. 2. 2 Non-dominated Sorting Genetic Algorithm II (NSGA-II)

The authors of [38] suggested a crowded-comparison strategy, this new strategy called Crowding Distance (CD) in NSGA-II algorithm, so the difference between NSGA and NSGA-II [98], [21], is that the NSGA-II involves the principle of "the crowded sorting of the solutions", also, the last front which could not be accommodated fully is sorted in the descending order of their crowding distance values and points from the top of the ordered list are chosen, the mechanism of crowding distance is described in Section 3.2.2.1.

The main steps of NSGA-II can be summarized as follows:

Step 1: Create an offspring population Q_t from the parent population P_t, then combine the P_t and Q_t populations: $R_t = P_t \cup Q_t$, perform a non-dominated sorting to R_t and find different fronts F_i.

Step 2: Set the new population $P_{t+1} = \emptyset$ and set $i = 1$. Until $|P_{t+1}| + |F_i| < N$ (where N is the size of P_t), perform $P_{t+1} = P_{t+1} \cup F_i$ and increase i by 1.

Step 3: Include the most widely spread solutions $(N - |P_{t+1}|)$ of F_i in P_{t+1} using the crowding distance value.

Step 4: Create an offspring population Q_{t+1} from P_{t+1} by using the crowded tournament selection, crossover and mutation operators.

For more details, in any generation t, the offspring population Q_t is first created by using the parent population P_t and the usual genetic operators. Next, the two populations are combined together to form a new population R_t of size 2*N*. Then, the population R_t is classified into different non-domination classes, see Figure 3.1 [16].

Thereafter, the new population is filled by points of different non-domination fronts, one at a time. The filling starts with the first non-domination front (of class one) and continues with points of the second non-domination front, and so on. Since the overall population size of R_t is 2*N*, not all fronts can be accommodated in *N* slots available for the new population. All fronts which could not be accommodated are deleted [16]. The pseudocode of the NSGA-II is shown in Algorithm 3.1 [99].

The NSGA-II algorithm is a multi-objective optimization algorithm that is commonly used in various fields such as engineering and computer science .

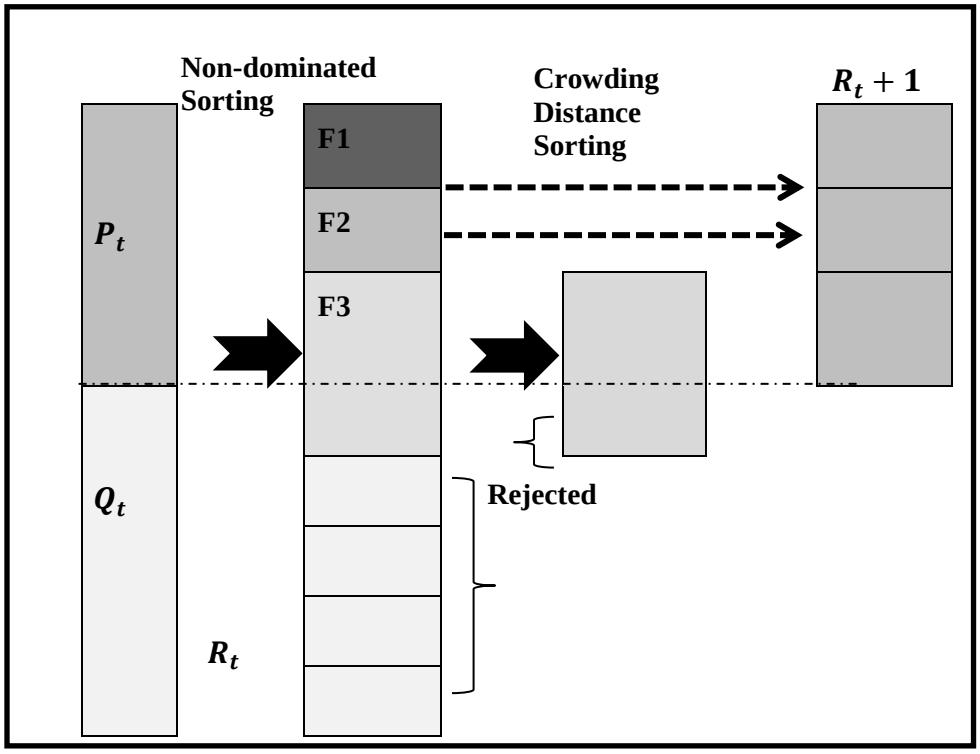

Figure 3. 1 Schematic of the NSGA-II procedure [16].

Procedure NSGA-II
Create random solutions to build an initial *population* set;
while stop Criterion Not Satisfied() **do**
for $s \leftarrow 0; s < N; s \leftarrow s + 2$ **do** % N is the size of *population*
Select two parents P_1 and P_2; % from the current *population*
Execute crossover of P_1, P_2 and generate offspring $Q_1$and Q_2;
Mutate Q_1 and Q_2;
Evaluate Q_1 and $Q_2$and insert them in the *offspring* set;
end for
Create a *union* set from *population* and *offspring* and clear *population* (old solutions);
Build Pareto fronts pfs from *union* based on the dominance rule;
Set the current front as $pfs[0]$; % the first non-dominated front
while length of $population < N$ do
Sort solutions from the current front by Crowding Distance;
for each solution in the ordered front **do**
if the length of $population < N$ **then**
Include the solution in *population*;
end if
end for
Go to the next Pareto front;
end while
end while
end procedure

Algorithm 3. 1 The pseudo-code of the NSGA-II [99]

NSGA-II uses non-dominated sorting to partition the population into different Pareto fronts. Solutions on the first front (*pfs*[0]) are non-dominated by any other solution in the population, while solutions on the second front (*pfs*[1]) are non-dominated by all solutions except those on the first front, and so on.

To compute the crowding distance of a solution, one typically looks at the distances to its nearest neighbors in the objective space. Solutions that are farther apart from their nearest neighbors are considered to have a higher crowding distance and are given more weight in the optimization process. The exact definition of crowding distance and the way it is used can vary depending on the specific algorithm and application [16].

3. 2. 2. 1 Crowding Distance

NSGA-II uses Crowding Distance (CD) measure to remove excess individuals. The individuals having lower values of CD are preferred to individuals with higher values of CD in removal process. "In MOEAs, the horizontal diversity of Pareto front is very important. The horizontal diversity is often realized by removing excess individuals in the non-dominated set (NDS) when the number of non-dominated solutions exceeds population size" [96].

The crowding distance dx_i of point x_i is a measure of the objective space around x_i which is not occupied by any other solution in the population. Here, we simply calculate this quantity dx_i by estimating the perimeter of the cuboid formed by using the nearest neighbors in the objective space as the vertices [21], see Figure 3.2.

The crowded tournament selection is based on ranking and distance. In other words, if a solution x_i has a better rank than x_{i+1} , we select x_i . If the ranks are the same but $dx_i > dx_{i+1}$, we select x_i based on its crowding distance. "we prefer the solution with the lower (better) rank. Otherwise, if both solutions belong to the same front, then we prefer the solution that is located in a lesser crowded region" [38].

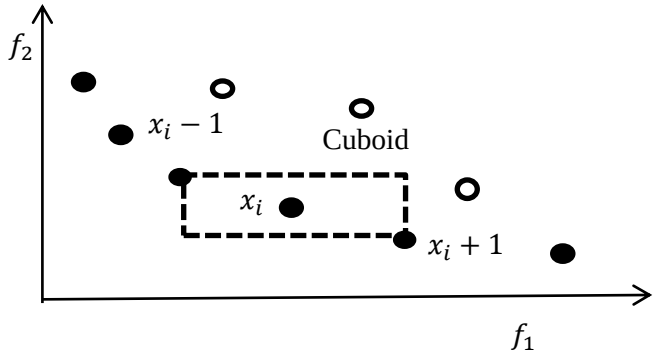

Figure 3. 2 The crowding distance calculation

Crowding distance is a measure of the "crowding" of solutions on the Pareto front. It is often used in evolutionary algorithms for multi-objective optimization to ensure that the diversity of solutions is maintained [100]. The idea is that solutions that are too close together in the objective space may be less useful for exploration, because they do not represent a significant improvement over the existing solutions. On the other hand, solutions that are farther apart may represent a larger improvement and should be given more weight in the optimization process [14].

3.3 Development of NSGA-II

The authors of [96] introduced new modification of NSGA-II called Dynamic Crowding Distance (DCD), where one individual with the lowest DCD value is removed every time and DCD is recalculated for the remaining individuals. In [96] the authors assumed that the population size is N and the nondominated set at t-th generation is $Q(t)$ and its size is M, where M is the population size of offspring $Q(t)$, If $M > N$, a DCD technique is based on removing the individual which has the lowest DCD value in the $Q(t)$ to eliminate redundant $M - N$ individuals from population.

In [95] the authors have proposed a new process called immigrants that has been added to the NSGA-II algorithm, this process depends on the results of crossover and mutation in the original algorithm. The idea is to introduce new individuals into population so that there will be diversity in society, only the successful offspring will be added to population, and the others will be replaced with immigrants.

3.4 Constraint Overview

In optimization, a constraint is a condition that must be satisfied by the solution to a problem. Constraints can be used to limit the possible solutions to a problem and ensure that the solution meets certain requirements or criteria [101], [14].

There are two types of constraints: equality constraints and inequality constraints. An equality constraint is a condition that must be satisfied exactly, such as $x + y = 10$. An inequality constraint is a condition that must be satisfied within a certain tolerance or range. It allows for some flexibility in the optimization problem, such as $x + y \leq 10$ [102]. In optimization algorithms, constraints are often represented as a set of inequalities or equalities that must be satisfied by the solution. The optimization algorithm then attempts to find a solution that meets all of the constraints while also minimizing or maximizing some objective function [102].

Constraints can be used to represent real-world conditions that must be satisfied in order for a solution to be feasible. For example, in a problem involving the design of a structure, constraints might be used to ensure that the structure can support the required load, that it meets certain safety requirements, and that it is built within a certain budget [103]. A constraint may represent a limit on a certain resource or on a certain physical phenomenon, for example, stress limitation, current or voltage restriction, etc. We should always identify the constraints associated with the optimization problem [102].

The authors of [104] have proposed a new constraint handling mechanism called Constrained Non-dominated Sorting (CNS) for constrained multi-objective optimization. In CNS, each individual of a population is assigned a constrained non-dominated rank (CNR) which is calculated according to the constraint violation degree and Pareto rank of each individual.

3. 5 The NSDSC algorithm

In our research we use the same strategy as in NSGA-II, but instead of crossover and mutation operators in Genetic Algorithm (GA), we use the dissimilarity and similarity operators, that are taken from [41], where they are applied in the DSC

algorithm (Dissimilarity and Similarity of Chromosomes). In this algorithm, the population is sorted based on the fitness function, then in the first and second quarters of population, dissimilarity and similarity operators (see Tables 3.1 and 3.2) are applied on chromosomes, then the remaining half of generation is regenerated randomly in each iteration. Figure 3.3 shows the flowchart of DSC algorithm, for more details see [41].

We summarize the idea of dissimilarity and similarity operators in the followoing points respectively.

1. The dissimilarity operator (applied in the first quarter of population): for each two sequential chromosomes A and B, if the two bits are equal, put a star (*) in the second chromosome (B); otherwise leave this bit without change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one, and so on, see Table 3.1.
2. The similarity operator (applied in the second quarter of population): for each two sequential chromosomes A and B, if the two bits are not equal, put a star (*) in the second chromosome (B); otherwise leave this bit without change in the second chromosome. Then put randomly 0 or 1 in the bits with stars (*). Compare this new second chromosome with the third one, and so on, see Table 3.2.

Table 3. 1 The dissimilarity operator [41]

Before change: example for the first quarter of chromosomes

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	0	1	1	0	0	0	1

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	*	0	1	1	0	*	0	*

After change: put randomly 0 or 1 in (*) bits

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	0	1	1	0	0	0	0

Table 3. 2 The similarity operator [41]

Before change: example for second quarter of chromosomes

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	0	1	1	0	0	0	1

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	*	*	*	*	0	*	1

After change: put randomly 0 or 1 in (*) bits

Chromosome A	1	1	0	0	1	0	1	1
Chromosome B	1	1	0	1	0	0	1	1

Briefly, the suggested algorithm called Non-dominated Sorting Dissimilarity and Similarity of Chromosomes (NSDSC) works as follows: in step one it initializes a parent population P_t of *N* solutions, then applies a part of the DSC algorithm (dissimilarity and similarity operators) to generate an offspring population Q_t, then performs non-dominated sorting of the entire population $R_t = P_t + Q_t$ to identify different fronts, then a crowding distance is applied to construct P_{t+1}. In Algorithm 3.2, we present the pseudocode of NSDSC.

Procedure NSDSC

Create random solutions to build an initial two population sets P, Q of size N;
Sort *P* by nondomination and build Pareto fronts;
while stop Criterion Not Satisfied() **do**
Copy *C* times one chromosome randomly chosen from the first front and put it in the first half of *P* randomly; % where *C* is equal to $N/8$
for $s \leftarrow 0; s < N; s \leftarrow s + 1$ **do** % s is the index of a chromosome
if the first quarter of population *P* **then**
Apply the dissimilarity operator on chromosomes ch_s, ch_{s+1} from population *P* and put the result in the offspring set *Q*;
else if the second quarter of population *P* **then**
Apply the similarity operator on chromosomes ch_s, ch_{s+1} from population *P* and put the result in the offspring set *Q*;
else
Generate randomly the second half of offspring set *Q*;

end for
Create a union set $R = P \cup Q$ from population and offspring and clear population *P*;
Build Pareto fronts pfs from union *R* based on the dominance rule;
Set the current front as $pfs[0]$; % the number will increase in each next front
while the length of $P < N$ **do** % until the last chromosome
Sort solutions from the current front by Crowding Distance;
for each solution in the ordered front **do**
if the length of $P < N$ **then**
Include the solution in population *P*;
end if
end for
Go to the next Pareto front;
end while
end while

end procedure

Algorithm 3. 2 The pseudo-code of the NSDSC Algorithm

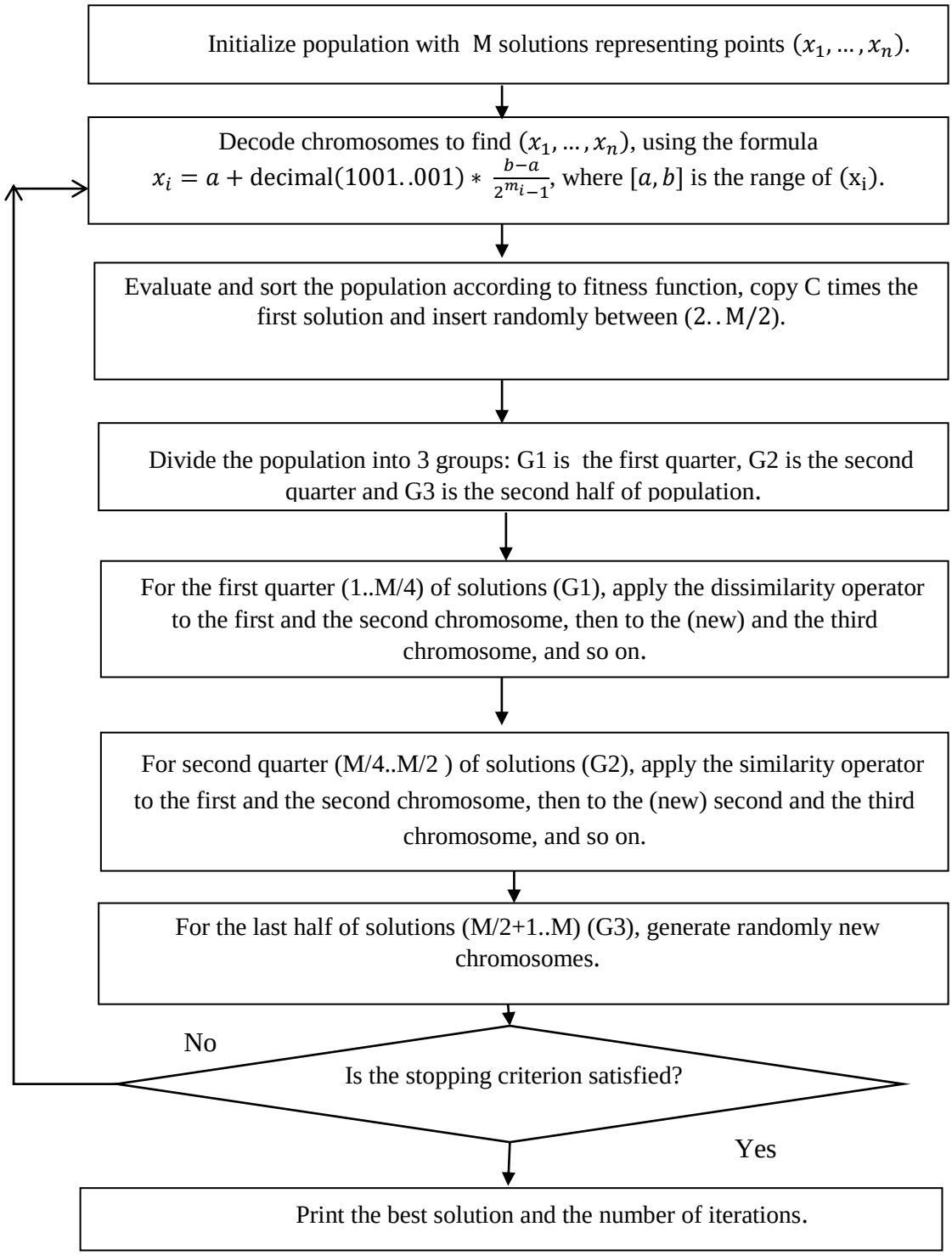

Figure 3.3 Flowchart of the DSC algorithm [41], [12].

3.6 Experiment results of the NSDSC algorithm

In this section we will show the experimental results (by using the Matlab software) of the NSDSC algorithm on 14 different functions, the first 9 functions in Appendix 1 are without constraint, then the last 5 functions have constraints.

The Schaffer function A1 (SCH) has one variable, while the following functions: Binh and Korn function (BNH), Binh and Korn function (BNH) with constraint, SRN, TNK, CONSTR, have 2 variables. The Kursawe function has 3 variables. The OSY function has 6 variables. The two functions: Zitzler–Deb–Thiele's function A4 (ZDT4) and Zitzler–Deb–Thiele's function A6 (ZDT6) have 10 variables. The three functions: Zitzler–Deb–Thiele's function N.1 (ZDT1), Zitzler–Deb–Thiele's function A5 (ZDT2) and Zitzler–Deb–Thiele's function A3 (ZDT3) have 30 variables.

The experimental results show that our suggested algorithm has good results and is faster than NSGA-II when the number of variables is equal to 1, 2, 3, 6, but when the number of variables is 10 or 30, the algorithm does not reach the optimum solutions. The following figures show results for all the functions, where the blue points are generated by the suggested algorithm (NSDSC) and the red points are generated by NSGA-II.

In Figure 3.4 and Figure 3.5, the NSDSC algorithm shows that the most of solutions are grouped (in the ellipse in Figures) close to the middle solutions in Pareto set.

The parameters used for the NSGA-II algorithm are set as follows: population size = 200, number of generations = 1000, crossover rate = 90%, mutation rate = 10%. While the parameters used for NSDSC algorithm are set as follows: population size = 200, number of generations = 1000.

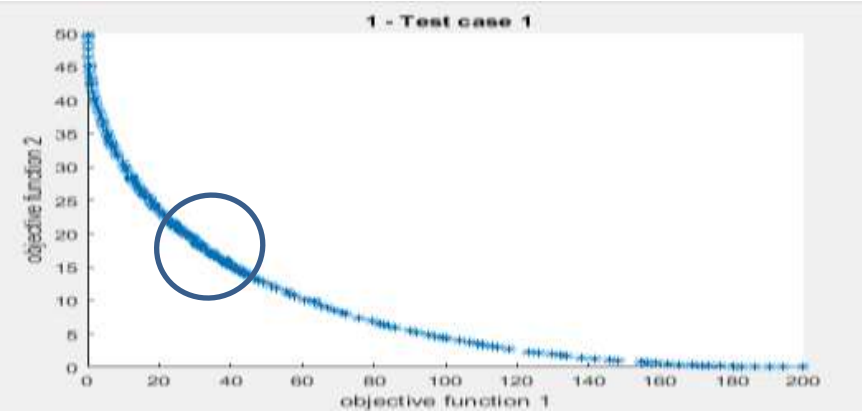

Figure 3. 4 Binh and Korn function (BNH), the points inside ellipse show the density of solutions in the Pareto front using NSDSC.

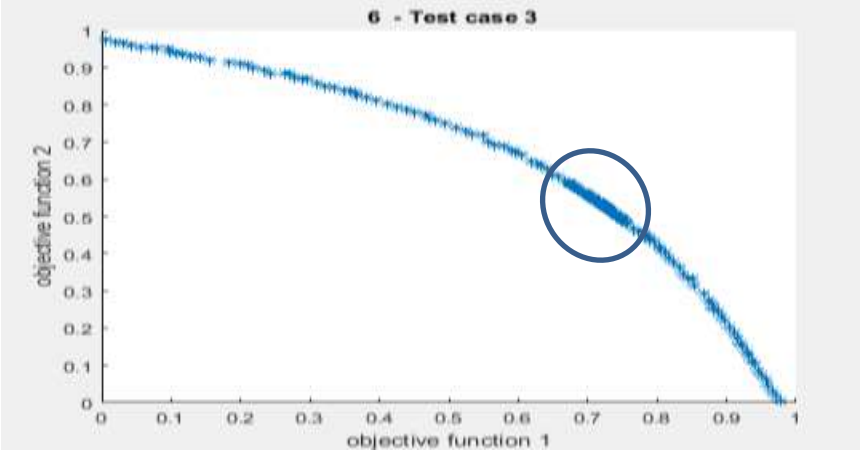

Figure 3. 5 Fonseca–Fleming function (FON), the points inside ellipse show the density of solutions in the Pareto front using NSDSC

In Figures 3.6 to 3.19, we show the results of applying two algorithms (NSDSC and NSGA-II) on all 14 problems, red points represent the NSGA-II algorithm and blue points represent the NSDSC algorithm. Also, the problems A2-ZDT1, A5-ZDT2, A7-ZDT3, A8-ZDT4, A9-ZDT6 don't have got good results in (NSDSC).

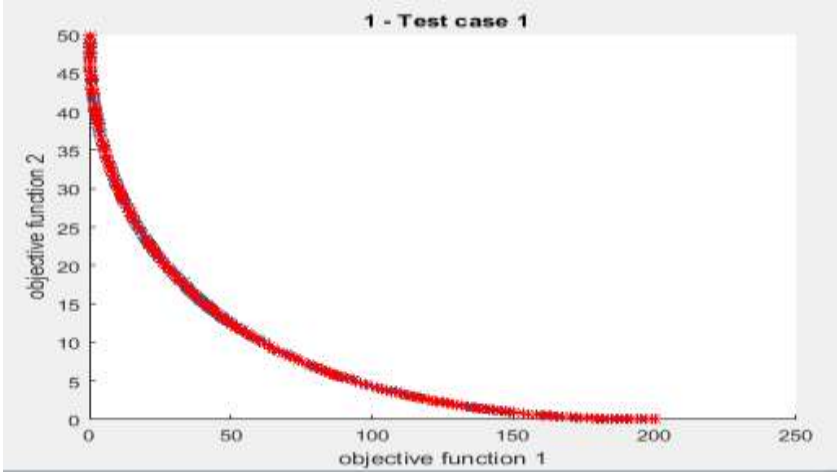

Figure 3.6 Binh and Korn problem using NSGA-II and NSDSC.

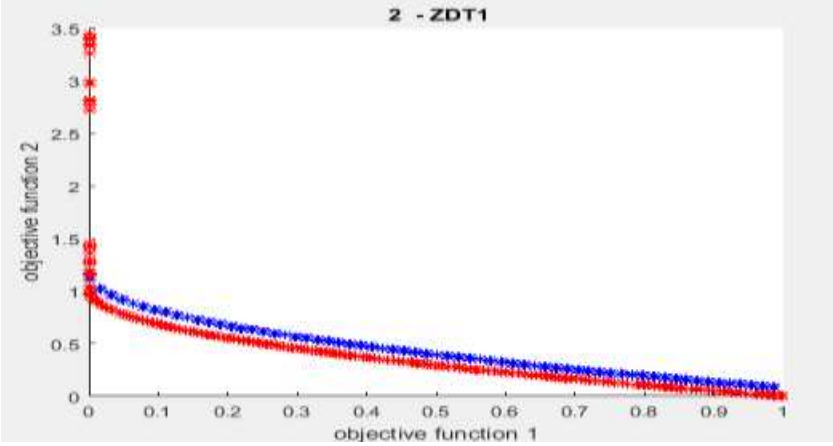

Figure 3.7 Zitzler–Deb–Thiele's problem A2-(ZDT1) using NSGA-II and NSDSC.

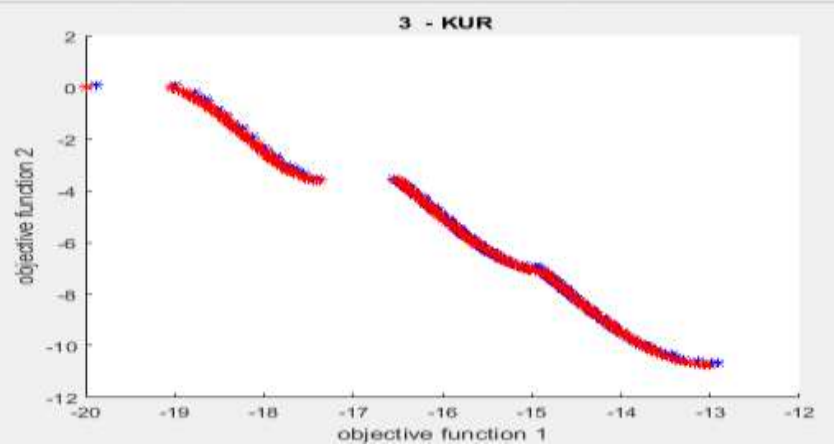

Figure 3.8 Kursawe problem using NSGA-II and NSDSC.

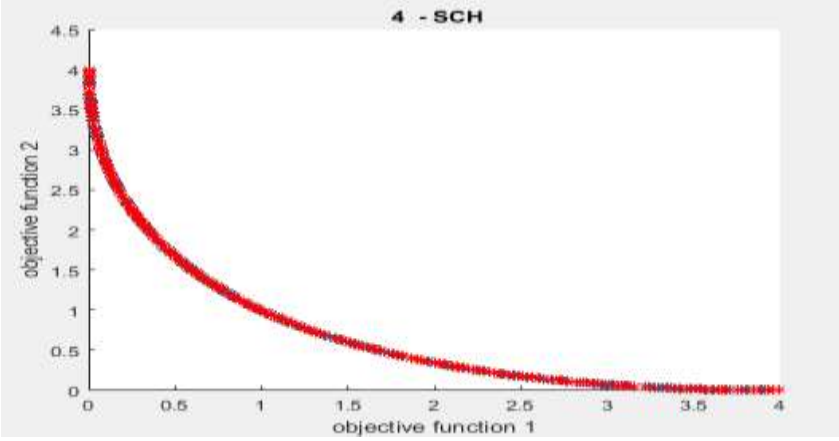

Figure 3.9 Schaffer problem using NSGA-II and NSDSC.

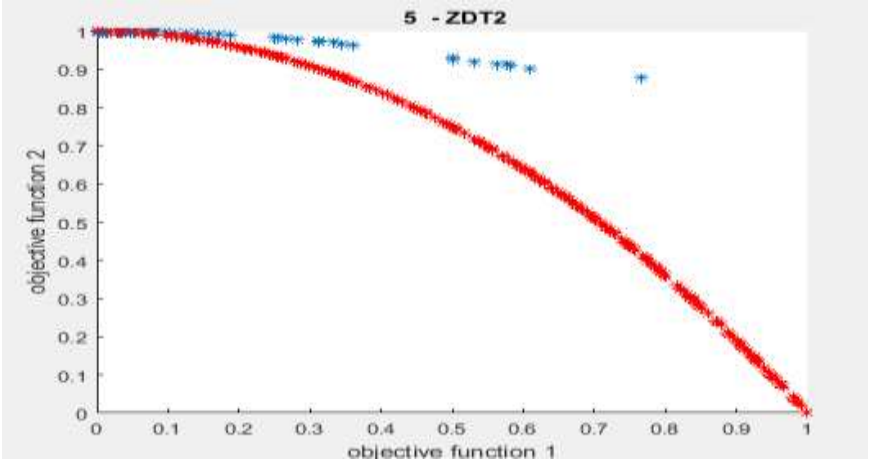

Figure 3.10 Zitzler–Deb–Thiele's A5-(ZDT2) problem using NSGA-II and NSDSC.

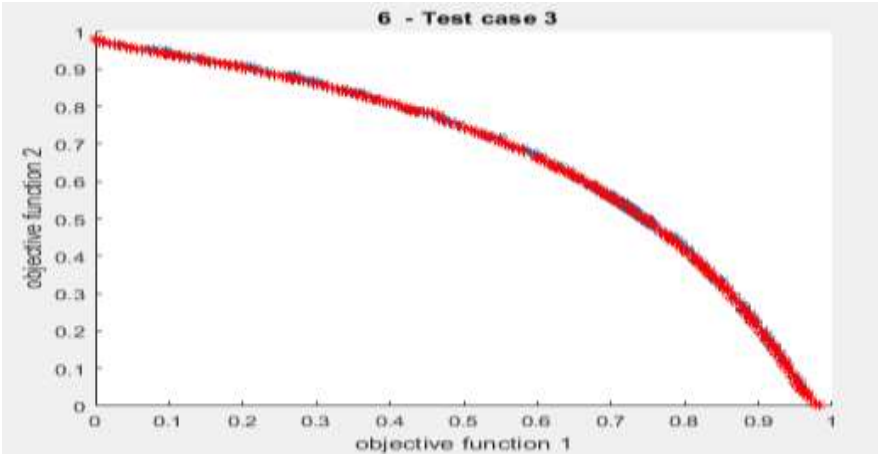

Figure 3. 2 Fonseca Fleming problem (FON) using NSGA-II and NSDSC.

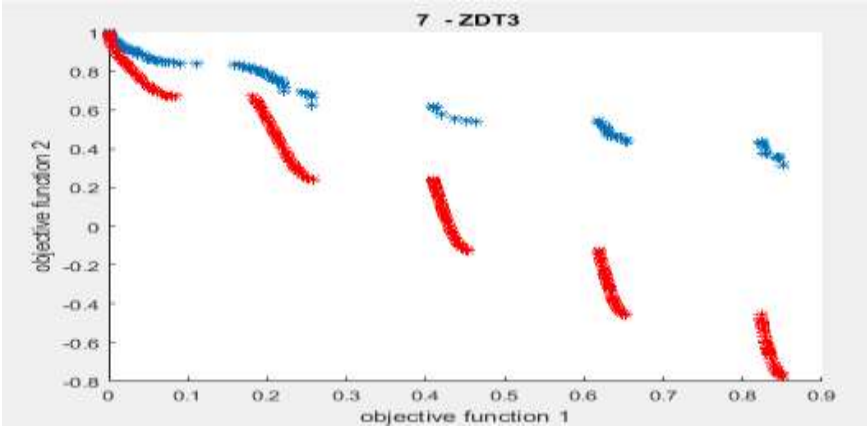

Figure 3. 12 Zitzler–Deb–Thiele's problem A7-(ZDT3) using NSGA-II and NSDSC.

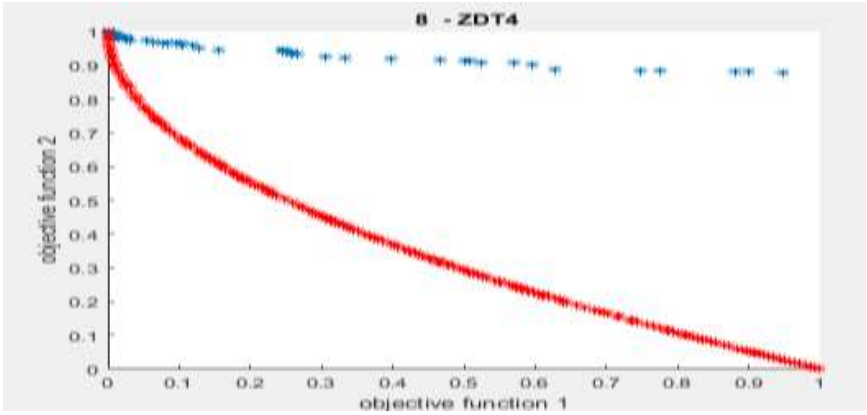

Figure 3. 13 Zitzler–Deb–Thiele's problem A8-(ZDT4) using NSGA-II and NSDSC.

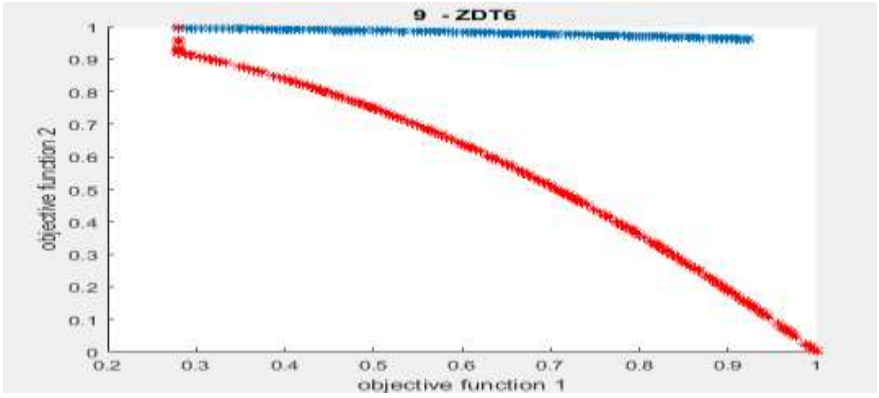

Figure 3. 14 Zitzler–Deb–Thiele's problem A9-(ZDT6) using NSGA-II and NSDSC.

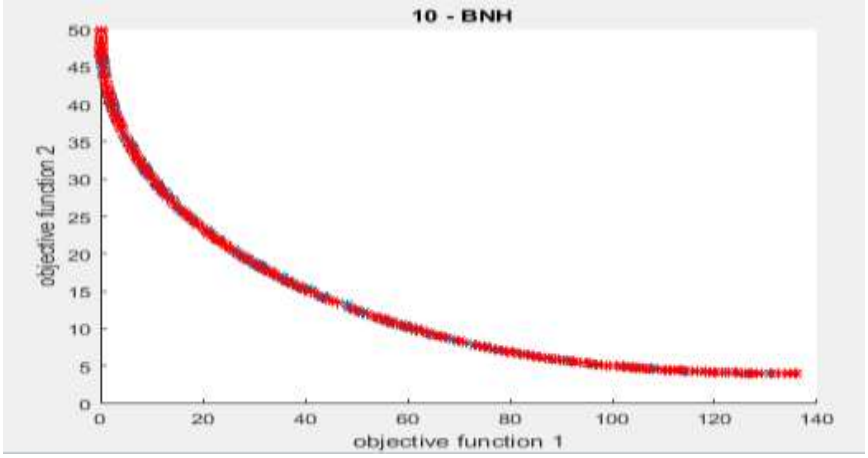

Figure 3. 15 Binh and Korn problem with constraint using NSGA-II and NSDSC.

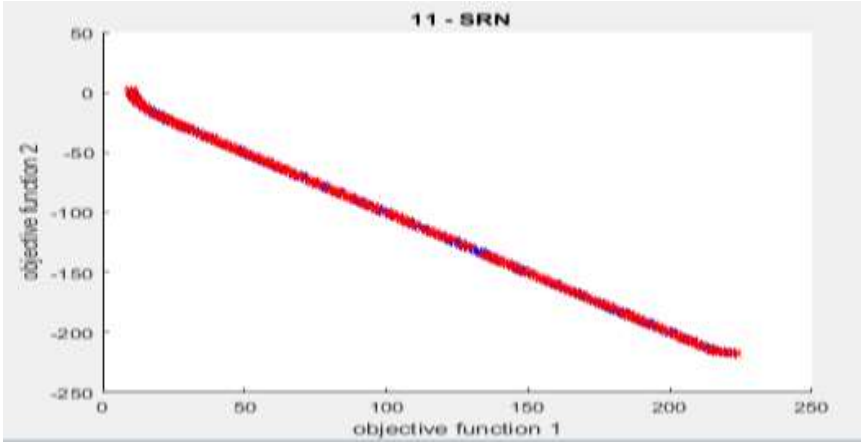

Figure 3. 16 SRN problem using NSGA-II and NSDSC.

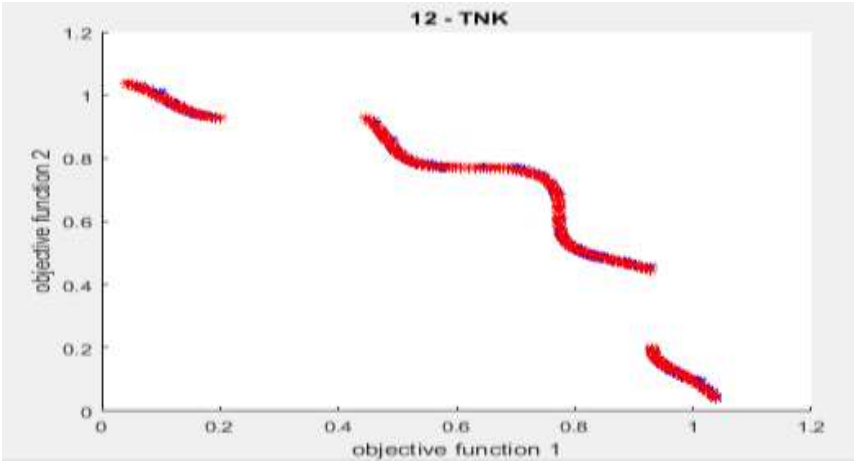

Figure 3. 17 The TNK problem using NSGA-II and NSDSC.

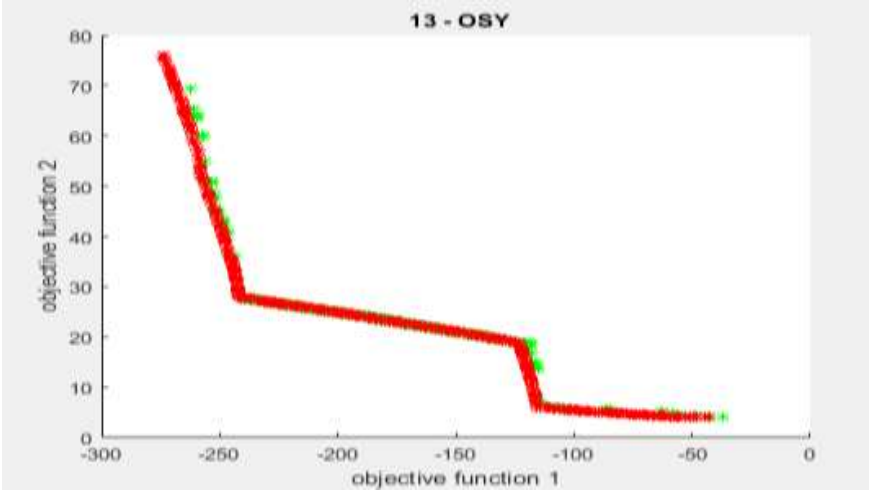

Figure 3. 18 the OSY problem using NSGA-II and NSDSC (green points).

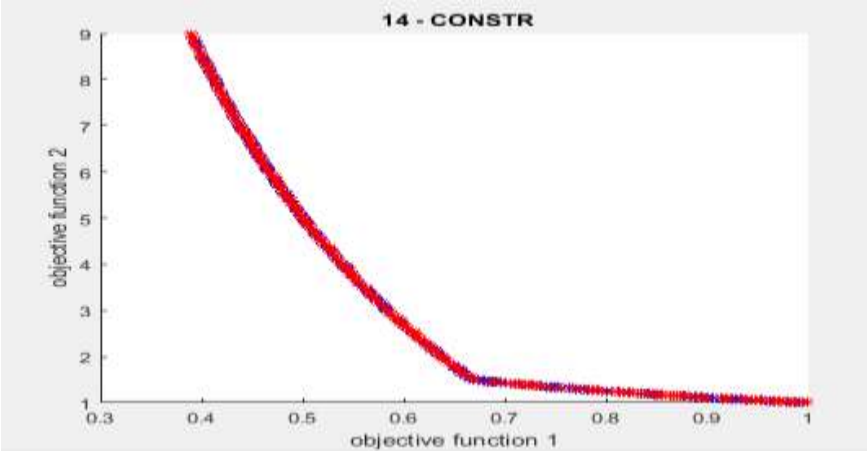

Figure 3. 19 The CONSTR function, using NSGA-II and NSDSC.

In Table 3.1, we show the results of two algorithms (NSDSC, NSGA-II) in run time in seconds, where for the problems A1-BNH, A3-KUR, A4-SCH, A6-FON, A10-BNH with constraint, A11-SRN, A12-TNK, A13-OSY and A14-CONSTR we have reached the Pareto front as an NSGA-II algorithm but superior in run time, but for some problems A2-ZDT1, A5-ZDT2, A7-ZDT3, A8-ZDT4 and A9-ZDT6, we have not reached the Pareto front, see Figures 3.6 to 3.19.

Table 3. 3 Comparison of NSDSC and NSGA-II in run time

Function name	NSDSC time in seconds	NSGA-II time in seconds
A1-BNH	24.1824	36.8989
A2-ZDT1	42.0908 Not pass *	48.0775
A3-KUR	27.9392	45.3970
A4-SCH	24.3879	40.6266
A5-ZDT2	40.1732 Not pass *	48.7190
A6-FON	27.1879	44.0588
A7-ZDT3	40.7832 Not pass *	46.5388
A8-ZDT4	28.2977 Not pass *	42.6555
A9-ZDT6	30.1160 Not pass *	43.2656
A10-BNH constraint	29.1140	37.0089
A11-SRN	25.3399	38.9880
A12-TNK	25.1719	36.6221
A13-OSY	125.9281	173.8189
A14-CONSTR	26.2113	35.2914

* In problems A2, A5 and A7 the NSDSC has not got the optimum solutions of Pareto because those problems have 30 variables, while A8 and A9 has 10 variables.

3.7 The NS-DSDSC algorithm

In this section we have introduced a new algorithm called Non-dominated Sorting of Dynamic Schema of chromosomes with Dissimilarity and Similarity of Chromosomes (NS-DSDSC), in this algorithm we used the DSDSC instead of GA to find the optimum solution. Figure 3.23 shows the flowchart of DSDSC algorithm, for more details see [41].

In the DSDSC algorithm, we divide the population into four equal parts (Q1, Q2, Q3, Q4); for Q1 and Q2 we apply the dissimilarity and similarity operator, respectively, for Q3 we apply the dynamic schema operator, and Q4 is regenerated randomly. The dynamic schema operator requires that the some high significant bit(s) are fixed for each variable x_i, then we put *'s on some of the remaining bits by using the similarity

operator. This type of schema is used to determine the area of the solution in search space, see Table 3.4, where m_1 represents the number of bits for variable x_1, R_1 represents the number of bits to be fixed (from m_1 bits), and $m_1 - R_1$ represents the bits that have to be changed based on similarity, and so on for all x_i, this operator is named the dynamic schema operator, where R_i was generated randomly, and *M* is the number of chromosomes in the population.

Table 3. 4 The dynamic schema operator [12]

Before change: an example for finding schema from the first chromosome and the chromosome on position *M*/4. Here shadow bits are not destroyed.

No. of Ch.	m_1						m_2			
	R_1		$m_1 - R_1$				R_2	$m_2 - R_2$		
Ch_1	1	1	0	0	1	0	1	0	1	0
$Ch_{M/4}$	0	1	1	0	0	1	0	0	0	1
Schema	1	1	*	0	*	*	1	0	*	*

After finding the schema: put it in *M*/2+1…*M*/2+*M*/4 positions

$Ch_{M/2+1}$	1	1	*	0	*	*	1	0	*	*
$Ch_{M/2+2}$	1	1	*	0	*	*	1	0	*	*
Ch. …	1	1	*	0	*	*	1	0	*	*
Ch. …	1	1	*	0	*	*	1	0	*	*
$Ch_{M/2+M/4}$	1	1	*	0	*	*	1	0	*	*

After change: put randomly 0 or 1 in (*) bits

$Ch_{M/2+1}$	1	1	1	0	1	0	1	0	0	1
$Ch_{M/2+2}$	1	1	1	0	0	0	1	0	1	1
Ch. …	1	1	0	0	1	0	1	0	1	0
Ch. …	1	1	0	0	0	1	1	0	0	0
$Ch_{M/2+M/4}$	1	1	1	0	1	1	1	0	1	1

The suggested algorithm NS-DSDSC in step one initializes population with M solutions representing parent population P_t, then, applies DSDSC algorithm for generating offspring Q_t, then performs non-dominated sorting of the entire population $R_t = P_t + Q_t$ to identify different fronts, the crowding distance for finding the P_{t+1}, this methodology is similar to the previous algorithm NSDSC that contains the crowding distance but only the optimization algorithm is different. The NS-DSDSC algorithm is described in the following Algorithm 3.3 as a pseudo-code.

Procedure NS-DSDSC
Create random solutions to build an initial two populations P, Q of size N Divide population P into four quarters; % divide population into four groups of equal size, each group has different operator applied to it
while stop Criterion Not Satisfied() **do**
Copy C times one chromosome randomly from first front and put it in first half of P randomly;
Apply the dissimilarity and similarity operator for group 1 and group 2 of population:
for $s \leftarrow 0; s < M; s \leftarrow s + 1$ **do** % M is population size, s is index of solution
if first quarter of population **then**
Apply dissimilarity operator, then find and copy the schema, put the schema on third quarter of offspring Q then put randomly 0 or 1 in (*) bits of the schema in the third quarter and put in offspring set Q;
else if second quarter of population **then**
Apply similarity operator and put in offspring set Q;
else
generate randomly fourth quarter and put in offspring set Q;
end for
Create a union set from population P and offspring Q and clear population;
Build Pareto fronts pfs from union based on dominance rule;
Set the current front as $pfs[0]$;
while length of $population < M$ do % until last chromosome
Sort solutions from the current front by Crowding Distance;
for each solution in the ordered front **do**
if the length of $population < M$ **then**
Include the solution in population;
end if
end for
Go to the next Pareto front;
end while
end while
end procedure

Algorithm 3. 3 The pseudo-code of the NS-DSDSC Algorithm

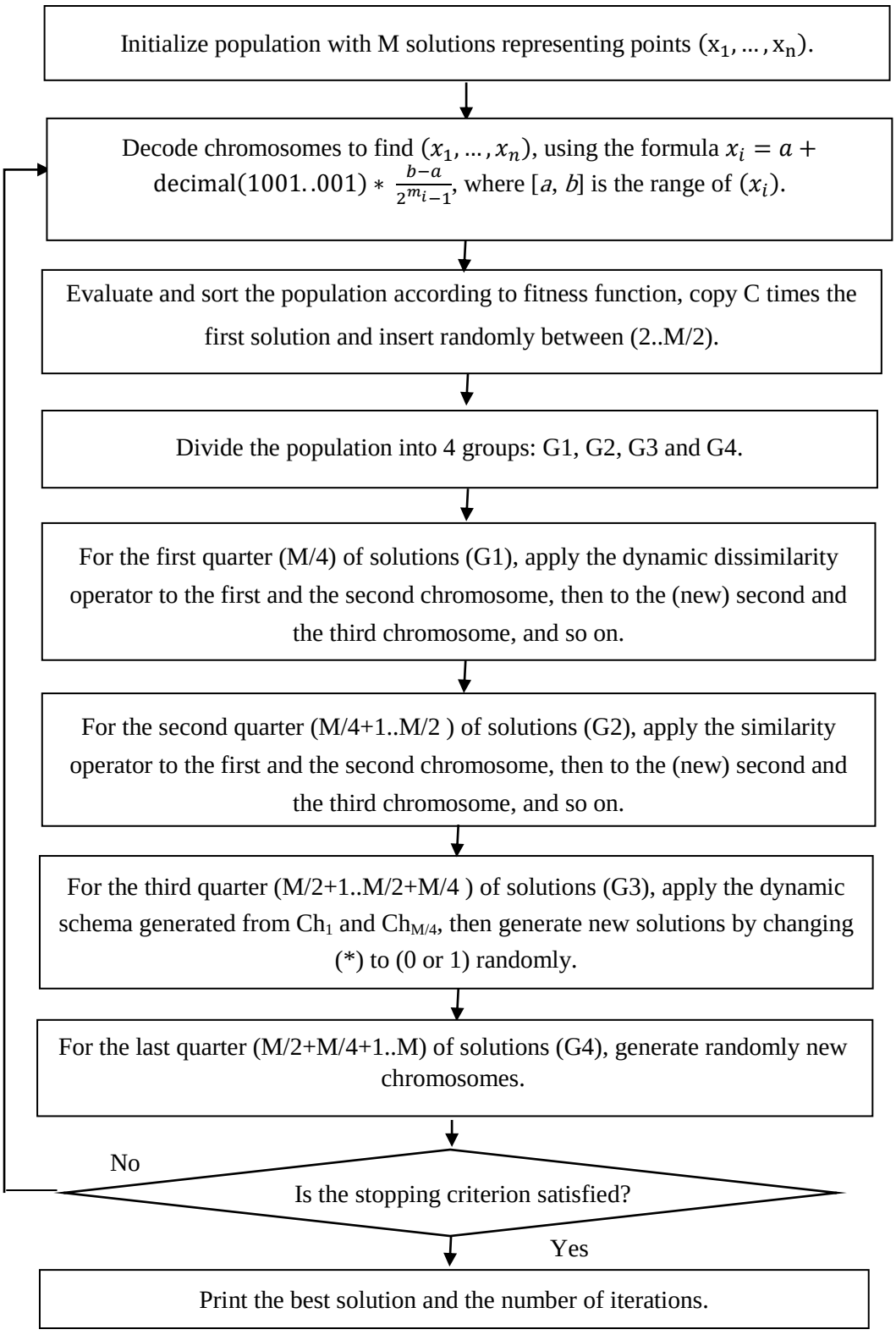

Figure 3. 20 Flowchart of the DSDSC algorithm [12].

3.8 Experiment results of NS-DSDSC algorithm

In this section the experimental results of the NS-DSDSC algorithm are shown, we apply this algorithm on 14 different problems, as mentioned in Section 3. 6. Also, see Appendix A.

The experimental results show that our suggested algorithm has good results and is faster than NSGA-II when the number of variables is equal to 1, 2, 3, 6, but when the number of variables is 10 or 30 the algorithm does not reach the optimum solutions. Except for the problem A2-ZDT1, we have got better results than previous algorithm NSDSC. The following figures show results for all the functions, where the blue points are generated by suggested algorithm (NS-DSDSC) and the red points are generated by NSGA-II.

The parameters used for NSGA-II algorithm are set according to following: (population size =200, number of generations =1000, crossover rate = 90%, mutation rate = 10%). While the parameters used for NS-DSDSC algorithm are set according to following: (population size =200, number of generations =1000).

In Figures 3.21 to 3.34, we show the results of applying two algorithms (NS-DSDSC and NSGA-II) on all 14 problems, red points represent the NSGA-II algorithm and blue points represent the NS-DSDSC algorithm. Also, the problems A2-ZDT1, A5-ZDT2, A7-ZDT3, A8-ZDT4 and A9-ZDT6 don't have got good results in NS-DSDSC, but we can observe a little bit improvement in A2-ZDT1 using NS-DSDSC.

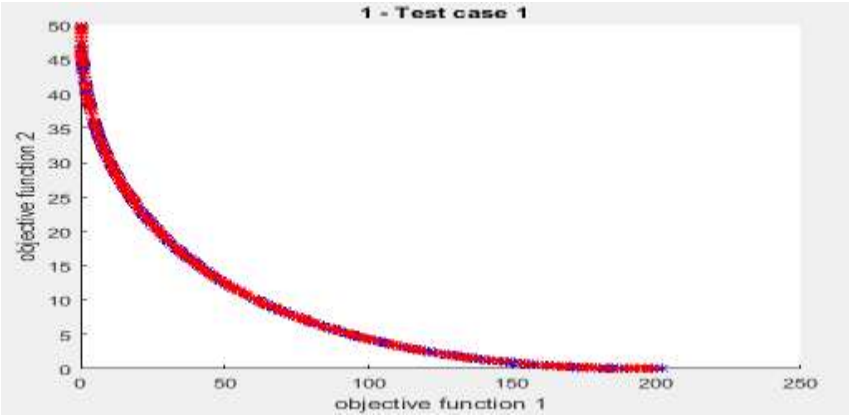

Figure 3. 3 Binh and Korn problem using NSGA-II and NS-DSDSC.

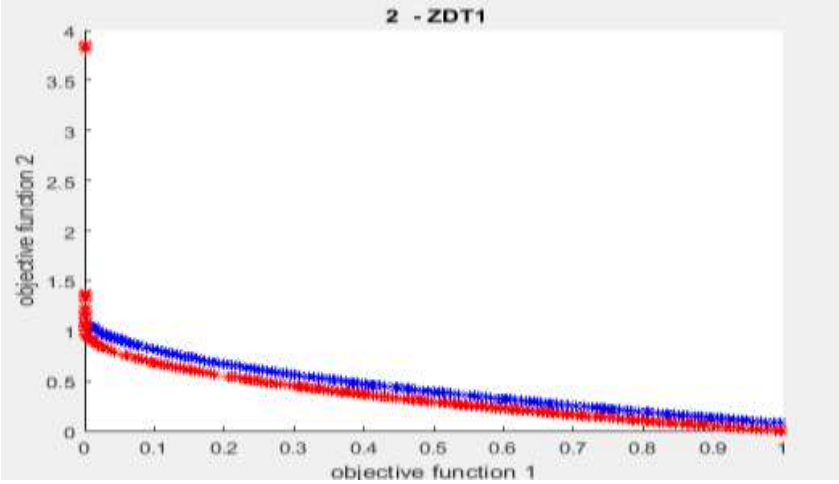

Figure 3. 4 Zitzler–Deb–Thiele's problem using NSGA-II and NS-DSDSC.

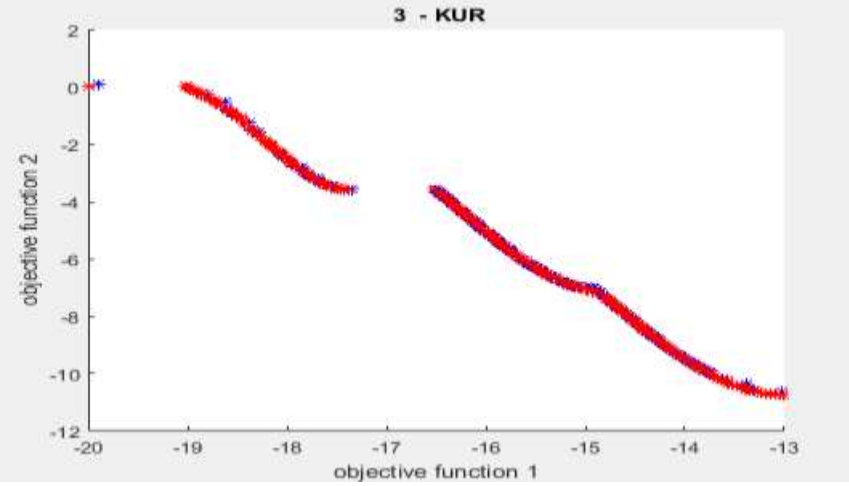

Figure 3. 5 Kursawe problem using NSGA-II and NS-DSDSC.

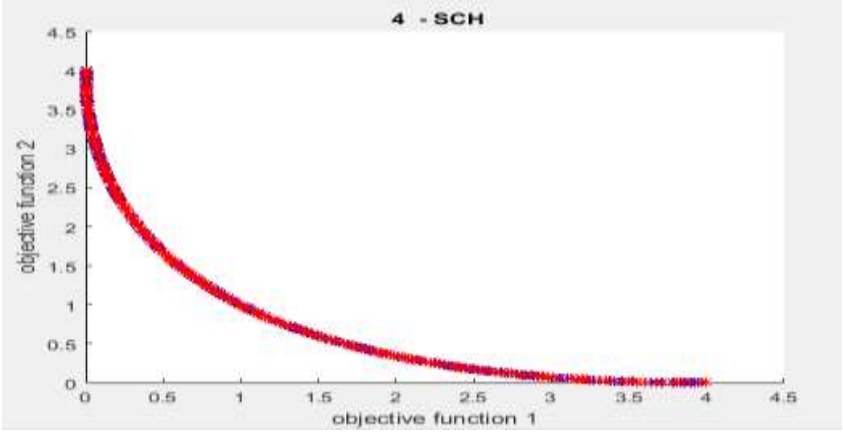

Figure 3. 6 Schaffer function (SCH), using NSGA-II and NS-DSDSC.

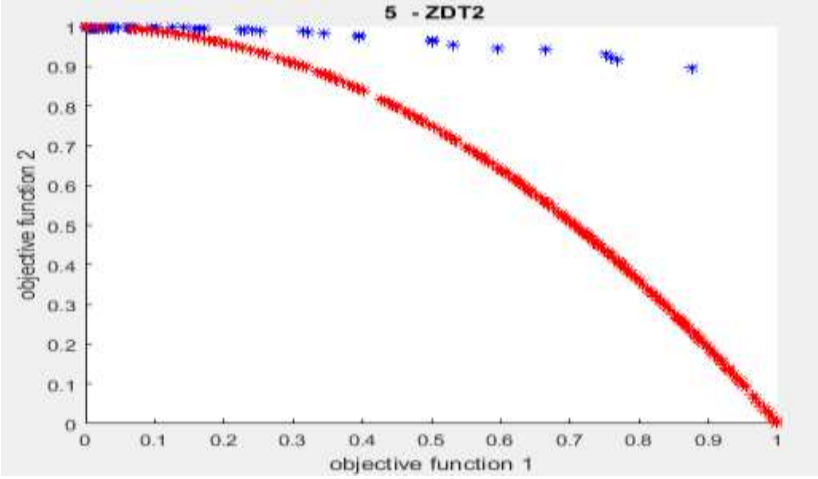

Figure 3. 7 Zitzler–Deb–Thiele's problem using NSGA-II and NS-DSDSC.

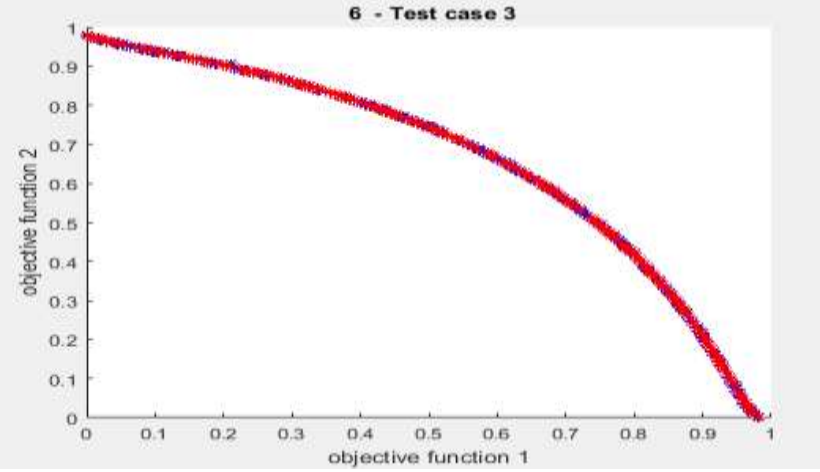

Figure 3. 8 Fonseca–Fleming problem using NSGA-II and NS-DSDSC.

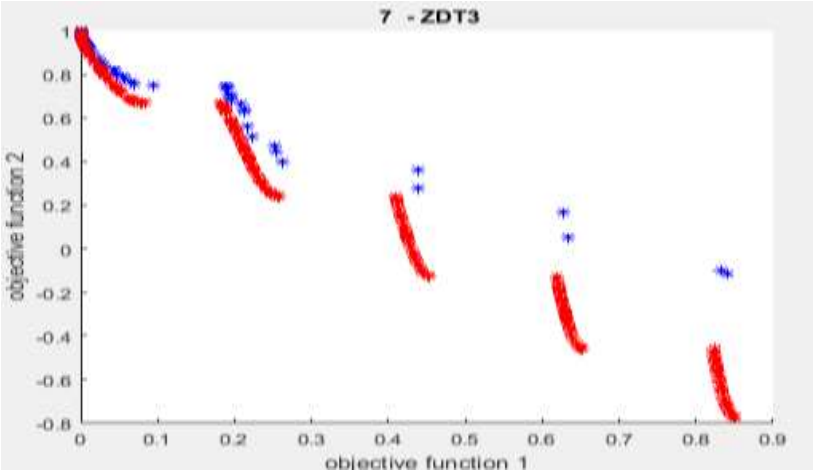

Figure 3. 27 Zitzler–Deb–Thiele's problem using NSGA-II and NS-DSDSC.

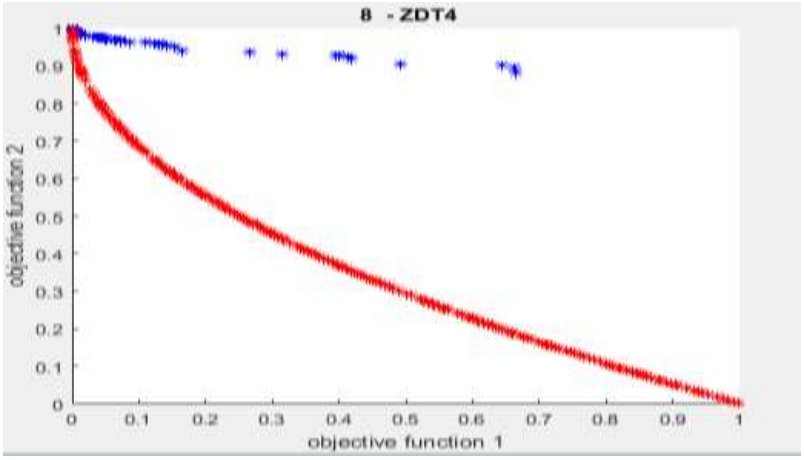

Figure 3. 28 Zitzler–Deb–Thiele's using NSGA-II and NS-DSDSC.

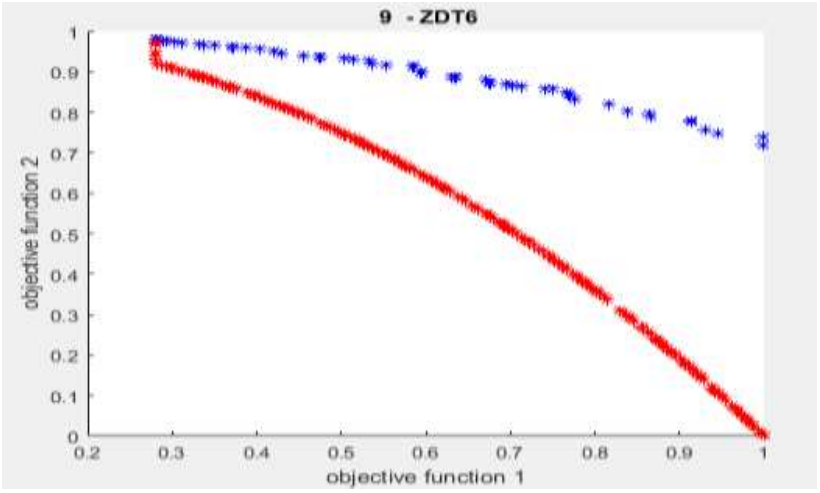

Figure 3. 29 Zitzler–Deb–Thiele's problem using NSGA-II and NS-DSDSC.

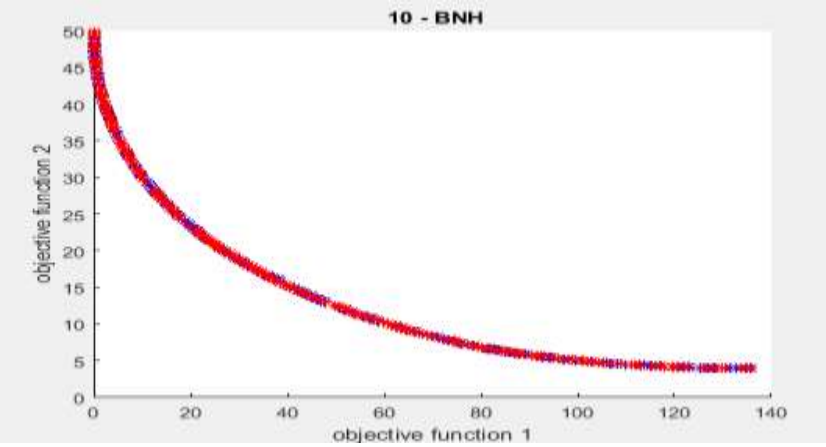

Figure 3. 30 Binh and Korn problem with constraint using NSGA-II and NS-DSDSC.

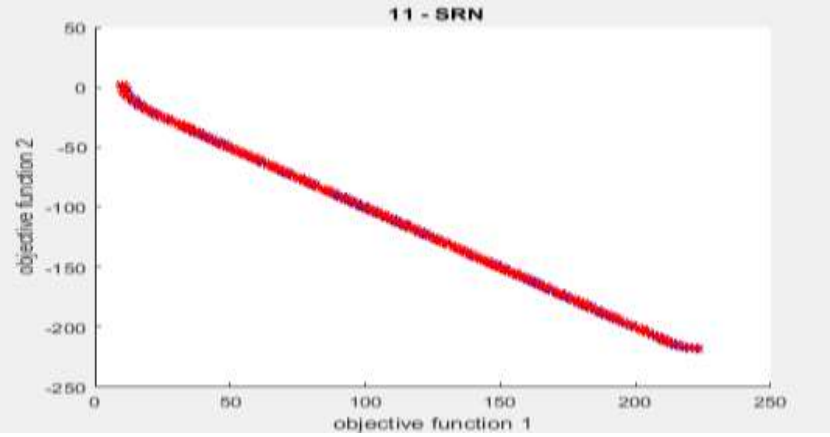

Figure 3. 9 SRN problem with constraint using NSGA-II and NS-DSDSC.

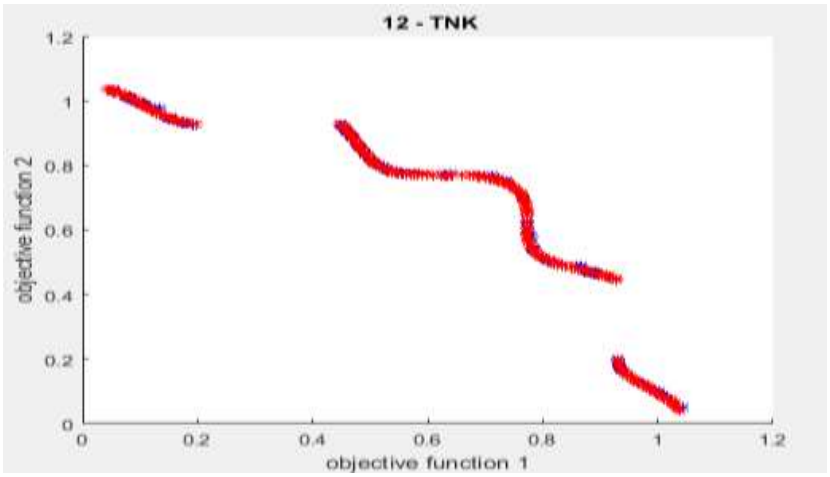

Figure 3. 10 The TNK problem with constraint using NSGA-II and NS-DSDSC.

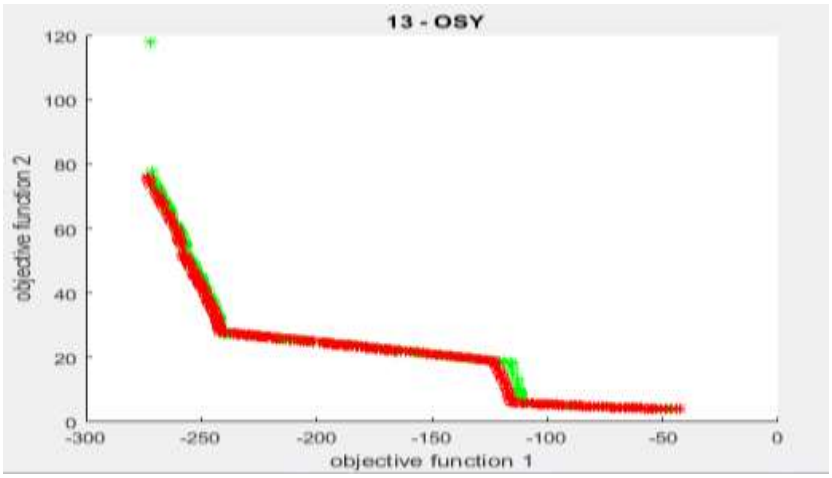

Figure 3. 11 The OSY problem with constraint using NSGA-II and NS-DSDSC.

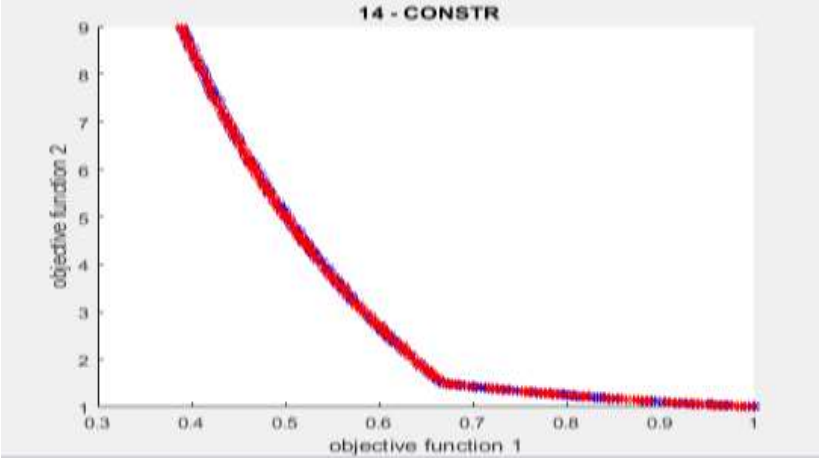

Figure 3. 12 The CONSTR problem with constraint using NSGA-II and NS-DSDSC.

In Table 3.5 we have made a comparison in run time among three algorithms (NSDSC, NS-DSDSC and NSGA-II), we have found that the run time of two new algorithms (NSDSC and NS-DSDSC) is less than NSGA-II in all 14 problems, we observed that NS-DSDSC is slower a little bit than NSDSC. But on the other side for some problems, we have not reached the optimum Pareto front by using those new algorithms NSDSC and NS-DSDSC on problems A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4 and A9-ZDT6, because they have 10 or 30 variables.

Table 3. 5 Comparison of NSDSC, NS-DSDSC and NSGA-II in run time

Function name	NSDSC Time in second	NS-DSDSC Time in second	NSGA-II Time in second
A1-BNH	24.1824	31.8522	36.8989
A2-ZDT1	42.0908 Not pass *	43.5415 Not pass *	48.0775
A3-KUR	27.9392	33.5209	45.3970
A4-SCH	24.3879	28.5681	40.6266
A5-ZDT2	40.1732 Not pass *	38.8339 Not pass *	48.7190
A6-FON	27.1879	32.5627	44.0588
A7-ZDT3	40.7832 Not pass *	32.4316 Not pass *	46.5388
A8-ZDT4	28.2977 Not pass *	32.2049 Not pass *	42.6555
A9-ZDT6	30.1160	34.7615	43.2656

	Not pass *	Not pass *	
A10-BNH constraint	29.1140	30.5356	37.0089
A11-SRN	25.3399	23.5226	38.9880
A12-TNK	25.1719	23.6143	36.6221
A13-OSY	125.9281	254.1377	173.8189
A14-CONSTR	26.2113	29.3958	35.2914

* In problems A2, A5 and A7 by using NSDSC and NS-DSDSC, we have not got the optimum solutions of Pareto because those problems have 30 variables, while A8 and A9 have 10 variables, and we also have not got optimum solutions.

3.9 Conclusion

We have observed that our suggested algorithm gives good results with functions that have the number of variables 1, 2, 3, 6, and with functions which have the number of variables 10 and 30, we haven't obtained good results.

We noticed that the two new algorithms (NSDSC and NS-DSDSC) did not reach the optimal solution in some of problems A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4 and A9-ZDT6, but they were faster in reaching the optimal solution in the rest of the problems A1-BNH, A3-KUR, A4-SCH, A6-FON, A10-BNH with constraint, A11-SRN, A12-TNK, A13-OSY and A14-CONSTR.

Also we noticed that the optimal solutions of the two new algorithms (NSDSC and NS-DSDSC) matched the optimal solutions of the NSGA-II algorithm in the problems that reached the optimal solution.

CHAPTER FOUR: Hybrid NSDSC with NSGA-II algorithm and Applying a First Big Population with Hybird algorithm

4. 1 Introduction

In this chapter we present two new algorithms: the first one is a hybrid NSDSC with NSGA-II algorithm, the second one applies first big population one time then hybrid NSDSC with NSGA-II algorithm. The results are better than the results we obtained from algorithms NSDSC and NS-DSDSC respectively in Chapter 3.

4. 2 Hybrid Multi-Objective Optimization

A hybrid Multi-Objective Optimization (MOO) algorithm is an optimization algorithm that combines multiple techniques or approaches in order to solve a multi-objective optimization problem [74].

Multi-objective optimization problems involve finding a set of solutions that are optimal with respect to multiple conflicting objectives. A common approach to solving multi-objective optimization problems is to use evolutionary algorithms, such as genetic algorithms or evolutionary strategies, which are able to search a large solution space and find a diverse set of high-quality solutions [73].

Hybrid MOO algorithms combine multiple techniques or approaches in order to improve the performance of the optimization process. For example, a hybrid MOO algorithm might combine an evolutionary algorithm with a local search method in order to find a set of high-quality solutions more efficiently. Other techniques that might be used in a hybrid MOO algorithm include particle swarm optimization, simulated annealing, and gradient-based optimization methods [5].

Overall, the goal of a hybrid MOO algorithm is to find a set of non-dominated solutions that represent a trade-off between the conflicting objectives of the optimization problem. These solutions can be used to help decision-makers understand the trade-offs involved in solving the problem and choose the best course of action [72].

4. 3 Background of Hybridization

Particle Swarm Optimization (PSO) and NSGA-II (Non-dominated Sorting Genetic Algorithm II) are two algorithms used in the field of optimization.

Particle Swarm Optimization (PSO) is a population-based optimization algorithm that is inspired by the behavior of a flock of birds or a school of fish. It is a simple but effective algorithm that can be used to find the optimal solution to a wide range of optimization problems [105].

NSGA-II (Non-dominated Sorting Genetic Algorithm II) is a multi-objective optimization algorithm that can be used to find a set of non-dominated solutions to a given optimization problem. It is based on the concept of Pareto optimality defined in Section 3.4. This algorithm was described in Section 3.2.2, see also [106].

A hybridized PSO-NSGA-II algorithm combines the strengths of both PSO and NSGA-II to find a set of non-dominated solutions to an optimization problem. The algorithm combines the global search capabilities of PSO with the non-dominated sorting and selection mechanisms of NSGA-II to find a diverse set of high-quality solutions. This can be particularly useful in cases where the optimization problem has multiple conflicting objectives that need to be balanced [105].

4. 4 A Hybrid NSDSC with NSGA-II Algorithm

In this section we introduce a new hybrid Multi-Objective Optimization algorithm, this new algorithm is based on two algorithms: the first one is NSDSC that we have introduced in Chapter 3, the second one is normal NSGA-II. This new algorithm is named it Hybrid NSDSC with NSGA-II, see Algorithm 4.1.

In this new method, we apply the NSDSC algorithm for 100 iterations, then we take the last generation as an input for NSGA-II for 350 iterations. Figure 4.1 shows a simple flowchart of this algorithm.

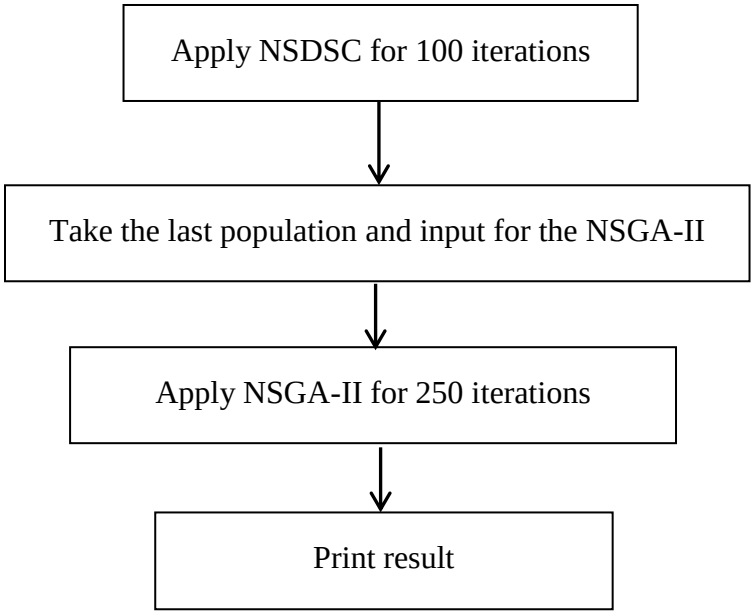

Figure 4. 1 A simple flowchart for hybrid NSDSC with NSGA-II

Procedure Hybrid NSDSC with NSGA-II

Create random solutions to build an initial two populations P, Q of size N;
Sort *P* by nondomination and build Pareto fronts;
iteration = 0;
while iteration < 100 **do**
Copy *C* times one chromosome randomly from first front and put it in first half of *P* randomly; % where *C* is $N/8$.
for $s \leftarrow 0; s < N; s \leftarrow s + 1$ **do** % N is population size, *s* is index of solution
if first quarter of population **then**
Apply dissimilarity on chromosomes ch_s, ch_{s+1} from population *P* and put the result in the offspring set *Q* respectively;
else if second quarter of population **then**
Apply similarity on chromosomes ch_s, ch_{s+1} from population *P* and put the result in the offspring set *Q* respectively*;*
else
generate randomly second half offspring set *Q;*
end for
Create a union set from population and offspring and clear population;
Build Pareto fronts pfs from union based on dominance rule;
Set the current front as $pfs[0]$*;*
while length of $population < N$do
Sort solutions from the current front by Crowding Distance;
for each solution in the ordered front **do**
if the length of $population < N$ **then**
Include the solution in population;

```
                    end if
                end for
                Go to the next Pareto front;
            end while
            iteration= iteration+1;
        end while

        Get the Pareto front as an initial population set;
        iteration =0 ;
        while iteration < 350  do
            for s←0; s < p; s ← s + 2 do
                Select two parents ch1, ch2 from population P;
                Execute crossover of ch1, ch2 and generate offspring set Q;
                Mutate ch1, ch2 from population Q;
                Evaluate and insert them in offspring set Q;
            end for
            Create a union set from population and offspring and clear population;
            Build Pareto fronts pfs from union based on dominance rule;
            Set the current front as pfs[0];
            while length of population < N do   % until last chromosome
                Sort solutions from the current front by Crowding Distance;
                for each solution in the ordered front do
                    if the length of population < N then
                        Include the solution in population;
                    end if
                end for
                Go to the next Pareto front;
            end while
            iteration= iteration+1;
        end while
end procedure
```

Algorithm 4. 1 The pseudo-code of the hybrid NS-DSC with NSGA-II

4.5 Experiment results of Hybrid NSDSC with NSGA-II algorithm

In Figures 4.2 to 4.4, we show the results of applying two algorithms (hybrid NSDSC with NSGA-II using 100 iterations and NSGA-II using 250 iterations). In this hybrid algorithm we have got good solutions for 5 problems A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4 and A9-ZDT6, that we have not got the optimal Pareto front by two previous algorithms (NSDSC, NS-DSDSC), red points represent the NSGA-II algorithm and blue points represent the Hybrid NSDSC with NSGA-II algorithm.

We observe that we have got good results also for the problem A2-ZDT1 by changing the number of iterations of NSDSC to 100 iterations and then 250 iterations of the NSGA-II algorithm. This is the least number of iterations that give good results for this function by using hybrid NSDSC with NSGA-II algorithm in 17 seconds, while in the original NSGA-II algorithm was 40 seconds for the same Pareto front results by using 500 iterations. So, we have noticed that we can change the number of iterations in our hybrid algorithm according to the type of problem. See Figure 4.3.

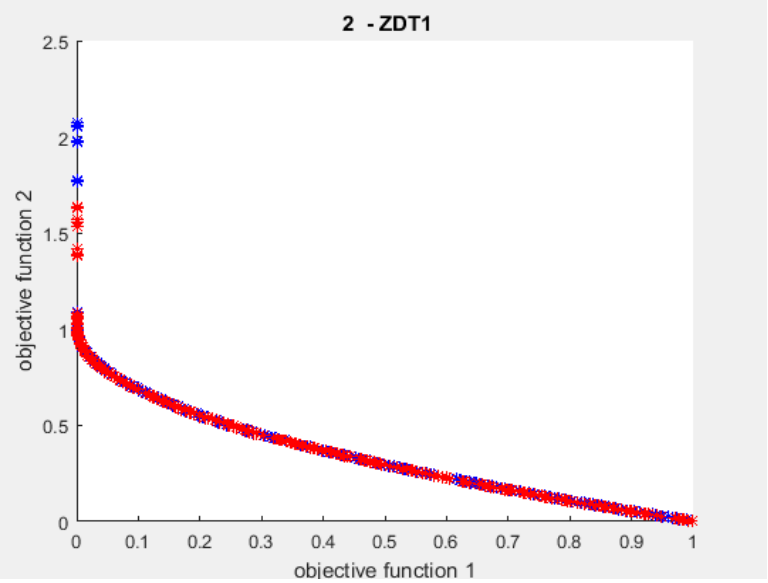

Figure 4. 2 A2-ZDT1 problem using (Hybrid NSDSC - NSGA-II)

We have observed that the results we have obtained by the (hybrid NSDSC with NSGA-II) algorithm even outperformed the results of the original algorithm NSGA-II, by using 300 iterations with NSDSC and 300 iterations of NSGA-II on problem A9-ZDT6. The illustration shows that the Hybrid NSDSC with NSGA-II results better than NSGA-II, see Figure 4.3, but the run time was longer, 14 seconds for Hybrid and 12 seconds for NSGA-II, In this special experiment where the number of iterations in NSGA-II was 500. Here we noticed the effect of first population that is got it from NSDSC and put it in NSGA-II as first population.

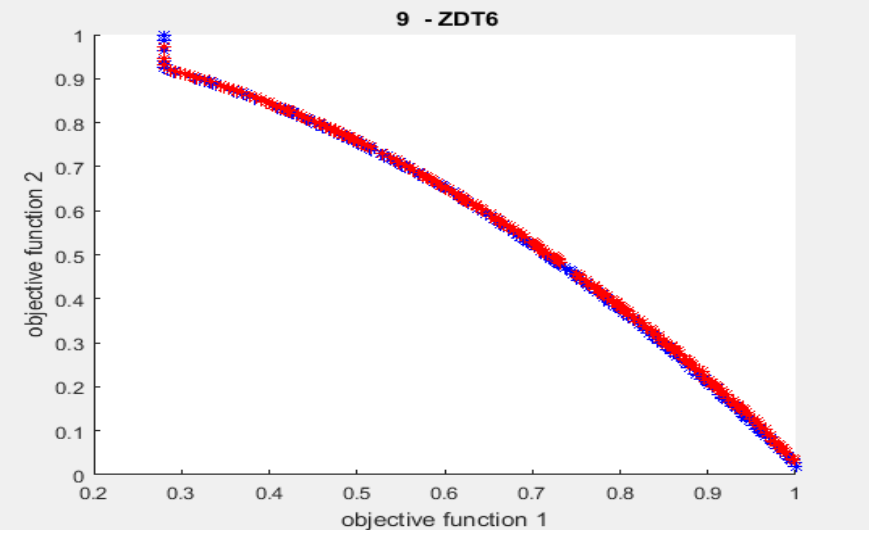

Figure 4. 3 A9-ZDT6 problem using (Hybrid NSDSC - NSGA-II)

We have noticed that the results we have obtained by the hybrid NSDSC with NSGA-II algorithm even outperformed the results of the original algorithm NSGA-II on problem A13-OSY as shown in Figure 4.5.

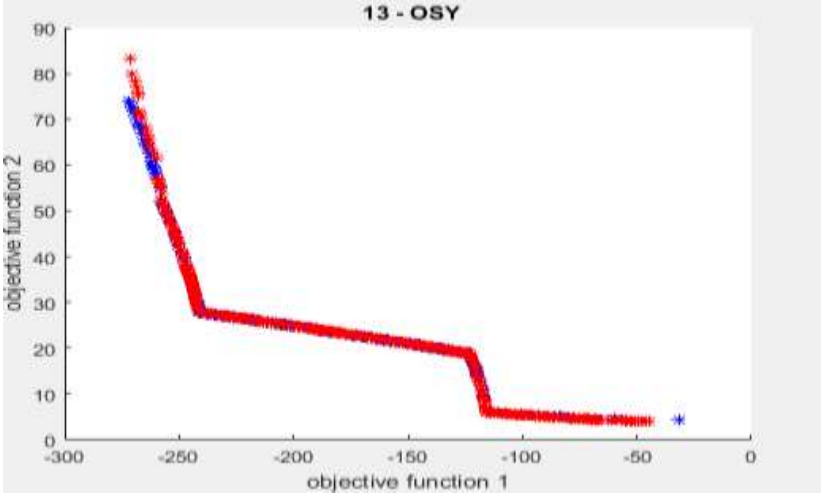

Figure 4. 4 A13-OSY problem using (Hybrid NSDSC - NSGA-II)

Table 4.1 shows the run time in seconds, and shows the superiority of the algorithm Hybrid NSDSC with NSGA-II over original algorithm NSGA-II in all 14 problems mentioned above. We have used 100 iterations for NSDSC and 350 iterations

for NSGA-II in hybrid algorithm. While we have used 500 iteration for original NSGA-II to get the results for all problems.

Table 4. 1 Comparison between Hybrid NSDSC with NSGA-II and NSGA-II, time in seconds

Function name	Hybrid NSDSC with NSGA-II Time in seconds	NSGA-II Time in seconds
A1-BNH	10.8411	17.8989
A2-ZDT1	29.8886	40.0775
A3-KUR	11.9881	19.3820
A4-SCH	11.3431	18.5226
A5-ZDT2	28.8200	43.7120
A6-FON	11.5086	19.2528
A7-ZDT3	30.0439	45.4677
A8-ZDT4	15.0783*	21.3687
A9-ZDT6	13.1751	19.8983
A10-BNH constraint	11.5523	19.0894
A11-SRN	10.4452	18.2766
A12-TNK	9.2073	17.9010
A13-OSY	50.5155	86.8131
A14-CONSTR	9.9825	16.9939

* In this problem, we increased the number of iterations for NSGA-II part to 450 iterations to get a true Pareto front.

4. 6 Hybrid First Big Population NSDSC with NSGA-II Algorithm

The initial population generation is required for starting a GA to solve an optimization problem. In general, the population size doesn't change over generations. The primary challenge with the initial population is that chromosomes created at random might not meet the problem's constraints. The original population may be insufficient to the problem, which presents another significant challenge [12].

The minimum population size should be determined according to the problem size [12].

The initial population is produced by randomly determining *p* chromosomes, where *p* is a population size [107].

Using a good initial population in evolutionary algorithms gives faster results [59]. Paying attention to the initial population may have an impact on the effectiveness of the genetic algorithm, and additional investigation and discussion are encouraged [93]. The convergence to the Pareto front is significantly sped up by a well-distributed beginning population [91].

4. 7 Methodology of Hybrid First Big Population NSDSC with NSGA-II

In this work we introduce a new method using a big initial population which is not presented in literature before. This method is named Hybrid First Big Population NSDSC with NSGA-II (Hybrid-FBP-NSDSC-NSGA-II).

Briefly, the hybridization algorithm consists of three steps to find the solution for multi-objective optimization: first we use a big initial population, the population size (Init=1000) chromosomes as initial population for NSDSC and we choose the best Z chromosomes (Z=200) as a primary population (Internal Population) for the second step, in the second step we apply the NSDSC algorithm, in the third step we apply the NSGA-II algorithm.

In this new method, we first take (Init=1000) big population with applying NSDSC one time, the NSDSC that mentioned in Section 3.4, then take the best 200

chromosomes from the best Pareto fronts as population size for second step, in step two we apply NSDSC algorithm for N iterations (N=50), we take then the last generation as an input for step three, in step three we apply NSGA-II for M iterations (M=350). The result was faster than previous work in most functions except for problem P8-ZDT4. But with increase of the first big population to (Init=2000), then (N=50) iterations in NSDSC, then (M=450) iterations in NSGA-II, we get a very good solution for function 8.

The pseudo-code of this algorithm the same as the previous one, only we start with a big initial population.

Procedure Hybrid First Big initial pop NSDSC with NSGA-II

Step1:

Generate the big initial population (Init=1000) **{Big Pop NSDSC}** % sorted by non-dominated sorting

Iteration = 1;

Create solutions to build an initial two populations P, Q of size N from first fronts taken from big population.

Sort *P* by nondomination and build Pareto fronts;

while iteration <= 1 **do** **{NSDSC one time }**

Copy *C* times one chromosome randomly from first front and put it in first half of P randomly; % where *C* equals $N/8$, *N* is the number of chromosomes in the population.

for $s \leftarrow 0; s < p; s \leftarrow s + 1$ **do** % p is population, s is index of solution

if first quarter of population **then**

Apply dissimilarity on chromosomes ch_s, ch_{s+1} from population *P* and put the result in the offspring set *Q*;

else if second quarter of population **then**

Apply similarity on chromosomes ch_s, ch_{s+1} from population *P* and put the result in the offspring set *Q;*

else generate randomly second half offspring set *Q;*

end for

Create a union set from population and offspring and clear population;

Build Pareto fronts pfs from union based on dominance rule;

Set the current front as $pfs[0]$*;*

while length of $population < N$ do % until last chromosome

Sort solutions from the current front by Crowding Distance;

for each solution in the ordered front **do**

if the length of $population < N$ **then**

Include the solution in population;

end if

end for

Go to the next Pareto front;

end while
iteration= iteration+1;
end while

Step 2: take the best (Z=200) chromosomes from step 1 as an initial population set;
Iteration = 0; **{NSDSC}**
while iteration <= 50 **do**
Copy C times first chromosome and put it in first half of P randomly;
for $s \leftarrow 0; s < p; s \leftarrow s + 1$ **do**
if first quarter of population **then**
Apply dissimilarity chromosomes ch_s, ch_{s+1} from population P and put the result in the offspring set Q;
else if second quarter of population **then**
Apply similarity on chromosomes ch_s, ch_{s+1} from population P and put the result in the offspring set Q;
else generate randomly second half offspring set Q;
end for
Create a union set from population and offspring and clear population;
Build Pareto fronts pfs from union based on dominance rule;
Set the current front as $pfs[0]$;
while length of $population < N$ do
Sort solutions from the current front by Crowding Distance;
for each solution in the ordered front **do**
if the length of $population < N$ **then**
Include the solution in population;
end if
end for
Go to the next Pareto front;
end while
iteration= iteration+1;
end while

Step 3: take the last population from step 2 as an initial population set;
Iteration = 0;
while iteration < 350 **do** **{NSGA-II}**
for $s \leftarrow 0; s < p; s \leftarrow s + 2$ **do**
Select two parents ch_1, ch_2 from population P;
Execute crossover of ch_1, ch_2 and generate offspring set Q;
Mutate $ch_1, ch_2\ f$ from population Q;
Evaluate and insert them in offspring set Q;
end for
Create a union set from population and offspring and clear population;
Build Pareto fronts pfs from union based on dominance rule;
Set the current front as $pfs[0]$;
while length of $population < N$ do
Sort solutions from the current front by Crowding Distance;

for each solution in the ordered front **do**
if the length of $population < N$ **then**
Include the solution in population;
end if
end for
Go to the next Pareto front;
end while
Iteration = Iteration +1;
end while
end procedure

Algorithm 4. 2 The pseudo-code of Hybrid First Big initial pop NSDSC with NSGA-II

4.8 Experiment results of hybrid first big population NSDSC with NSGA-II algorithm

In this section will describe the results that we have got by applying big initial population plus the previous hybrid algorithm, to see the effect of big initial population on 14 test problems. In Figures 4.5 to 4.7 we got very good Pareto result for problem A2-ZDT1, A5-ZDT2 and A7-ZDT3 that have 30 variables, while in previous algorithms (NSDSC, NS-DSDSC) we have not reached to the Pareto optimum front.

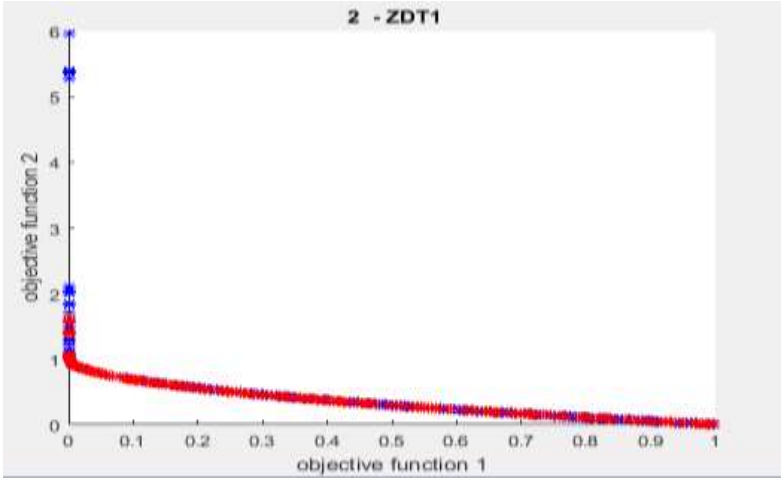

Figure 4. 5 The A2-ZDT1 problem by using Hybrid-FBP-NSDSC-NSGA-II compared with NSGA-II

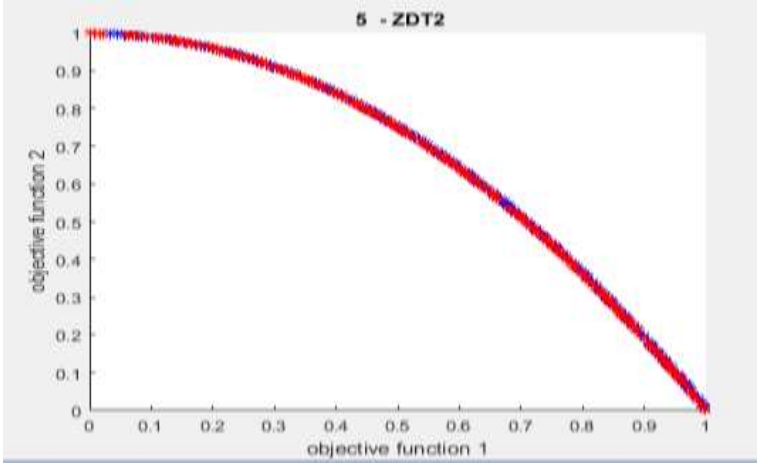

Figure 4. 4 The A5-ZDT2 problem by Hybrid-FBP-NSDSC-NGSA-II compared with NSGA-II

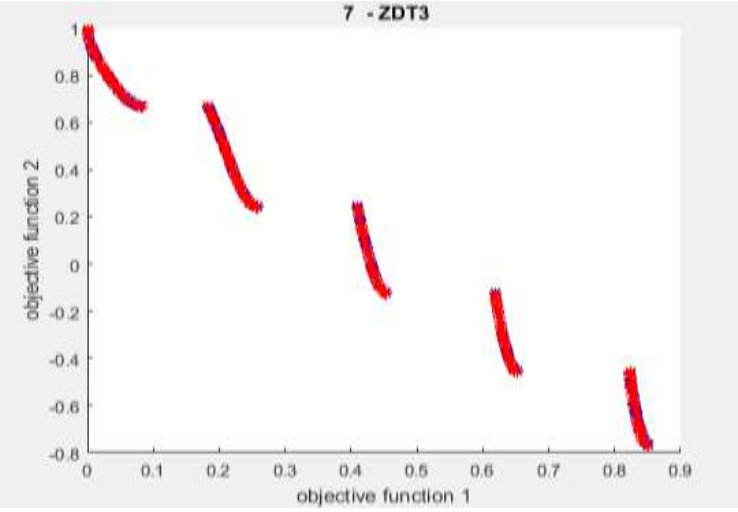

Figure 4. 5 The A7-ZDT3 problem by using Hybrid-FBP-NSDSC-NGSA-II compared with NSGA-II

We have noticed that for problem P8-ZDT4 we have not reached to optimal solutions when Init=1000, 50 iterations in NSDSC-II, 350 iterations in NSGA-II, as see Figure 4. 8. But, when we increase the size of a big population and the number of iterations (big pop=2000, 50 iterations in NSDSC, 450 iterations in NSGA-II) it is a little better than NSGA-II see Figure 4.9, (blue points represent Big Pop with hybrid NSDSC with NSGA-II, while red points represent NSGA-II), we notice that the NSGA-II did not reached the optimum Pareto front in problem A8-ZDT4 as shown in Figure 4.9 where we used 500 iterations for NSGA-II.

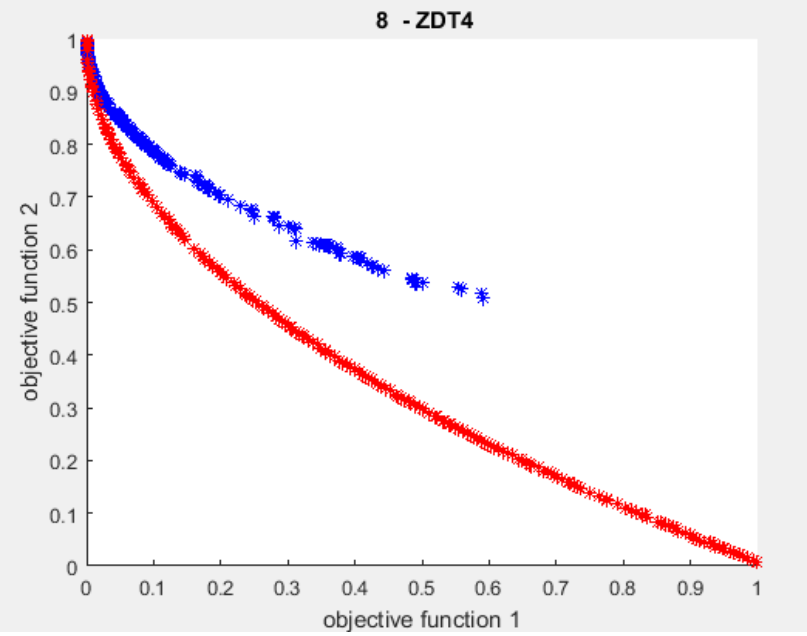

Figure 4. 6 The A8-ZDT4 problem by using Hybrid-FBP-NSDSC-NSGA-II compared with NSGA-II

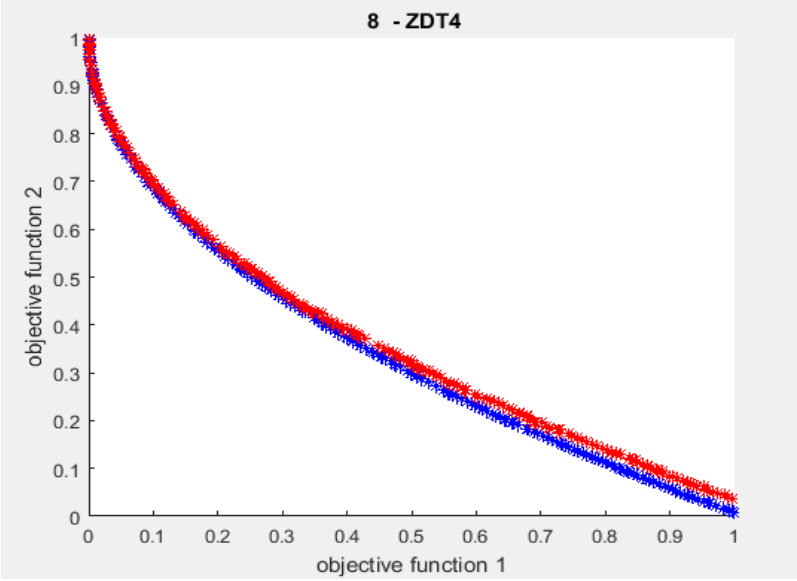

Figure 4. 7 The A8-ZDT4 problem by using Hybrid-FBP-NSDSC-NSGA-II compared with NSGA-II after increasing big pop, init=2000, 50 iterations in NSDSC, 450 iterations in NSGA-II

In Table 4. 2 we compared NSGA-II and Big Population Hybrid NSDSC with NSGA-II in run time, the results show that the Big Population Hybrid NSDSC with NSGA-II is better than NSGA-II in all 14 problems. For problem A8-ZDT4 we don't have reached the optimal solutions (not pass), but when we made new different experiments with different parameters for example as follow: (big pop=2000, iterations in NSDSC=50 , iterations in NSGA-II= 450) we have reached the Pareto optimal solutions in 19.478 seconds.

Table 4. 2 hybrid first big population NSDSC with NSGA-II time in seconds

Function name	1000 big pop one time, 50 iteration NSDSC, 350 iterations NSGA-II	NSGA-II Time in seconds 500 iteration
A1-BNH	9.4692	17.8989
A2-ZDT1	25.543	42.276
A3-KUR	11.3808	19.3820
A4-SCH	10.8477	18.5226
A5-ZDT2	25.0906	43.7120
A6-FON	10.9543	19.2528
A7-ZDT3	26.2437	45.4677
A8-ZDT4	11.9668 not pass*	21.3687
A9-ZDT6	10.5244	19.8983
A10-BNH constraint	9.5766	19.0894
A11-SRN	9.7232	18.2766
A12-TNK	9.3709	17.9010
A13-OSY	47.8866	86.8131
A14-CONSTR	9.3573	16.9939

* After increasing the number of iterations (big pop=2000, 50 iterations in NSDSC, 450 iterations in NSGA-II) we have reached the Pareto optimal solutions in 19.478 seconds.

4.6 Conclusion

In this chapter, we have noticed that the use of hybrid algorithms gives us better results than the two previous algorithms NSDSC and NS-DSDSC, when we tested them on 14 problems mentioned above, especially with the problems A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4 and A9-ZDT6 for which the NSDSC and NS-DSDSC algorithms have founded difficulties to reach optimal solutions.

In particular, the problem A8-ZDT4 doesn't pass, but when we increase the number of iterations (big pop=2000, iterations in NSDSC=50 , iterations in NSGA-II= 450) we get better results than NSGA-II with 500 iteration. Also, run times in the hybrid algorithm are much better than for NSDSC, NS-DSDSC and NSGA-II.

CHAPTER FIVE: Multitasking on MOO

5.1 Introduction

In this chapter we introduce a new algorithm called Multitasking on Multi-Objective Optimization (MMOO) and we show that the results are satisfactory compared to the algorithms in the previous chapters (NSDSC, NS-DSDSC, Hybrid NSDSC-NSGA-II, Hybrid-FBP-NSDSC-NSGA-II), for all 14 problems.

In the last part of this thesis, we present an examination of the algorithms mentioned in the third and fourth chapters, as well as the algorithm in this chapter on all 14 problems by the seven measures and compare our algorithms with the NSGA-II algorithm.

5.2 Multitasking background

In [3], for the purpose of resolving Constrained Multi-objective Optimization Problems (CMOPs), an Evolutionary Multitasking-based Constrained Multi-objective Optimization (EMCMO) framework is created. In EMCMO, a CMOP optimization is split into two tasks that are related to one another: one task is for the original CMOP, while the other task just considers the objectives while ignoring all constraints. The major goal of the second task is to help solve CMOP by providing reasonable knowledge about the first task's goals.

In [10] numerous potential benefits of implicit genetic transfer in a multitasking setting are revealed by the numerical studies. Most importantly, the authors show that for a range of difficult optimization functions, the creation and transfer of refined genetic material frequently results in rapid convergence.

The primary goal of [10] that introduce Multi Factorial Optimization (MFO) as a novel idea in optimization and evolutionary computation that is distinct from the concepts of SOO and MOO. The next step is to create a generic EA for MFO that can multitask across many optimization problems in the most effective way feasible.

For instance, while an individual's ability to learn may be influenced by their genotype, spreading of a cultural feature may produce or modify the selection operating on their genetic system [108], [10]. These cultural features frequently result from social learning or from parents teaching their children particular traditions and preferences. For the aim of effective evolutionary multitasking, the computational equivalent of multifactorial inheritance is provided by assuming that each optimization task contributes a different environment in which offspring can be raised.

As an example of multitasking [109] has used the binary clustering approach, where the population is separated into various classes, and the representative point for each class is computed. Then, a powerful method for generating multiple prediction directions by representative points is developed. Following that, a mutation strength adaptation strategy is suggested in accordance with the level of improvement for each class. Finally, the prediction directions and mutation strengths generate the predictive transferred solutions as transfer knowledge.

5.3 The Multitasking on MOO algorithm

Consider a multi-objective problem with two objectives, for example, one of the test problems listed in Appendix A. We apply the evolutionary multitasking method to this problem, where one task is to solve the original multiobjective problem, and the other two tasks are for minimizing two objective functions separately. In this way, we obtain three different solutions: one for the multiobjective problem, and the other two are the minimum points of the two objective functions, respectively.

The initial population may be disproportionate to the problem, that may delay reaching to optimum solution for the problem [12]. So, in our work we try to enhance all population by using a new idea in multitasking on MOO that put the best 6 chromosomes which taken from single objective functions in specific position in population, also, a random part in other specific positions, these some populations were represented the first 20% of generations.

Then we compare the results of the multitasking method with the results obtained by applying some multiobjective optimization algorithms (for example, NSDSC, NS-DSDSC, Hybrid NSDSC with NSGA-II, Hybrid first big pop NSDSC with NSGA-II, NSGA-II) to determine whether or not multitasking really helps to solve the multiobjective problem. Of course, the same can be done for more than two objectives.

Suppose that we have given the following two functions:

$$f_j: \mathrm{X} \longrightarrow \mathbb{R}, j = 1, 2,$$

where

$$X = \prod_{k=1}^{n} [\mathrm{a}_k, \mathrm{b}_k] \subset \mathbb{R}^n.$$

We consider the following two-objective minimization problem:

$$\text{Minimize } (f_1(x), f_2(x)) \text{ subject to } x \in X.$$

where:

$f_1(x)$ is the first objective function to be minimized,

$f_2(x)$ is the second objective function to be minimized.

The variable *x* represents a vector of decision variables, and the problem is to find the optimal *x*'s that give a Pareto optimum to this problem.

In our multitasking method, we use the following three tasks:

Task T_1: minimize f_1 on X, in this task we minimize the first objective function (f_1) as a single objective.

Task T_2: minimize f_2 on X, in this task we minimize the second objective function (f_2) as a single objective.

Task T_3: minimize $f := (f_1, f_2)$ on X (in the Pareto sense), in this task we apply a multiobjective optimization on (f_1 ,f_2).

The details of the Multitasking on MOO algorithm are explained in Algorithm 5.1 below. In our algorithm, we use one population for all three tasks.

Procedure Multitasking on MOO

Generate the initial population randomly

While (stopping criterion is not satisfied) do

If (number of iteration < 20% of max number of generations) then

do the tasks T_j , $j \in \{1,2,3\}$, according to the following steps:

1. For $j = 1, 2$ take the indexes of best six chromosomes (best three values for f_1 and f_2) from list of chromosomes that sorted in ascending order with respect to f_j.
2. Insert the best six chromosomes for f_1 and f_2 and put them in the first quarter of population in fixed locations (10, 15, 20, 25, 30, 35).
3. Generate randomly 20 chromosomes (10% of population) and insert them in the second quarter of population.
4. For $j = 3$, apply NSGA-II,

Else

For the rest of generations (80%) apply only the normal NSGA-II

End if

End

Algorithm 5. 1 The Multitasking on MOO algorithm

5.4 Experimental results of Multitasking of MOO Algorithm

We have used the Multitasking on MOO algorithm and applied it to the problems A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4 and A9-ZDT6, where we have not reached the optimal solutions using NSDSC and NS-DSDSC algorithms. We note that this Multitasking on MOO algorithm has reached the optimal solutions in a similar way as the NSGA-II algorithm. It also appears from the experimental results that the Multitasking on MOO algorithm is even better than the hybrid algorithms in the previous chapter, because the hybrid algorithms don't reach the Pareto front until we increase the number of iterations to 450 in NSGA-II with the eighth problem.

In this work, optimization is performed in MATLAB environment using the code from [110]. In this code we made some changes to suit the problems and algorithms that have been used in this thesis, with fourteen problems with and without constraints.

In the Figures below from Figure 5. 1 to Figure 5. 6, we note that the Multitasking on MOO algorithm has achieved good results in problems A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4 and A9-ZDT6 that the previous algorithms could not find solutions for.

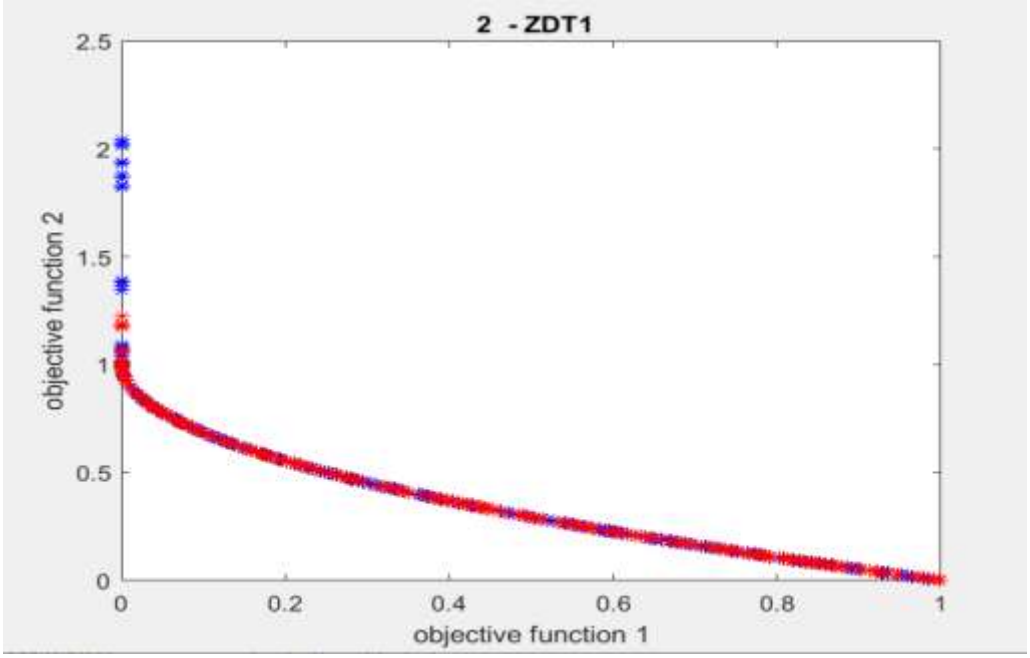

Figure 5. 1 The A2-ZDT1 problem by using Multitasking on MOO

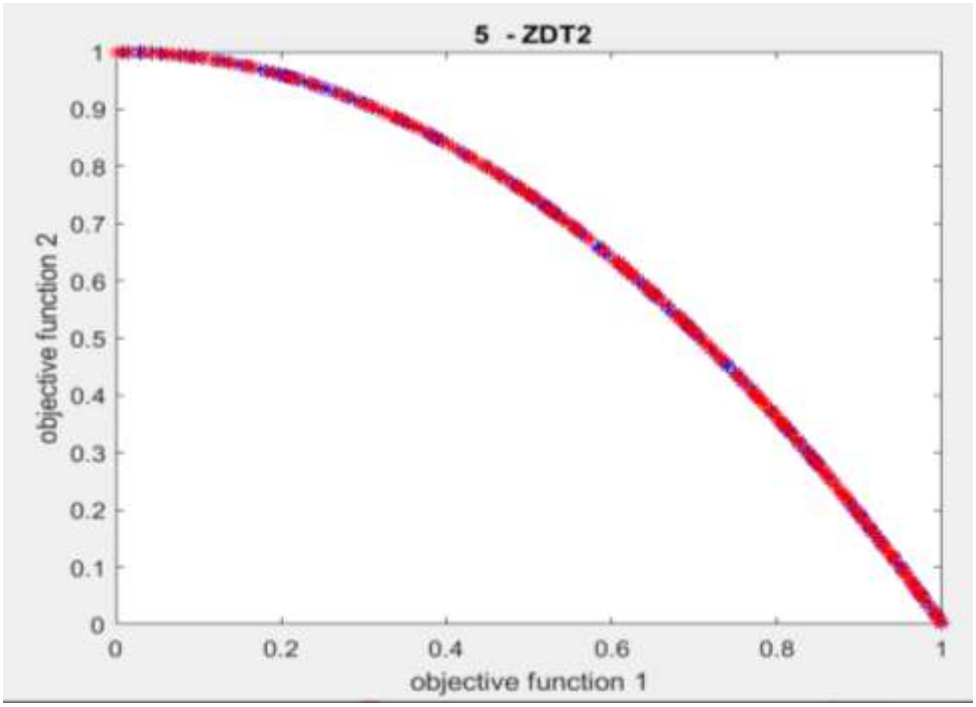

Figure 5. 2 The A5-ZDT2 problem by Multitasking on MOO

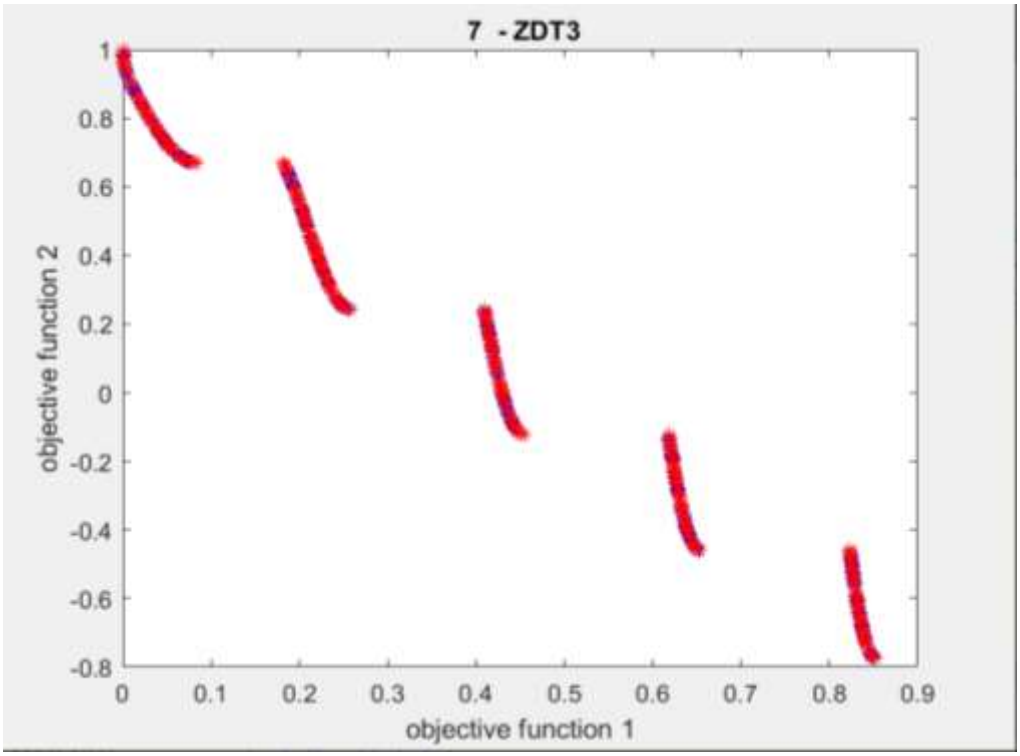

Figure 5. 3The A7-ZDT3 problem by using Multitasking on MOO

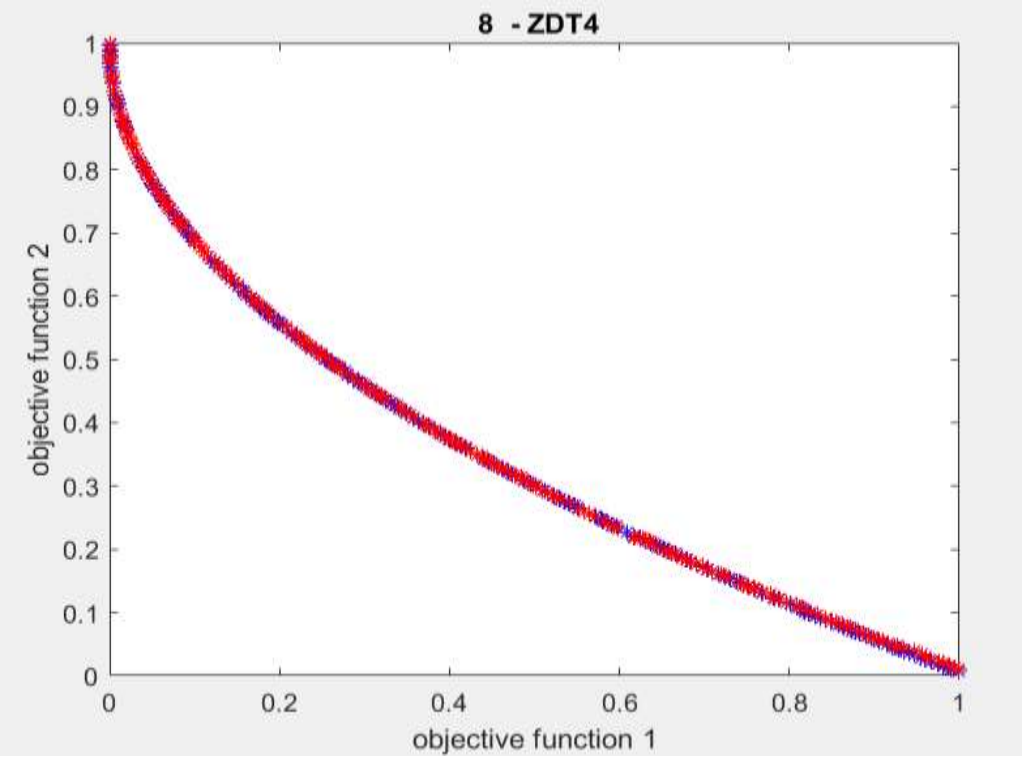

Figure 5. 4The A8-ZDT4 problem by using Multitasking on MOO

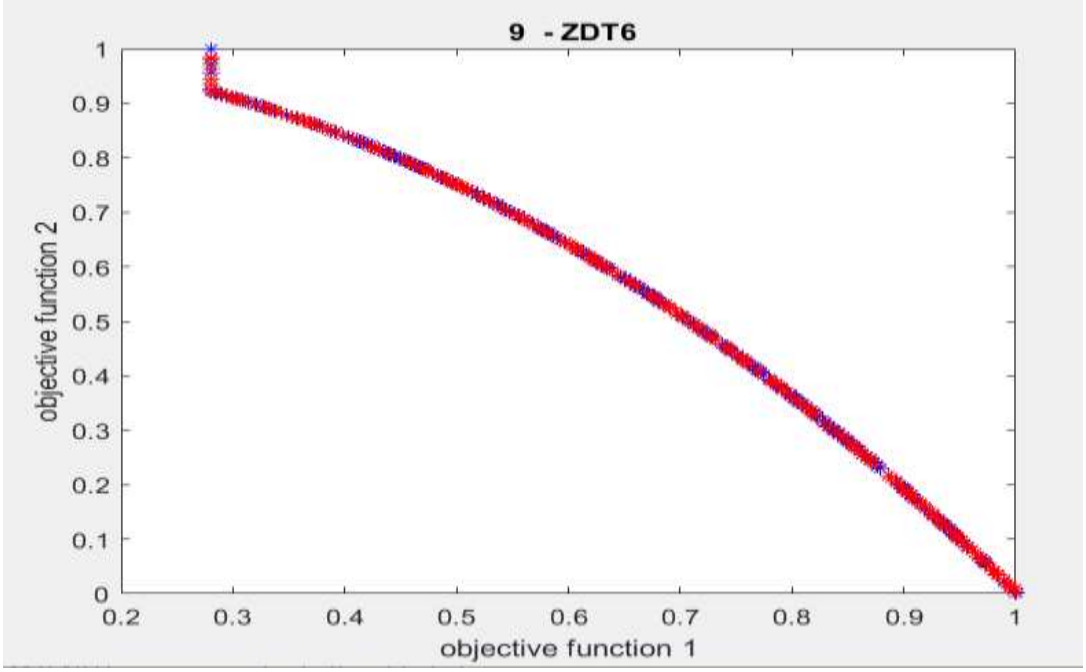

Figure 5. 5 A9-ZDT6 problem using Multitasking on MOO

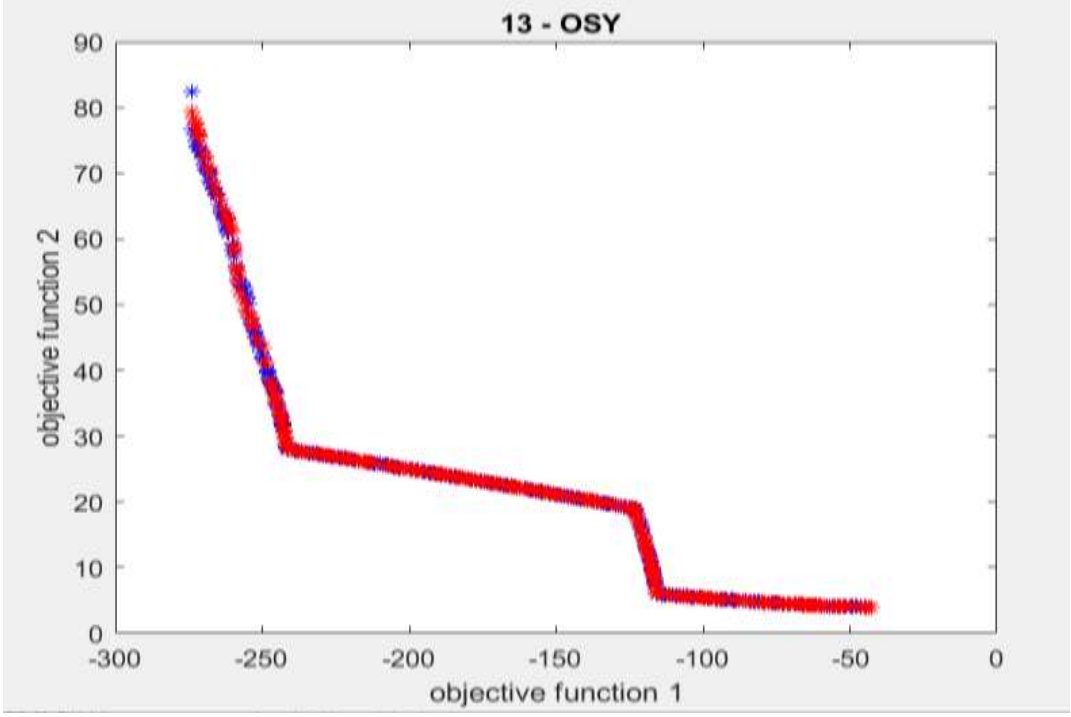

Figure 5. 6 A13-OSY problem using Multitasking on MOO

Table 5. 1 shows a comparison of the run time between the Multitasking on MOO algorithm and the NSGA-II algorithm. We note that the run time for the Multitasking on MOO algorithm is less than for the NSGA-II, which is due to the smaller number of iterations. However, when we limited the number of iterations to 500 only for Multitasking on MOO, we got the same results of the Pareto front as with 1000 iterations for NSGA-II (see Figures 5.1 – 5.6).

Table 5. 1 Comparison of Multitasking on MOO and NSGA-II in run time

Function name	**Multitasking Time in seconds 500 itr.**	**NSGA-II Time in seconds 1000 itr.**
A1-BNH	20.3891	36.8989
A2-ZDT1	43.3132	48.0775
A3-KUR	21.1785	45.3970
A4-SCH	16.7288	40.6266
A5-ZDT2	41.7142	48.7190
A6-FON	17.9614	44.0588
A7-ZDT3	43.4190	46.5388
A8-ZDT4	21.6583	42.6555
A9-ZDT6	18.0601	43.2656
A10-BNH constraint	20.9853	37.0089
A11-SRN	17.5782	38.9880
A12-TNK	17.3287	36.6221
A13-OSY	89.2943	173.8189
A14-CONSTR	18.1737	35.2914

We have also tested 14 problems by using 50 iterations for the Multitasking on MOO algorithm and 50 iterations for the NSGA-II algorithm. We observed that Multitasking on MOO is, in most cases, superior than NSGA-II, and achieves the Pareto front faster, because it has

a positive impact on the initial generations. Figures 5.7 – 5.12 show some of test problems, where the blue points represent Multitasking on MOO and red points represent NSGA-II.

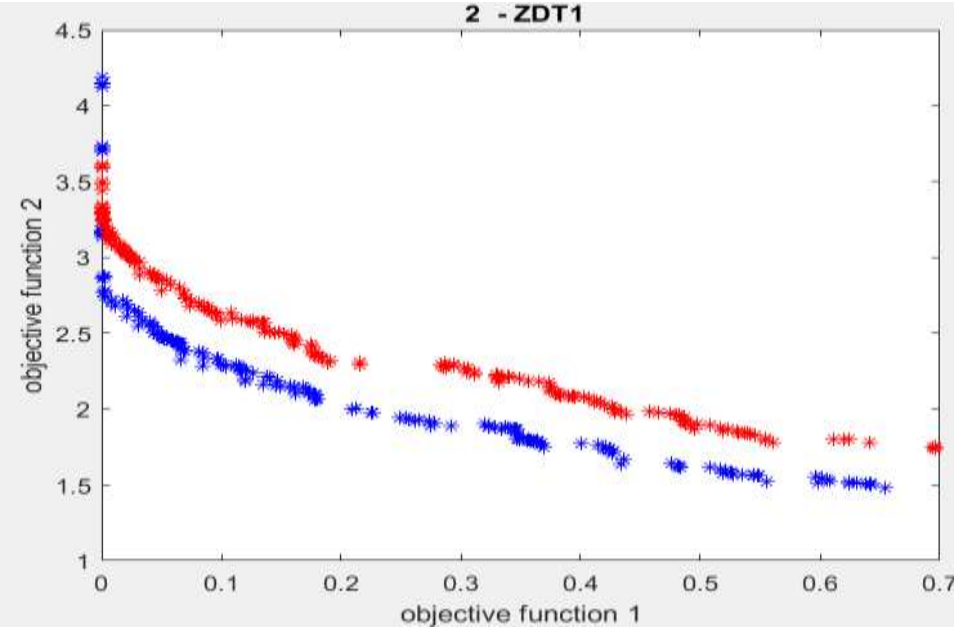

Figure 5. 7 A2-ZDT1 problem using Multitasking on MOO and NSGA-II with 50 iterations for each algorithm

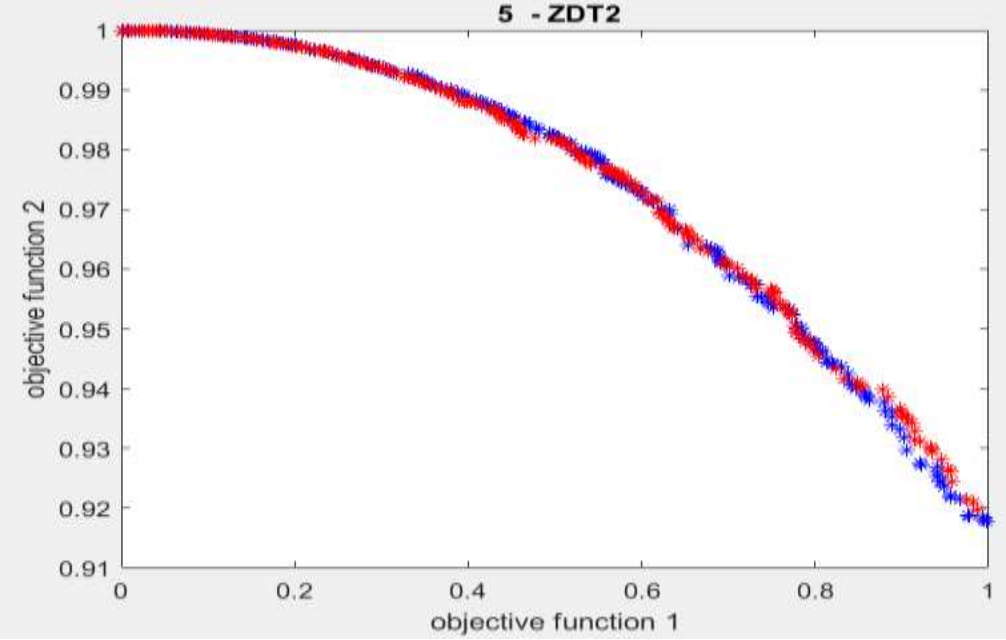

Figure 5. 8 A5-ZDT2 problem using Multitasking on MOO and NSGA-II with 50 iterations for each algorithm

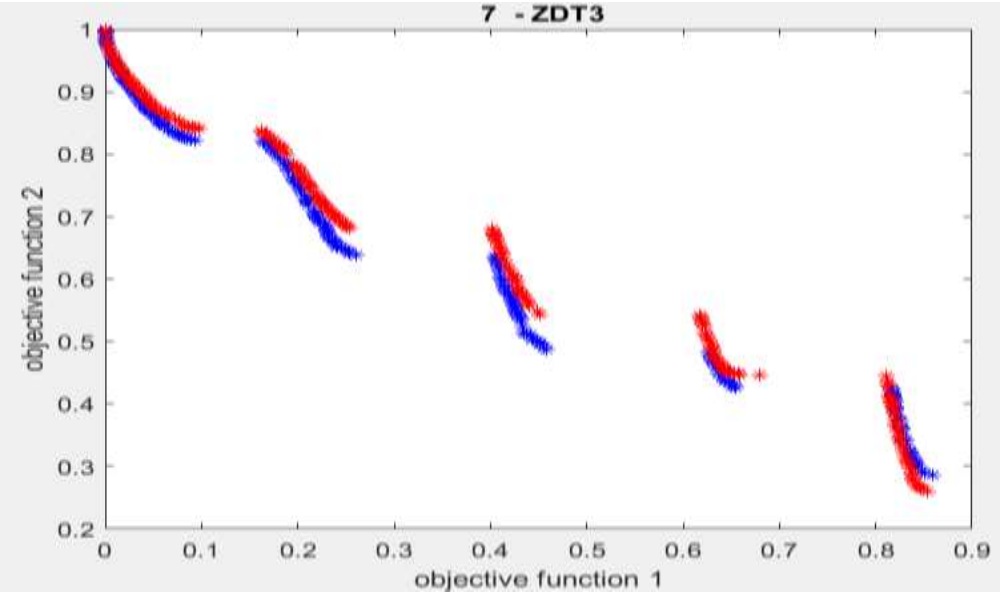

Figure 5. 9 A7-ZDT3 problem using Multitasking on MOO and NSGA-II with 50 iteration for each algorithm

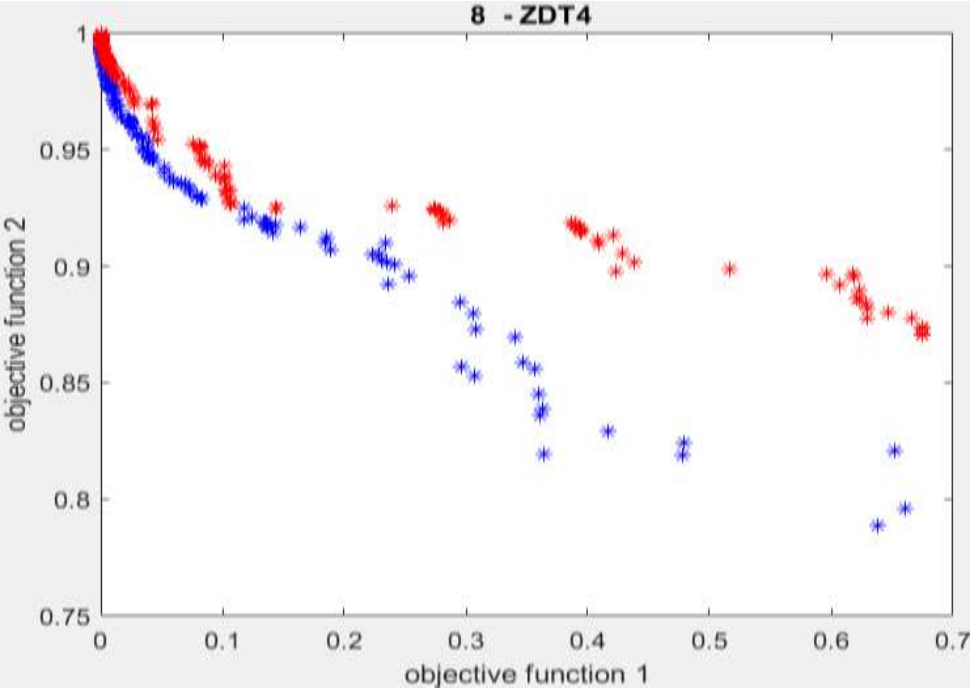

Figure 5. 10 A8-ZDT4 problem using Multitasking on MOO and NSGA-II with 50 iterations for each algorithm

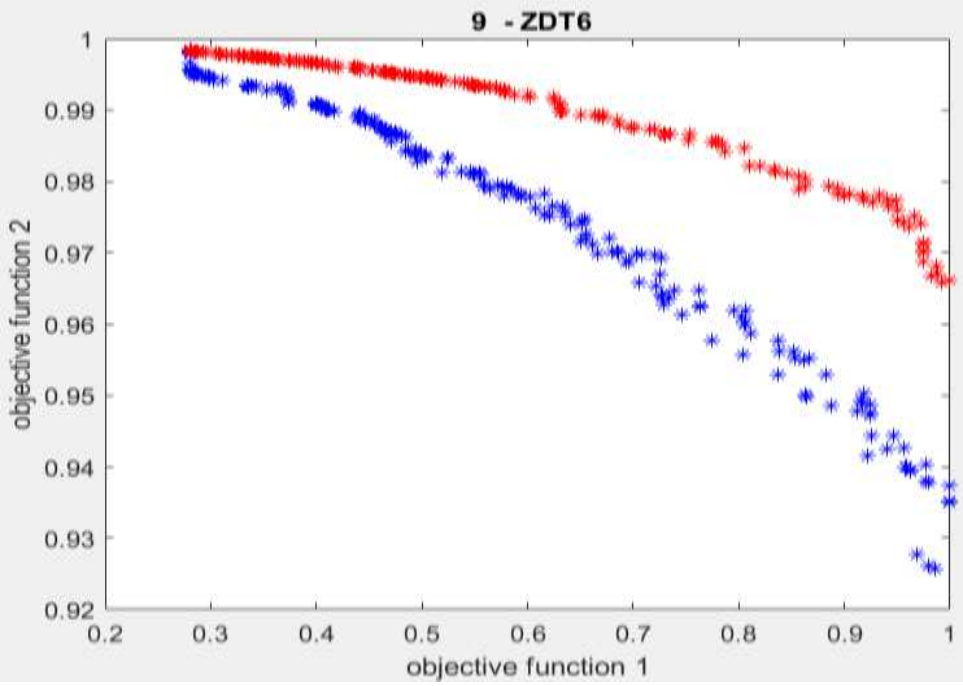

Figure 5. 11 A9-ZDT6 problem using Multitasking on MOO and NSGA-II with 50 iteration for each algorithm

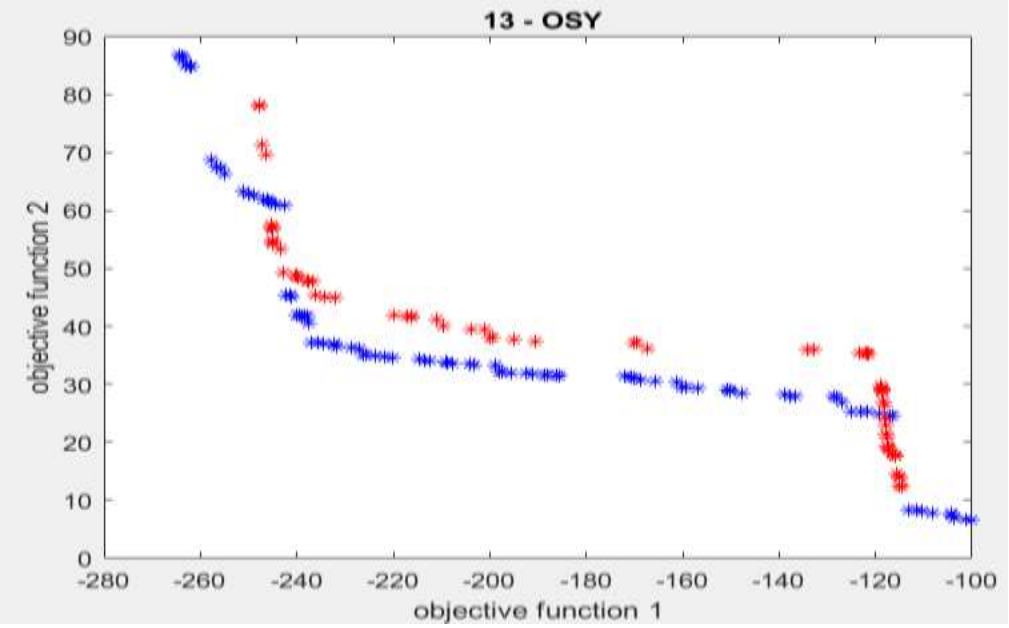

Figure 5. 12 A13-OSY problem using Multitasking on MOO and NSGA-II with 50 iteration for each algorithm

Tables 5. 2 shows comparison of all algorithms: NSGA-II, NSDSC, NS-DSDSC, Hybrid NSDSC with NSGA-II, Big Population Hybrid NSDSC with NSGA-II and Multitasking on MOO in run time (time in seconds). This table illustrates that our algorithms are better than NSGA-II in run time with the same number of 1000 iterations for NSDSC and NS-DSDSC, except some problems that NSDSC and NS-DSDSC could not solve. We noticed that the best run time is for Big population Hybrid NSDSC with NSGA-II algorithm with total number of iterations equal to 400, but only the problem A8-ZDT4 has not reached Pareto front, on the other hand, for the same problem (A8-ZDT4) when we increase the number of iterations in NSGA-II to 450 the algorithm will reach to Pareto front. The reference set was taken form NSGA-II for all metrics (IGD, GD, HV, Spacing Metric , Delta P, Spread metric).

Table 5. 2 Comparison of all algorithms: NSGA-II, NSDSC, NS-DSDSC, Hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO in run time

A1-BNH	17.8989	24.1824	31.8522	10.8411	9.4692	20.3891
A2-ZDT1	42.276	42.0908*	43.5415*	29.8886	25.543	43.3132
A3-KUR	19.3820	27.9392	33.5209	11.9881	11.3808	21.1785
A4-SCH	18.5226	24.3879	28.5681	11.3431	10.8477	16.7288
A5-ZDT2	43.7120	40.1732*	38.8339*	28.8200	25.0906	41.7142
A6-FON	19.2528	27.1879	32.5627	11.5086	10.9543	17.9614
A7-ZDT3	45.4677	40.7832*	32.4316*	30.0439	26.2437	43.4190
A8-ZDT4	21.3687	28.2977*	32.2049*	15.0783*	11.9668 **	21.6583
A9-ZDT6	19.8983	30.1160*	34.7615*	13.1751	10.5244	18.0601
A10-BNH constraint	19.0894	29.1140	30.5356	11.5523	9.5766	20.9853
A11-SRN	18.2766	25.3399	23.5226	10.4452	9.7232	17.5782
A12-TNK	17.9010	25.1719	23.6143	9.2073	9.3709	17.3287
A13-OSY	86.8131	125.9281	254.1377	50.5155	47.8866	89.2943
A14-CONSTR	16.9939	26.2113	29.3958	9.9825	9.3573	18.1737

* Not pass to the Pareto front.

** If we increase the number of iterations to 450 in NSGA-II, the algorithm will reach to the optimum solutions as shown in Figure 4.10 Blue points.

Table 5. 3 shows the results of all Algorithms, NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using IGD metric on tested problems

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Big_pop+ Hybrid NSDSC with NSGA-II	Multitasking on MOO
A1-BNH	0.218597	0.33696	0.3629	0.34194	0.32173	0.26935
A2-ZDT1	0.0040749	0.11187	0.075271	0.012054	0.01721	0.017231
A3-KUR	0.020047	0.0714	0.025259	0.02083	0.020674	0.019964
A4-SCH	0.0082489	0.01038	0.010953	0.0081676	0.0094426	0.0084038
A5-ZDT2	0.0025518	0.35667	0.38423	0.0025745	0.0065798	0.002325
A6-FON	0.0025878	0.0025047	0.0028267	0.0026423	0.0025123	0.0027971
A7-ZDT3	0.002765	0.26306	0.1088	0.0034024	0.0045244	0.0026206
A8-ZDT4	0.002412	0.50553	0.50219	0.14408	0.13818	0.0033605
A9-ZDT6	0.0030738	0.34736	0.40755	0.0070988	0.0027567	0.0022695
A10-BNH constraint	0.27149	0.23217	0.2625	0.24374	0.26853	0.26796
A11-SRN	0.54692	0.49751	0.54241	0.56506	0.52795	0.5286
A12-TNK	0.001971	0.0049136	0.0074075	0.0020734	0.0021483	0.0020745
A13-OSY	0.3393	1.5682	1.9782	5.3141	1.9908	0.70325
A14-CONSTR	0.011618	0.01357	0.013106	0.011887	0.013729	0.012022

5.5 Experiment Results of Performance Measures

In this section we present experimental results of performance for all algorithms (NSGA-II, NSDSC, NS-DSDSC, Hybrid NSDSC with NSGA-II, Big population with Hybrid NSDSC with NSGA-II and Multitasking on MOO) on 14 test multi-objective problems, we used seven performance metrics (IGD, GD, HV, Spacing, Spread, DeltaP

and $s(x)$ metric), six of them were mentioned in Section 2.5, and the $s(x)$ metric will be described in Section 5.6. For more details see [111].

In Tables 5.3 - 5.8 we show the six performance metrics (IGD, GD, HV, Spacing metric, Spread metric, DeltaP metric), computed for the results of our proposed algorithms, where we took the NSGA-II algorithm and considered it the true Pareto front solutions to find these metrics. In Figures 5.13 - 5.18 we show the performance metrics (IGD, GD, HV, Spacing metric, Spread metric, DeltaP metric) for the same results as in Tables, respectively.

We observe that the lower value of IGD, GD and DeltaP metrics means the higher efficiency of the algorithm, while the higher value of HV, Spacing and Spread metrics also means the higher efficiency of the algorithm.

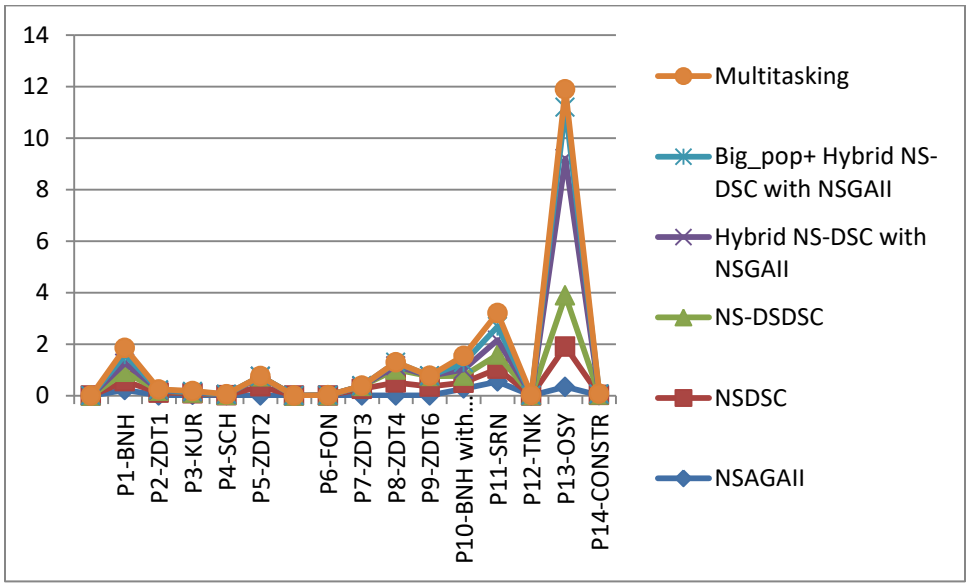

Figure 5. 13 shows the results of all Algorithms, NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using IGD metric on tested problems

Table 5. 4 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big Population Hybrid NSDSC with NSGA-II and Multitasking on MOO using GD metric on tested problems

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Big_pop+ Hybrid NSDSC with NSGA-II	Multitasking On MOO
A1-BNH	0.037062	0.033173	0.032214	0.031276	0.029445	0.029545
A2-ZDT1	0.004409	0.0072457	0.0029545	0.0073475	0.051861	0.00034191
A3-KUR	0.001631	0.0045643	0.0020911	0.0017161	0.0016587	0.0015548
A4-SCH	0.000720	0.0007688	0.0007900	0.0007182	0.0007753	0.0007663
A5-ZDT2	0.0002909	0.0027252	0.0037056	0.0002451	0.0005008	0.00022645
A6-FON	0.0002191	0.0002180	0.0002342	0.0002141	0.0002148	0.00021881
A7-ZDT3	0.0002865	0.0030255	0.001904	0.0003118	0.0003513	0.00027506
A8-ZDT4	0.0001973	0.010681	0.0067269	0.0039666	0.0042	0.00027216
A9-ZDT6	0.00029559	0.016413	0.019825	0.0005283	0.0002162	0.00021883
A10-BNH constraint	0.025267	0.023411	0.025843	0.023925	0.024158	0.024976
A11-SRN	0.045413	0.046442	0.046125	0.047245	0.047789	0.047981
A12-TNK	0.00016868	0.00024073	0.00023633	0.00017234	0.00018145	0.00017391
A13-OSY	0.061796	0.18696	0.11466	0.047288	0.02756	0.12148
A14-CONSTR	0.0011564	0.0012272	0.0012166	0.0011821	0.0013074	0.0012005

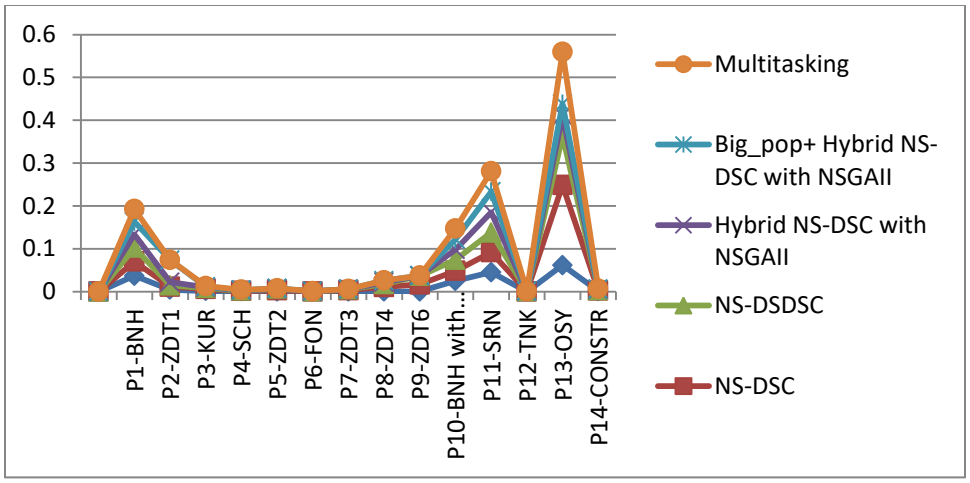

Figure 5. 14 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using GD metric on tested problems

Table 5. 5 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using HV metric on tested problems

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Big_pop+ Hybrid NSDSC with NSGA-II	Multitasking on MOO
A1-BNH	0.86009	0.86023	0.86008	0.86016	0.86011	0.86090
A2-ZDT1	0. 84748	0.77363	0.79545	0.84582	0.83801	0.84747
A3-KUR	0.53571	0.53862	0.53592	0.53571	0.53581	0.5358
A4-SCH	0.86072	0.86062	0.8604	0.86075	0.8608	0.86081
A5-ZDT2	0.44609	0.14208	0.12167	0.44462	0.43763	0.44617
A6-FON	0.43220	0.43198	0.43183	0.43231	0.43234	0.43241
A7-ZDT3	0.60027	0.63031	0.66268	0.59937	0.60014	0.60031

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Big_pop+ Hybrid NSDSC with NSGA-II	Multitasking on MOO
A8-ZDT4	0.72084	0.17049	0.17743	0.56039	0.56044	0.71761
A9-ZDT6	0.41151	0.13069	0.093806	0.3986	0.40509	0.40595
A10-BNH constraint	0.78474	0.78492	0.78471	0.78493	0.78468	0.78485
A11-SRN	0.59342	0.59407	0.59396	0.59344	0.5935	0.59362
A12-TNK	0.40072	0.396	0.39649	0.40061	0.40059	0.40072
A13-OSY	0.76391	0.74342	0.76781	0.81581	0.79849	0.76018
A14-CONSTR	0.48166	0.48161	0.4816	0.48161	0.48164	0.48166

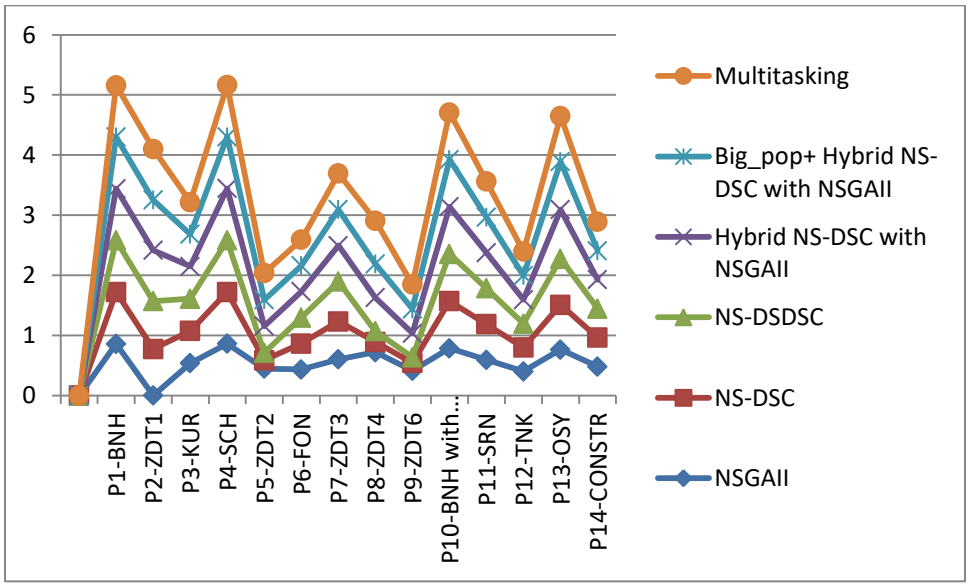

Figure 5. 15 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using HV metric on tested problems

Table 5. 6 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using Spacing Metric on tested problems

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Bigpop+ Hybrid NSDSC with NSGA-II	Multitasking on MOO
A1-BNH	0.5961	0.56665	0.48151	0.57847	0.58872	0.5593
A2-ZDT1	0.03765	0.004718	0.0003815	0.089579	0.008001	0.00347
A3-KUR	0.06996	0.06078	0.069982	0.070142	0.070052	0.07047
A4-SCH	0.01028	0.012869	0.012196	0.010959	0.011073	0.01054
A5-ZDT2	0.003759	0.019553	0.011106	0.003094	0.002727	0.003383
A6-FON	0.003807	0.002335	0.0027227	0.003274	0.003158	0.003221
A7-ZDT3	0.003453	0.028031	0.019944	0.003917	0.003585	0.003483
A8-ZDT4	0.003089	0.013755	0.016597	0.006197	0.004551	0.003259
A9-ZDT6	0.003186	0.002124	0.0019765	0.002747	0.002938	0.002890
A10-BNH constraint	0.38417	0.36059	0.33839	0.33839	0.36678	0.38447
A11-SRN	0.73881	0.81101	0.6317	0.73098	0.68232	0.70796
A12-TNK	0.00223	0.00535	0.00583	0.002470	0.002603	0.002336
A13-OSY	0.40935	0.47911	0.54839	0.84108	0.75347	0.48552
A14-CONSTR	0.019822	0.023928	0.02473	0.022508	0.02308	0.021149

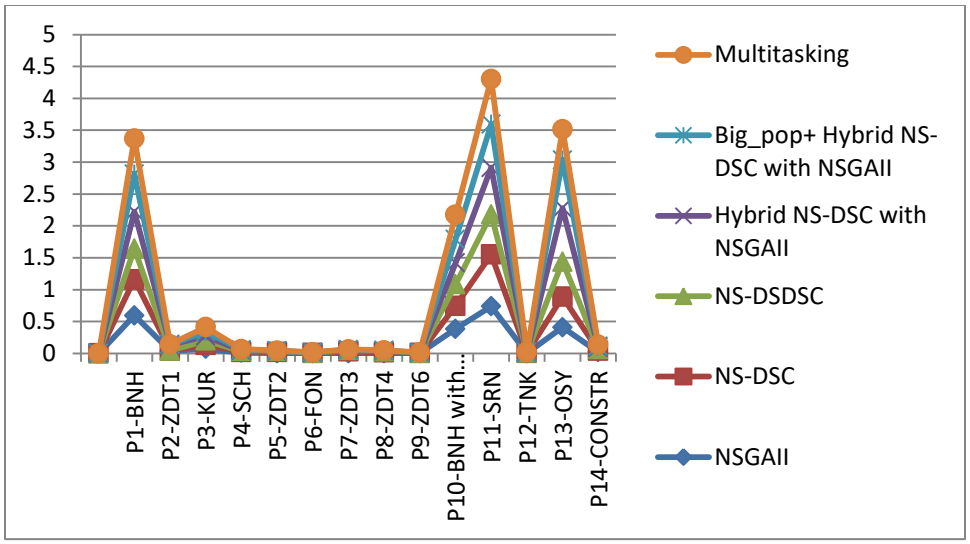

Figure 5. 7 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using Spacing Metric on tested problems

Table 5. 7 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using Spread Metric on tested problems

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Big pop+ Hybrid NSDSC with NSGA-II	Multitasking on MOO
A1-BNH	0.57437	0.52413	0.46573	0.52238	0.52768	0.52539
A2-ZDT1	0.97834	0.7774	1.0127	1.2227	0.65725	0.64806
A3-KUR	0.45015	0.57234	0.39988	0.42236	0.45985	0.46585
A4-SCH	0.28069	0.39845	0.35037	0.27786	0.29091	0.28365
A5-ZDT2	0.43246	1.3254	1.2223	0.32853	0.31169	0.37805
A6-FON	0.34146	0.23984	0.28528	0.35334	0.33519	0.35057
A7-ZDT3	0.39718	1.3466	1.2837	0.44075	0.38896	0.3832
A8-ZDT4	0.34949	1.2705	1.2446	0.85432	0.73496	0.35609
A9-ZDT6	0.38055	0.86746	0.89461	0.34829	0.41941	0.37613
A10-BNH constraint	0.5256	0.49571	0.46664	0.46888	0.48215	0.46803
A11-SRN	0.38891	0.41285	0.3247	0.3758	0.35875	0.36303
A12-TNK	0.30347	0.95284	1.0387	0.35513	0.38185	0.32494
A13-OSY	0.622	0.81751	0.75041	0.92576	0.88962	0.79495
A14-CONSTR	0.59736	0.7129	0.74199	0.57323	0.68624	0.68886

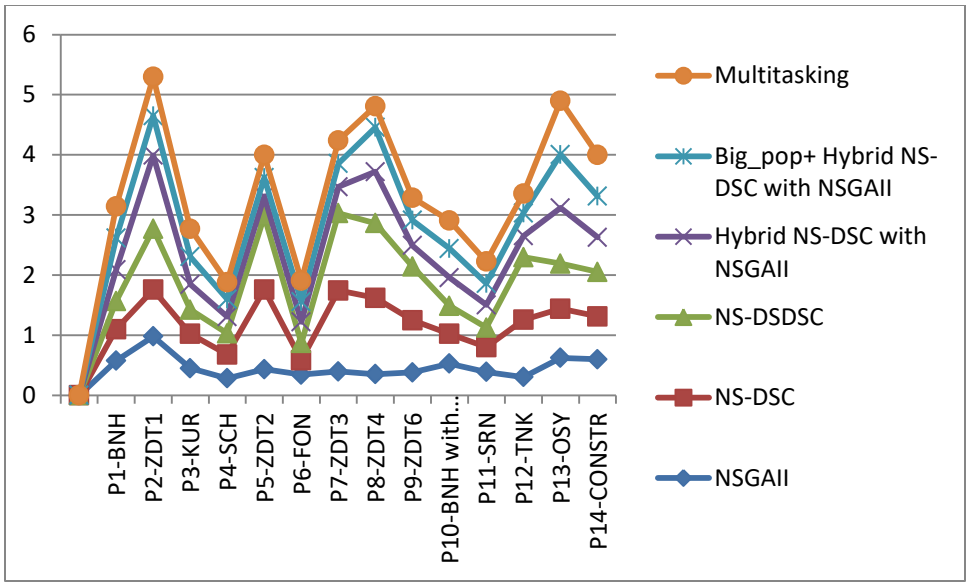

Figure 5. 8 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using Spread Metric on tested problems

Table 5. 8 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using DeltaP Metric on tested problems

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Big_pop+ Hybrid NSDSC with NSGA-II	Multitasking on MOO
A1-BNH	0. 34662	0.34985	0.3629	0.34194	0.32173	0.34156
A2-ZDT1	0. 008686	0.11187	0.075271	0.015551	0.19639	0.017231
A3-KUR	0.020047	0.0714	0.025259	0.02083	0.020674	0.019964
A4-SCH	0.008523	0.01038	0.010953	0.008202	0.009442	0.0091038
A5-ZDT2	0.002551	0.35667	0.38423	0.002716	0.006715	0.0023735
A6-FON	0.002587	0.002620	0.002826	0.002642	0.002555	0.0027975
A7-ZDT3	0.002910	0.26306	0.1088	0.003488	0.004524	0.0028701

Function name	NSGA-II	NSDSC	NS-DSDSC	Hybrid NSDSC with NSGA-II	Big_pop+ Hybrid NSDSC with NSGA-II	Multitasking on MOO
A8-ZDT4	0.002412	0.50553	0.50219	0.14408	0.13818	0.0033684
A9-ZDT6	0.002648	0.34736	0.40755	0.007098	0.002756	0.0024873
A10-BNH constraint	0.27149	0.24575	0.27119	0.25027	0.26853	0.26796
A11-SRN	0.54692	0.52435	0.54241	0.56506	0.55087	0.54289
A12-TNK	0.002017	0.004913	0.007407	0.002097	0.002185	0.0021155
A13-OSY	0.47384	2.1352	1.9782	5.3141	1.9908	0.89448
A14-CONSTR	0.012319	0.01357	0.013106	0.012023	0.013997	0.012662

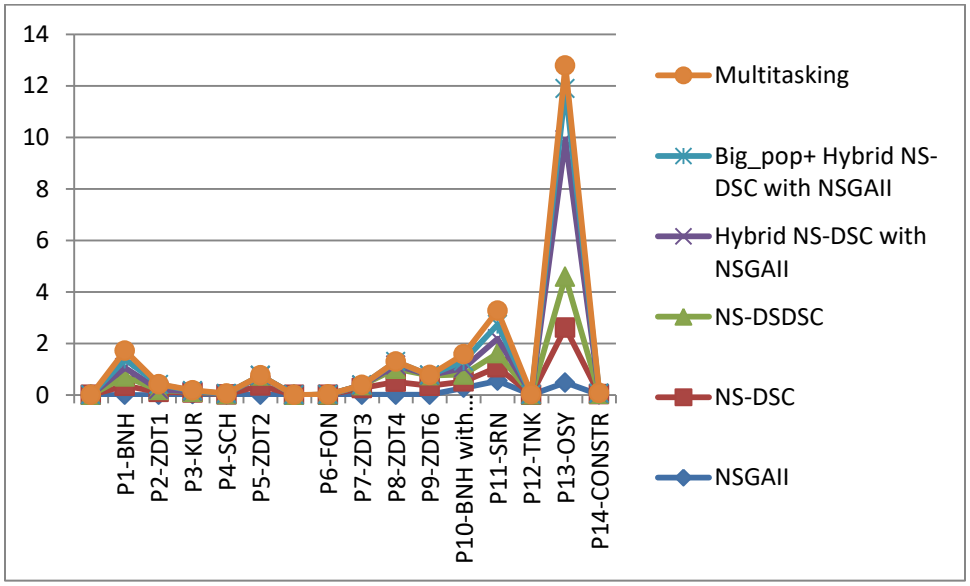

Figure 5. 9 shows the results of all algorithms: NSGA-II, NSDSC, NS-DSDSC, hybrid NSDSC with NSGA-II, Big population Hybrid NSDSC with NSGA-II and Multitasking on MOO using DeltaP Metric on tested problems

5.6 The scalarization function as a quality indicator for MOO algorithms

An important question regarding various MOO algorithms is how far are the generated solutions from the true Pareto front. To answer this question, many authors have constructed and used different *Quality Indicators* (QIs) to measure the distance between solutions generated by the MOO and true Pareto optimal solutions. A comprehensive survey of quality indicators is given in [112].

The quality indicators defined above in Section 2.5 require the knowledge of the true Pareto front *PF* or at least some elements of it. However, when we apply some numerical algorithm to solve an MOO problem, we usually do not know what the exact Pareto optimal solutions are. The authors of [112] write in (Section 4.4.2): "A reference set representing the Pareto front is needed in many QIs, such as IGD, the unary ϵ-indicator, and the relative HV (i.e., *hyperarea ratio* Van Veldhuizen (1999)). Ideally, such a reference set is expected to consist of sufficient points that are distributed uniformly and densely on the Pareto front. This, though, is not feasible in most cases. A practical alternative is to use the collection of the solution sets under consideration as the reference set. However, this common practice may come with two issues. First, whenever new solution sets are included in the comparison, the reference set needs to be reconstructed. And more importantly, such a collection of the sets may not well represent the Pareto front, and consequently the QI could return inaccurate results."

Here we propose a new quality indicator which does not require knowledge of the Pareto front. This indicator is defined as the scalarization function *s* which is described below. This new scalarization function was introduced in [113] and applied to some MOO problems in [111].

Let $f = (f_1, \dots, f_p): \mathbb{R}^n \longrightarrow \mathbb{R}^p$ be a locally Lipschitzian mapping (see [113]). Suppose that we want to solve the following MOO problem:

$$\text{minimize } f(x) \text{ subject to } x \in \mathbb{R}^n. \qquad (1)$$

A point $\bar{x} \in \mathbb{R}^n$ is called a *weak Pareto minimizer* for problem (1) if there exists no $x \in \mathbb{R}^n$ such that

$f_i(x) < f_i(\bar{x})$ for all $i \in I := \{1, \ldots, p\}$.

It is known (see [111]) that, if $\bar{x}$ is a weak Pareto minimizer for problem (1), then there exists a vector $\lambda = (\lambda_1, \ldots, \lambda_p) \in \mathbb{R}^p$ such that:

$$\lambda_i \geq 0 \ (i \in I), \quad \sum_{i \in I} \lambda_i = 1 \ and \ 0 \in \sum_{i \in I} \lambda_i \, \partial f_i(\bar{x}). \qquad (2)$$

where $\partial f_i(\bar{x})$ the generalized gradient of f_i at $\bar{x}$ in the sense of F.H. Clarke [114].

We say that a point $\bar{x} \in \mathbb{R}^n$ is a weak Pareto stationary point for problem (1) if there exists a vector $\lambda \in \mathbb{R}^p$ satisfying condition (2).

We now define the scalarization function $s : \mathbb{R}^n \longrightarrow [0, +\infty)$ as follows:

$$s(x) := d\big(0, co \bigcup_{i \in I} \partial f_i(x)\big), \qquad (3)$$

where $d(x, A)$ is the Euclidean distance from a point x to a set A (see [113]), and "co" denotes the convex hull.

It can also be proved (see [113], Proposition 3) that $\bar{x}$ is a weak Pareto stationary point for problem (1) if and only if $s(x) = 0$.

Let us now describe how the function s can be used as a performance indicator. Suppose that some algorithm has generated a point x as a possible solution of problem (1). Then the value $s(x)$ gives some information about how far is x from the set of Pareto optimal solutions. The points x with $s(x) = 0$ can be considered as "good" solutions (although they are not necessarily Pareto optimal points), whereas $s(x) > 0$ means that x is separated from the set of Pareto points by some positive distance. In fact, in this case, $s(x)$ is a measure of "unfulfillment" of the necessary optimality conditions (2).

Computing the value $s(x)$ for the general case as in formula (3) is difficult but, for two optimality criteria and continuously differentiable f, there exists another formula that much simpler, see (4) below.

Let $p = 2$ be the number of objectives and suppose that the mapping $f = (f_1, f_2)$ is continuously differentiable on $\mathbb{R}^n$. Then a representation of the scalarization function s can be found in terms of the gradients ∇f_1 and ∇f_2 [113]:

$$s(x) = d(0, co\{\nabla f_1(x), \nabla f_2(x)\}). \qquad (4)$$

where the gradients can be computed by some symbolic computation package, and *d* is the distance function as in (3).

It is shown in [113] that the following formula can be used for computing s:

$$s(x) = \begin{cases} \| b \| & if\ t_0 \leq 0, \\ \| b + t_0 a \| & if\ 0 < t_0 < 1, \\ \| b + a \| & if\ t_0 \geq 1. \end{cases} \qquad (5)$$

where $a := \nabla f_2(x) - \nabla f_1(x)$ is a line direction and $b := \nabla f1(x)$ is a point on the line

Below we present several examples of two-objectives unconstrained problems, where we first use some MOO evolutionary algorithms to generate a population of possible solutions, and then evaluate each of them by computing the values of *s*.

In our thesis we have used the IGD Metric, GD Metric, HV Metric, Spacing Metric, Spread Metric and DeltaP Metric, for all five new algorithms (ND-DSC, NSDSDSC, Hybrid NSDSC with NSGA-II, first big population with Hybrid NSDSC with NSGA-II and the last algorithm Multitasking on MOO). Also we have used a new metric that is $s(x)$ scalarization function, The results are shown in Chapter 5, we have tested all 5 algorithms on 14 multi-objectives problems.

The evolvement of novel multi-objective optimization algorithms has significantly increased during the past few years. To evaluate the fineness of Pareto front approximations generated by these algorithms, numerous performance measures have been established. In [86] the authors review 63 performance indicators that have been divided into four groups based on: cardinality, convergence, distribution, and spread. Also applications for these indicators are discussed [86]. Due to the fact that conflicting objectives are given, the Pareto set, or set of solutions, is defined as the collection of best decision vectors that correspond to the best trade-off points in the objective space.

Important for the evaluation of stopping criteria, the comparison of algorithms, or even the creation of multi-objective optimization techniques, a Pareto set approximation should meet the criteria listed in [115]:

- The Pareto front and its representation in the objective space should be as close as possible.
- It is preferred that the points of the associated approximated front have a good (by some metric) distribution in the objective space.
- The extent of the obtained nondominated front should be maximized, i.e., for each objective, a wide range of values should be covered by the nondominated solutions.

In our thesis, we have used the NSGA-II as a standard method to measure the performance of all our methods with drawing of true Pareto front, and the time in seconds was the measure of the speed to access the Pareto solution.

5.7 Application of a new metric $s(x)$

In this thesis, we use a new metric $s(x)$ to measure the efficiency of an algorithm, or the efficiency of solutions on a Pareto front. This concept was taken from the scalarization method described in [111]. We apply as a sample of unconstrained problems (SCH and FON) and a sample of constrainted problems (BNH and SRN) on this metric by using two algorithms first one is NSGA-II and the second algorithm is Multi-tasking on MOO algorithm

We found Also we apply as on this metic the results that were taken from the first nondominated front obtained by the NSGA-II algorithm and the Multitasking on MOO algorithm. When the result is $s(x) = 0$, the point x is a weak Pareto stationary point. This metric is considered as a new measure to know the efficiency of points on the Pareto front [11], NSGA-II often incorporates constraint handling by assigning a penalty to infeasible solutions. This penalty, which is based on the scale of the constraint violation, permits unfeasible solutions to be "ranked" lower than feasible ones [5].

In NSGA-II there special procedure which give lower rank of infeasible chromosomes which don’t satisfy the constraints, so these chromosomes will not be

allowed to create new population. Therefore in our work the function $s(x)$, which was constructed for unconstrained problems, is also used for constrained problems.

Tables 5.9 and 5.10 illustrate a sample of results (20 chromosomes) got from applying $s(x)$ on the SCH problem by using the NSGA-II and Multitasking on MOO algorithms, respectively. The SCH problem has one variable.

Table 5. 9 shows the results of applying $\boldsymbol{s(x)}$ on SCH problem using NSGA-II algorithm

No.	x1	f1	f2	front	Sx
1	0.000351460	1.2352447e-07	3.99859428	1	0
2	0.011288425	0.0001274285	3.95497372	1	0
3	0.022222346	0.0004938326	3.91160444	1	0
4	0.032691999	0.0010687668	3.87030076	1	0
5	0.042674786	0.0018211373	3.83112199	1	0
6	0.050893483	0.00259014670	3.79901621	1	0
7	0.062960296	0.00396399889	3.75212281	1	0
8	0.071358688	0.00509206242	3.71965730	1	0
9	0.085939295	0.007385562551	3.66362837	1	0
10	0.094248304	0.008882742981	3.63188952	1	0
11	0.101997081	0.010403404727	3.60241507	1	0
12	0.112916415	0.012750116827	3.56108445	1	0
13	0.118562698	0.014057113560	3.53980631	1	0
14	0.132227544	0.017484123584	3.48857394	1	0
15	0.138783261	0.019260793688	3.464127747	1	0
16	0.148228161	0.021971587978	3.429058940	1	0
17	0.155743713	0.024256104171	3.401281251	1	0
18	0.162774168	0.026495429961	3.375398755	1	0
19	0.173907638	0.030243866658	3.334613313	1	0
20	0.185772220	0.034511318072	3.291422434	1	0

Table 5. 10 shows the results of applying $\boldsymbol{s(x)}$ on SCH problem using Multi-tasking on MOO algorithm

No.	x1	f1	f2	front	Sx
1	-0.002667938	7.1178944e-06	4.01067887082	1	2.605e-05
2	0.00552586	3.0535120e-05	3.97792708449	1	0
3	0.01495059	0.00022327	3.94042113073	1	0
4	0.02360917	0.00055739	3.90612067409	1	0
5	0.03592094	0.001290314	3.85760653881	1	0
6	0.04447931	0.001978409	3.82406116907	1	0
7	0.05386822	0.002901782	3.78742888018	1	0
8	0.06971584	0.004860299	3.72599691571	1	0
9	0.07194611	0.005176245	3.71739161065	1	0
10	0.08688943	0.007549775	3.65999203891	1	0
11	0.10470221	0.010962553	3.59215369455	1	0
12	0.11485404	0.013191455	3.55377516067	1	0
13	0.12553033	0.015757962	3.51363578138	1	0
14	0.13997856	0.019593998	3.45967974741	1	0
15	0.16158902	0.02611102	3.37975490916	1	0
16	0.16972261	0.028805762	3.34991534700	1	0
17	0.17965961	0.032277572	3.31363916952	1	0
18	0.19117095	0.036546334	3.27186251258	1	0
19	0.20249460	0.04100406	3.23102565672	1	0
20	0.21178612	0.044853365	3.19770883559	1	0

Tables 5.11 and 5.12 illustrate the results of applying $s(x)$ on FON problem using NSGA-II and Multitasking on MOO algorithms respectively, where FON problem has three variables.

Table 5. 11 shows the results of applying $\boldsymbol{s}(\boldsymbol{x})$ on FON problem using NSGA-II algorithm

No.	x1	x2	x3	f1	f2	front	Sx
1	-0.57733	-0.57714	-0.57733	0.981674	4.58E-08	1	0.000315
2	-0.57528	-0.56319	-0.5615	0.980285	0.000456	1	0.021467
3	-0.43382	-0.46246	-0.42423	0.955258	0.055639	1	0.067166
4	-0.41611	-0.4657	-0.47437	0.958457	0.047885	1	0.104149
5	-0.39638	-0.32374	-0.35877	0.928383	0.134846	1	0.143547
6	-0.39492	-0.44716	-0.38598	0.946223	0.083192	1	0.118086
7	-0.37595	-0.37057	-0.36115	0.931992	0.121958	1	0.028931
8	-0.36385	-0.41912	-0.47347	0.949362	0.078183	1	0.193076
9	-0.27527	-0.20651	-0.21439	0.8603	0.302681	1	0.200501
10	-0.1785	-0.18122	-0.18353	0.821951	0.375676	1	0.015117
11	-0.1566	-0.1656	-0.13222	0.796918	0.419991	1	0.110763
12	-0.15472	-0.15773	-0.20728	0.815833	0.388384	1	0.180647
13	-0.14318	-0.0639	-0.13589	0.762853	0.476399	1	0.303184
14	-0.14186	-0.22614	-0.22389	0.835499	0.354634	1	0.278706
15	-0.13985	-0.08044	-0.13181	0.76542	0.471039	1	0.221413
16	-0.13061	-0.10295	-0.14918	0.775047	0.455541	1	0.156647
17	-0.03016	-0.04616	0.031969	0.651902	0.614361	1	0.31777
18	-0.01067	0.062989	-0.02335	0.621365	0.645863	1	0.359716
19	0.034846	0.039613	0.053326	0.576022	0.684369	1	0.072595
20	0.043676	0.061689	0.068685	0.5549	0.702221	1	0.09633

Table 5. 12 shows the results of applying $\boldsymbol{s}(\boldsymbol{x})$ on FON problem using Multitasking on MOO algorithm

No.	x1	x2	x3	f1	f2	front	Sx
1	-0.576906	-0.576907	-0.577233	0.9816418	4.06537e-07	1	0.0005335
2	-0.522932	-0.546195	-0.553145	0.976505	0.0045076	1	0.0463913
3	-0.504056	-0.498587	-0.535174	0.9716972	0.0132654	1	0.0598216
4	-0.475604	-0.490064	-0.518223	0.9682025	0.0212381	1	0.0672860
5	-0.475450	-0.489065	-0.461521	0.9640233	0.0311002	1	0.0439404
6	-0.448169	-0.429658	-0.440496	0.9550305	0.0556224	1	0.0314293
7	-0.442763	-0.401336	-0.516407	0.9590238	0.0514383	1	0.1934738
8	-0.441041	-0.470334	-0.499214	0.9628858	0.0354925	1	0.0936319
9	-0.431677	-0.449917	-0.501561	0.9607368	0.0422835	1	0.1184536
10	-0.427386	-0.381084	-0.476388	0.9520913	0.0687268	1	0.1648334
11	-0.419576	-0.407296	-0.337380	0.9391975	0.1054156	1	0.1653640
12	-0.415122	-0.370541	-0.471024	0.9493385	0.0772462	1	0.1771907
13	-0.411179	-0.405155	-0.406778	0.9455797	0.0827344	1	0.0111522
14	-0.392376	-0.373291	-0.421289	0.9416542	0.0953530	1	0.0885834
15	-0.386724	-0.373104	-0.442873	0.9435082	0.0916618	1	0.1343623
16	-0.383648	-0.399240	-0.359263	0.9363588	0.11024148	1	0.0760718
17	-0.381780	-0.334202	-0.385825	0.9313386	0.12545355	1	0.1115207
18	-0.380040	-0.392890	-0.346846	0.9336030	0.11846300	1	0.0911088
19	-0.374554	-0.439010	-0.399659	0.9446268	0.0877470	1	0.1171867
20	-0.3706262	-0.40142527	-0.35080168	0.93400175	0.11751076	1	0.092719

Tables 5.13 and 5.14 illustrate the results of applying $s(x)$ on constrained BNH problem using NSGA-II and Multitasking on MOO algorithms respectively, where BNH problem has two variables.

Table 5. 13 shows the results of applying $\boldsymbol{s}(\boldsymbol{x})$ on BNH constrained problem using NSGA-II algorithm

No.	x1	x2	f1	f2	front	Sx
1	7.43659964e-05	1.715217e-05	2.3297994e-08	49.999084824	1	0
2	0.01597180576	0.02879349376	0.00433665545	49.553431168	1	0
3	0.06249870813	0.02356568552	0.01784572021	49.143817493	1	1.11022e-16
4	0.04027674237	0.05803763457	0.01996233201	49.021846813	1	0
5	0.09143958217	0.06995541177	0.05301982730	48.399305017	1	2.2204e-16
6	0.09775907196	0.10255881040	0.08030058297	48.016896322	1	0
7	0.10848420787	0.13213833510	0.11691745185	47.623003933	1	2.220446e-16
8	0.08064094282	0.20627615859	0.19621126106	47.179881801	1	0
9	0.146089687315	0.19327495846	0.23478962523	46.665050948	1	0
10	0.19603752848	0.17113589661	0.27087283073	46.395983956	1	0
11	0.16329901784	0.25040344442	0.35747381683	45.952343831	1	0
12	0.16082371572	0.26089519122	0.37572227338	45.876741498	1	4.44089e-16
13	0.23576710642	0.23217327579	0.43796223386	45.430086736	1	0
14	0.26452830269	0.25077620305	0.53145570777	44.979818869	1	0
15	0.33279033127	0.24239433554	0.67801767396	44.417657750	1	8.881784e-16
16	0.32262540041	0.35556961221	0.92206759248	43.448566771	1	0
17	0.29589438179	0.47663303622	1.25893014561	42.589458356	1	0
18	0.43711062762	0.37191493665	1.31754568356	42.239130778	1	0
19	0.44407491889	0.39521888482	1.41360200205	41.960462463	1	8.88178e-16
20	0.45938525327	0.42641123514	1.57144540953	41.534896468	1	0

Table 5. 14 shows the results of applying $\boldsymbol{s(x)}$ on BNH constrained problem using Multi-tasking on MOO algorithm

No.	x1	x2	f1	f2	front	Sx
1	1.45484e-05	1.3887019e-06	8.54344092e-10	49.9998406	1	0
2	0.0016300034	0.087792231	0.030840531605	49.11348758694	1	0
3	0.0205881350	0.0125138119	0.002321867171	49.6695609970	1	0
4	0.0960152516	0.0505391919	0.047092553885	48.5462287023	1	0
5	0.1247673115	0.4045607339	0.716945077869	44.885955814	1	4.44089e-16
6	0.137576608	0.1687778019	0.189653078301	46.983869167	1	0
7	0.143261675	0.1168269805	0.136689804192	47.433285890	1	8.88178e-16
8	0.190240432	0.2028111275	0.309295101882	46.146808178	1	4.44089e-16
9	0.195845508	0.2121598447	0.333469051934	46.003313728	1	0
10	0.2038462791	0.2451994581	0.406704319220	45.611218706	1	8.88178e-16
11	0.2540274942	0.2528542870	0.513861033203	45.05964744557	1	0
12	0.2627115143	0.2042004896	0.442860718843	45.44159514	1	8.8817e-16
13	0.2633428071	0.3689690542	0.821950388379	43.88236898	1	0
14	0.3142024868	0.3839168761	0.984461482342	43.26492173	1	8.88178e-16
15	0.3158034185	0.2920813871	0.740173343496	44.10619527	1	0
16	0.3230037645	0.4298573223	1.156434997783	42.76049788	1	0
17	0.3628354955	0.4682001009	1.403443725415	42.04050496	1	0
18	0.3648345314	0.4413610264	1.311615164033	42.26594821	1	0
19	0.368413932	0.3079113903	0.922152999518	43.46728502	1	0
20	0.4119860808	0.4692248273	1.559617877537	41.57779538	1	0

Tables 5.14 and 5.15 illustrate the results of applying Sx on FON problem using NSGA-II and Multitasking on MOO algorithms respectively, where FON problem has three variables.

Table 5. 15 shows the results of applying $\boldsymbol{s}(\boldsymbol{x})$ on SRN constrained problem using NSGA-II algorithm

No.	x1	x2	f1	f2	front	Sx
1	-2.17532663922	3.02291879082	23.5255529785	-23.6701401873	1	8.8817e-16
2	-2.26078458006	3.01951023645	24.2327068328	-24.4254828157	1	8.8817e-16
3	-2.26440437173	3.20321517791	25.0393018117	-25.2337965142	1	8.8815e-16
4	-2.20276555883	3.48006283082	25.8139499873	-25.9756016743	1	0
5	-2.35658710759	3.30232391794	26.2805466491	-26.5099793914	1	8.8817e-16
6	-2.73604590983	2.57728612849	26.9179623912	-27.1122447196	1	0
7	-2.42400005761	3.53114314155	27.9784621127	-28.2226861215	1	4.4408e-16
8	-2.21387591011	4.10438123963	29.3939330668	-29.5620660720	1	0
9	-2.79093205015	3.35091220585	30.4798181088	-30.6451766510	1	0
10	-2.73739192369	3.64262241198	31.4263354510	-31.6199805256	1	0
11	-2.14727227977	4.59848221191	32.1489415920	-32.2745247474	1	0
12	-2.73203972947	4.08915189588	33.9350594375	-34.1312170011	1	0
13	-2.00810526169	5.27633692409	36.3519652772	-36.3600048436	1	0.44732615
14	-2.57944380482	4.75293769616	37.0558469128	-37.2995355964	1	2.2201e-16
15	-2.17371868764	5.26953588484	37.6488643555	-37.7924048607	1	0.19163439
16	-2.24851303656	5.31787544419	38.6939113733	-38.8806656806	1	0.13872481
17	-2.45028672459	5.19367200607	39.3919368257	-39.6394654159	1	1.1102e-16
18	-2.31463266915	5.54257973675	41.2510857345	-41.4667246871	1	0
19	-2.54048859062	5.43121426741	42.2516965252	-42.5000571993	1	2.775559e-17
20	-2.54081997842	5.56937881152	43.4982687996	-43.7466025289	1	0

Table 5. 16 shows the results of applying $s(x)$ on SRN constrained problem using Multi-tasking on MOO algorithm

No.	x1	x2	f1	f2	front	Sx
1	-3.28133668	14.44209436	210.5824179	-210.221930	1	0
2	-3.09060112	12.20775727	153.5280428	-153.429233	1	0
3	-3.08810730	11.74117557	143.2616886	-143.165818	1	1.7763e-15
4	-3.05102502	2.58698121	30.03136318	-29.9777346	1	0
5	-3.03295295834	9.79714757489	104.720420935	-104.686382	1	0
6	-3.01841492953	14.6897359975	214.593360086	-214.574606	1	0
7	-3.00513797349	13.3888119074	180.534066610	-180.528902	1	0
8	-2.96968183766	14.6330670212	212.558253972	-212.587652	1	0
9	-2.95052481675	14.6853161990	213.795575428	-213.842602	1	0
10	-2.93428441472	6.23532539001	53.7557946247	-53.8171916	1	0
11	-2.926485175	12.57841144	160.3298677	-160.397978	1	0
12	-2.923524624	12.30542674	154.0537685	-154.124395	1	0
13	-3.28133668	14.44209436	210.5824179	-210.221930	1	0
14	-3.09060112	12.2077574	153.5280428	-153.429233	1	0
15	-3.08810730	11.74117557	143.2616886	-143.165818	1	0
16	-3.05102502	2.586981214	30.03136318	-29.9777346	1	0
17	-3.03295295	9.797147574	104.7204209	-104.686382	1	0
18	-3.01841492	14.68973599	214.5933600	-214.574606	1	0
19	-3.00513797	13.38881190	180.5340666	-180.528902	1	
20	-2.969681837	14.63306702	212.5582539	-212.5876529	1	0

We found that the results was taken form Pareto front of algorithms NSGA-II and Multitasking on MOO reached to zero value for $s(x)$ in unconstraint test problems (SCH, FON), and constraint test problems (BNH, SRN).

This indicates that this measure $s(x)$ is considered as a new measure for measuring the efficiency of solutions on approximate of Pareto front.

5. 7 Conclusions

In this chapter, we have used a new algorithm called Multitasking on MOO, which includes the idea of combining single-objective and multi-objective optimization to solve problems, so the algorithm has worked to solve the MOO problems by dividing the problem into several tasks. So if we have two objective functions, the first task solves the first objective problem and the second task solves the second objective problem as a SOO, then put the first best three chromosomes for task 1 and best three chromosomes for task 2 in the first quarter of population and generate randomly 10% of population and put it in second quarter, while the third task is used to solve the first and second objective problems as MOO. These tasks are applied for first 10% of generations, then we use NSGA-II.

The results showed that the Multitasking on MOO algorithm was very effective and outperformed NSGA-II in some problems.

We also used in this chapter a new metric $s(x)$ that proved its ability in measuring the efficiency of the solutions resulting from two algorithms: Multitasking on MOO and NSGA-II on SCH and FON problems. Also, this metric $s(x)$ is considered a new addition to the existing performance measures.

CHAPTER SIX: Conclusions

In this thesis, two new algorithms called Non-dominated Sorting with Dissimilarity and Similarity of Chromosomes (NSDSC) and Non-dominated Sorting with Dynamic Schema Dissimilarity and Similarity of Chromosomes (NS-DSDSC) are introduced. In these algorithms, we used DSC and DSDSC instead of GA to find the best solutions.

With problems that have 1, 2, 3, or 6 variables, we noticed that our previous suggested algorithms (NSDSC, NS-DSDSDC) perform well but, with problems that have 10 or 30 variables, these algorithms didn't reach the Pareto front.

We noticed that the two new algorithms (NSDSC and NS-DSDSC) did not reach the optimal solution in some of problems (A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4, A9-ZDT6), but they were faster in reaching the optimal solution in the rest of the problems (A1-BNH,A3-KUR, A4-SCH, A6-FON, A10-BNH with constraint, A11-SRN, A12-TNK, A13-OSY, A14-CONSTR).

In the problems (A1 to A14) mentioned above, that reached the optimal solutions, we found that the optimal solutions of the two new algorithms NSDSC and NS-DSDSC matched the optimal solutions of the NSGA-II algorithm.

Also in this thesis, we used the concept of hybridization to obtain optimal solutions, hybridization has many advantages, the most important one, which is the integration of benefits and the elimination of defects, thus the good design for hybrid algorithms enables one to reach best solutions (Pareto front) faster than the original algorithms. So, we presented two hybrid algorithms: Hybrid NSDSC with NSGA-II and First Big population with Hybrid NSDSC with NSGA-II algorithm.

When we tested the first two algorithms NSDSC, NS-DSDSC Hybrid NSDSC with NSGA-II on the 14 problems listed in Appendix A, we found that hybrid algorithms performed better, especially for the problems A2-ZDT1, A5-ZDT2, A7-ZDT, A8-ZDT4 and A9-ZDT6 for which the NSDSC and NS-DSDSC algorithms have found difficulties to reach optimal solutions.

Specifically, for A8-ZDT4, we have not obtained a good solution, but when the number of iterations is increased (2000 big pop, 50 iterations in NSDSC, and 450 in NSGA-II), the First Big population with Hybrid NSDSC with NSGA-II algorithm performs better than NSGA-II. Moreover, hybrid algorithms have significantly shorter run times than NSDSC, NS-DSDSC, and NSGA-II.

Finally using a novel algorithm called Multitasking on MOO, which involves the concept of mixing single-objective and multi-objective problems, we found that this algorithm had the ability to solve the MOO problems by breaking them up into smaller tasks.

In our work, we have used seven efficiency metrics IGD, GD, HV, Spacing, Spread, DeltaP and $s(x)$ metric, we found that Multitasking on MOO algorithm outperformed NSGA-II and other algorithms from Chapters 3 and 4.

Appendix A: Minmiztion Test Functions

Unconstrained problems

A.1 Binh and Korn function (BNH) (P1)

$$f_1(x) = (4 * x_1^2) + (4 * x_2^2))$$
$$f_2(x) = (x_1 - 5)^2 + (x_2 - 5)^2$$

where $x_1 \in [0,5]$, $x_2 \in [0,3]$, this function is convex, the number of variables is 2.

A.2 Zitzler–Deb–Thiele's function N. 1 (ZDT1)

$$f_1(x) = x_1$$
$$f_2(x) = g(x) * \left[1 - \sqrt{x_1/g(x)}\right]$$
$$g(x) = 1 + 9 * \left(\sum_{i=2}^{n} (x_i - 0.2)^2\right)/(n-1)$$

where $n = 30$, $x_i \in [0,1], i = 1, 2, \dots, n$, the optimal solution is $x_1 \in [0,1]$, $x_i = 0$, $i = 2, \dots, n$, this function is convex.

A.3 Kursawe function

$$f_1(x) = \sum_{i=1}^{2} \left[-10exp\left(-0.2\sqrt{x_1^2 + x_{i+1}^2}\right)\right]$$
$$f_2(x) = \sum_{i+1}^{3} [|x_i|^{0.8} + 5\sin(x_i^3)]$$

where $x_i = [-5,5], i = 1, \dots, 3$, this function is non-convex , number of variables is 3.

A.4 Schaffer function N. 1 (SCH)

$$f_1(x) = x^2;$$
$$f_2(x) = (x-2)^2$$

where $x \in [0,2]$, this function is convex, the number of variables is 1.

A.5 Zitzler–Deb–Thiele's function N. 2 (ZDT2) [9]

$$f_1(x) = x_1$$
$$f_2(x) = g(x)[(1-(x_1/g(x))^2]$$
$$g(x) = 1 + 9 * \left(\sum_{i=2}^{n} x_i/(n-1)\right)$$

where $n = 30$, $x_i \in [0,1], i = 1, 2, \dots, n$, the optimal solution is $x_1 \in [0,1]$, $x_i = 0$, $i = 2, \dots, n$, this function is non-convex.

A.6 Fonseca–Fleming function (FON)

$$f_1 = 1 - exp\left(-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}})^2\right)$$
$$f_2 = 1 - exp\left(-\sum_{i=1}^{3}(x_i - \frac{1}{\sqrt{3}})^2\right)$$

where $x_i \in [-4,4]$ this function is non-convex, number of variables is 3.

A.7 Zitzler–Deb–Thiele's function N. 3 (ZDT3)

$$f_1 = x_1$$
$$f2 = g(x)\left[(1 - \sqrt{\left(\frac{x_1}{g(x)}\right)} - \left(\frac{x_1}{g(x)}\right) * sin(10\pi * x_1)\right]$$
$$g(x) = 1 + 9 * \left(\sum_{i=2}^{n} \frac{x(i)}{n-1}\right)/(n-1)$$

where $n = 30$, $x_i \in [0,1], i = 1, 2, \dots, n$, the optimal solution is $x_1 \in [0,1]$, $x_i = 0$, $i = 2, \dots, n$, this function is convex and disconnected, i.e. the Pareto front is not continuous .

A.8 Zitzler–Deb–Thiele's function N. 4 (ZDT4)

$$f_1(x) = x_1$$

$$f_2(x) = g(x) * \left(1 - \sqrt{\left(\frac{x_1}{g(x)}\right)}\right)$$

$$g(x) = 1 + \left(10 * (n-1)\right) + \sum_{i=2}^{n} (x_i^2 - 10 * \cos(4\pi x_i))$$

where $n = 10,\ x_1 \in [0,1], x_i \in [-5,5],\ i = 2, \dots, n,$ the optimal solution is $x_1 \in [0,1]$, $x_i = 0,\ i = 2, \dots, n$, this function is non-convex.

A.9 Zitzler–Deb–Thiele's function N. 6 (ZDT6)

$$f_1(x) = 1 - exp(-4 * x_1) . sin^6 6\pi x_1$$
$$f_2(x) = g(x) * \ (1 - f_1(x)/g(x))^2$$
$$g(x) = 1 + 9 * \left(\sum_{i=2}^{n} x_i/n - 1\right)$$

where $n = 10,\ x_i \in [0,1],\ i = 1, \dots, n,$ the optimal solution is $x_1 \in [0,1]$, $x_i = 0,\ i = 2, \dots, n$, this function is non-convex.

Constrained problems

A.10 Binh and Korn function (BNH) with constraint [116]

Minimize

$$f_1(x) = (4 * x_1^2) + (4 * x_2^2)$$
$$f_2(x) = (x_1 - 5)^2 + (x_2 - 5)^2$$

subject to

$$c(1) = (x_1 - 5)^2 + x_2^2 \leq 25$$
$$c(2) = -(x_1 - 8)^2 - (x_2 + 3)^2 \geq 7.7$$

This function has 2 variables.

A.11 SRN [117]

Minimize

$$f_1(x) = (x_1 - 2)^2 + (x_2 - 1)^2 + 2$$
$$f_2(x) = 9 * x_1 - (x_2 - 1)^2$$

Subject to:

$$c(1) = x_1^2 + x_2^2 \leq 225;$$
$$c(2) = x_1 - (3 * x_2) \leq -10;$$

This function has 2 variables.

A.12 TNK

Minimize

$$f_1(x) = x_1$$
$$f_2(x) = x_2$$

Subject to:

$$x_1^2 + x_2^2 \geq 1 + 0.1 * cos(16 * arctan\frac{x_2}{x_1})$$
$$(x_1 - 0.5)^2 + (x_2 - 0.5)^2 \leq 0.5;$$

where $0 < x_1, x_2 < \pi$, the function has 2 variables.

A.13 OSY [24]

Minimize

$$f_1(x) = -((25 * (x_1 - 2)^2) + ((x_2 - 2)^2) + ((x_3 - 1)^2) + ((x_5 - 4)^2) + ((x_5 - 1)^2));$$
$$f_2(x) = (x_1^2) + (x_2^2) + (x_3^2) + (x_4^2) + (x_5^2) + (x_6^2)$$

Subject to:

$$C_1(x) = x_1 + x_2 - 2 \geq 0$$
$$C_2(x) = 6 - x_1 - x_2 \geq 0;$$
$$C_3(x) = 2 + x_1 - x_2 \geq 0;$$
$$C_4(x) = 2 - x_1 - 3x_2 \geq 0;$$
$$C_5(x) = 4 - (x_3 - 3)^2 - x_4 \geq 0;$$
$$C_1(x) = (x_5 - 3)^2 + x_6 - 4 \geq 0;$$

where $0 \leq x_1, x_2, x_6 \leq 10, 1 \leq x_3, x_5 \leq 5,\ 0 \leq x_4 \leq 6,$ the function has 6 variables.

A.14 CONSTR function

Minimize

$$f_1(x) = x_1$$
$$f_2(x) = (1 + x_2)/x_1$$

Subject to:

$$c_1(x) = x_2 + 9 * x_1 \geq 6$$
$$c_2(x) = -x_2 + 9 * x_1 \geq 1$$

where $x_1 \in [0.1,1.0], x_2 \in [0,5]$, the function has 2 variables.

References

[[1] E. K.P. Chong and S. H. Zak, *An Introduction to optimization*. Wiley, USA, 2001.

[2] R. Subramani and C. Vijayalakshmi, "A review on advanced optimization techniques," *ARPN J. Eng. Appl. Sci.*, vol. 11, no. 19, pp. 11675–11683, 2016.

[3] K. Qiao, K. Yu, B. Qu, J. Liang, H. Song, and C. Yue, "An evolutionary multitasking optimization framework for constrained multi-objective optimization problems," *IEEE Trans. Evol. Comput.*, vol. PP, no. 8, p. 1, 2015, doi: 10.1109/TEVC.2022.3145582.

[4] M. Mladineo, I. Veža, and N. Gjeldum, "Single-objective and multi-objective optimization using the HUMANT algorithm," *Croat. Oper. Res. Rev. CRORR*, vol. 6, pp. 459–473, 2015, doi: 10.17535/crorr.2015.0035.

[5] C. A. C. Coello, *Evolutionary algorithms for solving multi-objective problems*, Second Edi. Springer, Science+Business Media, LLC All, 2007.

[6] T. T. Binh and U. Korn, "Multiobjective evolution strategy with linear and nonlinear constraints," *Proc. 15 IMACS World Congr. Sci. Comput.*, pp. 357–363, 1997.

[7] Y. Zhou, Y. Xiang, and X. He, "Constrained multiobjective optimization: test problem construction and performance evaluations," *IEEE Trans. Evol. Comput.*, vol. 25, no. 1, pp. 172–186, 2021, doi: 10.1109/TEVC.2020.3011829.

[8] I. Giagkiozis and P. J. Fleming, "Methods for multi-objective optimization: an analysis," *Inf. Sci. (Ny).*, vol. 293, pp. 338–350, 2015, [Online]. Available: http://eprints.whiterose.ac.uk/86090/

[9] M. G. C. Resende and J. P. de Sousa, *Metaheuristics: computer-decision making*. Springer Science+ Business Media, LL, 2004. doi: 10.1007/978-1-4757-4137-7.

[10] A. Gupta, Y. Ong, and L. Feng, "Multifactorial evolution: toward evolutionary multitasking," *IEEE Trans. Evol. Comput.*, vol. 20, no. 3, pp. 343–357, 2015.

[11] A. Younus, "Converting a multi-Objective optimization problem to a single-objective optimization problem by using a new scalarization method," M.Sc. Thesis, University of Lodz, 2016.

[12] R. Al-jawadi, "New evolutionary optimization algorithms using similarities and dissimilarities in binary strings," *Ph.D. thesis, Univ. Warsaw*, 2018.

[13] E. Ashis, K. Mishra, E. Y. Mohapatra, E. Anil, and K. Mishra, "Multi-objective genetic algorithm: a comprhensive survey," vol. 3, no. 2, 2013.

[14] M. Chen, J. Weng, X. Li and X. Zhang, "Handling multiple objectives with integration of particle swarm optimization and extremal optimization, " *conf.* Advances in Intelligent Systems and Computing 277, Springer-Verlag Berlin Heidelberg 2014, doi: 10.1007/978-3-642-54924-3_27.

[15] I. Boussaïd, J. Lepagnot, and P. Siarry, "A survey on optimization metaheuristics," *Inf. Sci. Elsevier*, vol. 237, pp. 82–117, 2013, doi: 10.1016/j.ins.2013.02.041.

[16] K. Deb, "Multi-objective optimization using evolutionary algorithms: an introduction," *Multi-objective Evol. Optim. Prod. Des. Manuf. Springer, London .*, pp. 3–34, 2011.

[17] P. K. Shukla, K. Deb, and S. Tiwari, "Comparing classical generating methods with an evolutionary multi-objective optimization method," Lecture Notes in Computer Science, vol 3410. Springer, Berlin, Heidelberg, pp. 311-325,2005. https://doi.org/10.1007/978-3-540-31880-4_22.

[18] R. Sarker and C. A. C. Coello, "Assessment methodologies for multiobjective evolutionary algorithms," In: Evolutionary Optimization. International Series in Operations Research & Management Science, Springer, Boston, MA, vol. 48, pp. 177–196, 2002. https://doi.org/10.1007/0-306-48041-7_7.

[19] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach , part I: solving problems with box constraints," in IEEE Transactions on Evolutionary Computation, vol. 18, no. 4, pp. 577–601, 2014.

[20] T. Dingsoyr, G. K. Hanssen, T. Dyba, G. Anker, and J. O. Nygaard, *Multiobjective optimization, interactive and evolutionary approaches*. Springer Science & Business Media, ebook, 2008.

[21] P. Chand and J. R. Mohanty, "Solving vehicle routing problem with proposed non-dominated sorting genetic algorithm and comparison with classical evolutionary algorithms," *Int. J. Comput. Appl.*, vol. 69, no. 26, pp. 34–41, 2013.

[22] N. Srinivas and K. Deb, "Muiltiobjective optimization using nondominated sorting in genetic algorithms," *Evol. Comput.*, vol. 2, no. 3, pp. 221–248, 1994, doi: 10.1162/evco.1994.2.3.221.

[23] E. Zitzler, "Evolutionary Algorithms for Multiobjective Optimization: Methods and Applications," *Ph.D. thesis*, 2002.

[24] K. Amouzgar, "Multi-objective optimization using genetic algorithms," *Thesis Work. Tek. Hogsk.*, p. 79, 2012.

[25] C. Grosan, M. Oltean, and M. Oltean, "The role of elitism in multiobjective optimization with evolutionary algorithms," *Acta Univ. Apulensis. Math. - Informatics*, vol. 5, no. January 2017, pp. 83–90, 2003.

[26] C. M. Fonseca and P. J. Fleming, "Multiobjective optimization and multiple constraint handling with evolutionary algorithms - Part I: aunified formulation," *IEEE Trans. Syst. Man, Cybern. Part ASystems Humans.*, vol. 28, no. 1, pp. 26–37, 1998, doi: 10.1109/3468.650319.

[27] D. Mart and A. Rosete, "A Multi-objective evolutionary algorithm for mining quantitative association rules," *11th Int. Conf. Intell. Syst. Des. Appl.*, pp. 1397–1402, 2011.

[28] J. Horn, N. Nafpliotis, and D. E. Goldberg, "A niched Pareto genetic algorithm for multiobjective optimization," *First IEEE Conf. Evol. Comput. (held Orlando),piscataw. IEEE Neural Networks Counci*, pp. 82–87, 1994.

[29] S. Y. Ivanov, "Application of multi-objective optimization in the design and operation of industrial catalytic," pp. 1–28, 2016, doi: 10.1515/psr-2015-0017.

[30] B. Jarraya and A. Bouri, "Metaheuristic optimization backgrounds: a literature review," *Int. J. Contempopary Bus. Stud.*, vol. 3, no. 12, pp. 31–44, 2012.

[31] T. El-Ghazali, *Metaheuristics from design to implementation*. JohnWiley & Sons, Inc., Hoboken, New Jersey, Canada, 2009.

[32] A. Osyczka and S. Kundu, "A new method to solve generalized multicriteria optimization problems using the simple genetic algorithm," *Struct. Optim.*, vol. 10, no. 2, pp. 94–99, 1995, doi: 10.1007/BF01743536.

[33] P. Schadler, J. D. Berdugo, T. Hanne, and R. Dornberger, "A distance-based pareto evolutionary algorithm based on SPEA for combinatorial problems," *2016 4th Int. Symp. Comput. Bus. Intell. ISCBI 2016*, pp. 112–117, 2016, doi: 10.1109/ISCBI.2016.7743268.

[34] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: improving the strength Pareto evolutionary algorithm," *Evol. Methods Des. Optim. Control with Appl. to Ind. Probl.*, pp. 95–100, 2001, doi: 10.1.1.28.7571.

[35] N. Mori, Y. Yabumoto, H. Kita, and Y. Nishikawa, "Multi-objective optimization by means of the thermodynamical genetic algorithm," *Trans. Inst. Syst. Control Inf. Eng.*, vol. 11, no. 3, pp. 103–111, 1998, doi: 10.5687/iscie.11.103.

[36] R. Dreżewski and L. Siwik, "Agent-based co-evolutionary techniques for solving multi-objective optimization problems," *Source Adv. Evol. Algorithms, B. Ed. by Witold Kosiński, ISBN 978-953-7619-11-4, pp. 468, Novemb. 2008, I-Tech Educ. Publ. Vienna, Austria*.

[37] E. Zitzler, "Evolutionary algorithms for multiobjective optimization: methods and applications,"Ph.D. thesis, 1999.

[38] K. Deb, A. Member, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[39] K. Deb, *Multi-objective optimization using evolutionary algorithms*. John Wiley & Sons, New York, 2001.

[40] S. Greco, *Trends in multiple criteria decision analysis*. Springer New York Dordrecht Heidelberg London, 2010.

[41] R. Al-jawadi, M. Studniarski, and A. Younus, "New genetic algorithm based on dissimilaries and similarities," *Comput. Sci. Journal, AGH Univ. Sci. Technol. Pol.*, vol. 19, no. 1, p. 19, 2018, doi: 10.7494/csci.2018.19.1.2522.

[42] G. Chi-Keong, O. Yew-Soon, and T. Kay Chen, *Multi-objective memetic algorithms*, vol. 1. Springer-Verlag Berlin Heidelberg, 2009. doi: 10.1017/CBO9781107415324.004.

[43] K. W. C. Ku and M. W. Mak, "Exploring the effects of Lamarckian and Baldwinian learning in evolving recurrent neural networks," Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97), Indianapolis, IN, USA, 1997, pp. 617-621, doi: 10.1109/ICEC.1997.592386.

[44] X.-S. Yang, "Multi-objective optimization," *Nature-Inspired Optim. Algorithms,* " pp. 197–211, 2014, doi: 10.1016/b978-0-12-416743-8.00014-2.

[45] A. Konak, D. W. Coit, and A. E. Smith, "Multi-objective optimization using genetic algorithms: a tutorial," *Reliab. Eng. Syst. Saf.*, vol. 91, pp. 992–1007, 2006, doi: 10.1016/j.ress.2005.11.018.

[46] D. Savic, "Single-objective vs . multiobjective optimisation for integrated decision support," *1ST Int. Congr. Environ. Model. Softw. - LUGANO, Switz. - JUNE 2002*, pp. 7–12, 2002.

[47] A. Fita, "Multiobjective programming with continuous genetic algorithm," International Journal of Scientific & Technology Research, vol. 3, pp.135-149, 2014.

[48] A. Quemy and S. Marc, "True Pareto fronts for multi-objective AI planning," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9026, pp. 197–208, 2015, doi: 10.1007/978-3-319-16468-7.

[49] K. Zhuo *et al.*, "A new evolutionary decision theory for many-objective optimization problems," *Adv. Comput. Intell.*, Lecture Notes in Computer Science

(LNTCS),vol. 4683, pp. 1–11, 2007.

[50] M. Chen, Y. Guo, H. Liu, and C. Wang, “The evolutionary algorithm to find robust Pareto-optimal solutions over time,” vol. 2015, 2015.

[51] Q. Lin, W. Lin, Z. Zhu, M. Gong, J. Li, and C. A. C. Coello, “Multimodal multiobjective evolutionary optimization with dual clustering in decision and objective spaces,” *IEEE Trans. Evol. Comput.*, vol. 25, no. 1, pp. 130–144, 2021, doi: 10.1109/TEVC.2020.3008822.

[52] S. Bandyopadhyay and S. Saha, "*some single- and multiobjective optimization techniques,"* Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-32451-2_2

[53] O. L. De Weck, “Multiobjective optimization: history and promise,” *IEEE Trans. Evol. Comput.*, vol. 13, no. 4, pp. 810–824, 2004, doi: 10.1109/TEVC.2009.2017515.

[54] N. Gunantara, “A review of multi-objective optimization: methods and its applications,” *Cogent Eng.*, vol. 5, no. 1, pp. 1–16, 2018, doi: 10.1080/23311916.2018.1502242.

[55] K. Talukder, “On the Pareto-following variation operator for fast converging multiobjective evolutionary algorithms master of engineering science,”master thesis, Department of Computer Science and Software Engineering ,the university of Melbourne, Australia, 2008.

[56] R. Al-jawadi, “An optimization algorithm based on dynamic schema with dissimilarities and similarities of chromosomes,” *Int. J. Comput. Electr. Autom. Control Inf. Eng.*, vol. 7, no. 8, pp. 1278–1285, 2016, doi: 10.5281/zenodo.1126041.

[57] L. Shi and S. Olafsson, “Two-stage nested partitions method for stochastic optimization,” *Methodol. Comput. Appl. Probab.*, vol. 2, no. 3, pp. 271–291, 2000, doi: 10.1023/B:MCAP.0000012413.54789.cc.

[58] S. Ólafsson, “Metaheuristics,” *Handbooks Oper. Res. Manag. Sci.*, pp. 633–654,

2006, doi: 10.1016/S0927-0507(06)13021-2.

[59] K. Manda, S. C. Satapathy, and B. Poornasatyanarayana, "Population based meta-heuristic techniques for solving optimization problems : A selective survey," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 2, no. 11, pp. 206–211, 2012.

[60] A. Fita, "Metaheuristic start for gradient based optimization algorithms," *Am. J. Comput. Appl. Math.*, vol. 5, no. 3, pp. 88–99, 2015, doi: 10.5923/j.ajcam.20150503.03.

[61] A. Cruz-bernal, "Meta-Heuristic optimization techniques and its applications in robotics," *INTECH, World Larg. Sci. , Technol. Med. Open Access B. Publ.*, pp. 53–75, 2013.

[62] I. Younas, "Using genetic algorithms for large scale optimization of assignment , planning and rescheduling problems," Doctoral Thesis in Electronics and Computer Systems Stockholm, Sweden, 2014.

[63] J. Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control and artificial intelligence*. (Second edition: MIT Press, ISBN 0-262-58111-6. NB original printing 1975).

[64] S. N. Sivanandam and S. N. Deepa, *Introduction to genetic algorithms*. Springer-Verlag Berlin Heidelberg, 2008.

[65] K. F. Man, K. S. Tang, and S. Kwong, "Genetic algorithms : concepts and applications," *IEEE Trans. Ind. Electron.*, vol. 43, no. 5, pp. 519–534, 1996, doi: 10.1109/41.538609.

[66] S. A.-H. Soliman and A.-A. H. Mantawy, *Modern optimization techniques with applications in electric power systems, chapter 2, mathematical optimization techniques*. Springer, 2012. doi: 10.1007/978-1-4614-1752-1.

[67] G. Subashini and M. C. Bhuvaneswari, "Comparison of multi-objective evolutionary approaches for task scheduling in distributed computing systems," Sadhana , vol. 37, pp. 675–694, 2012. https://doi.org/10.1007/s12046-012-0102-4

[68] G. P. Rajappa, "Solving combinatorial optimization problems using genetic

algorithms and ant colony optimization," PhD diss., University of Tennessee, 2012. https://trace.tennessee.edu/utk_graddiss/1478 2012.

[69] B. Tang, Z. Zhu, and J. Luo, "Hybridizing particle swarm optimization and differential evolution for the mobile robot global path planning," *Int. J. Adv. Robot. Syst.*, vol. 13, no. 3, p. 17, 2016, doi: 10.5772/63812.

[70] V. Savsani, "HBBABC: a hybrid optimization algorithm combining biogeography based optimization (BBO) and artificial bee colony (ABC) optimization for obtaining global solution of discrete design problems," *I nternational J. Comput. Eng. Res.*, vol. 2, no. 7, pp. 85–97, 2012.

[71] W. Y. Lin, "A GA-DE hybrid evolutionary algorithm for path synthesis of four-bar linkage," *Mech. Mach. Theory, Elsevier*, vol. 45, no. 8, pp. 1096–1107, 2010, doi: 10.1016/j.mechmachtheory.2010.03.011.

[72] M. H. Marghny, E. A. Zanaty, W. H. Dukhan, and O. Reyad, "A hybrid multi-objective optimization algorithm for software requirement problem," *Alexandria Eng. J.*, vol. 61, no. 9, pp. 6991–7005, 2022, doi: 10.1016/j.aej.2021.12.043.

[73] R. Jiang, S. Ci, D. Liu, X. Cheng, and Z. Pan, "A hybrid multi-objective optimization method based on NSGA-II algorithm and entropy weighted TOPSIS for lightweight design of dump truck carriage," *Machines*, vol. 9, no. 8, pp. 1–16, 2021.

[74] C. Krasopoulos, I. Armouti, and A. Kladas, "Hybrid multi-objective optimization algorithm for PM motor," in *IEEE Transactions on Magnetics*, 2017, no. February, pp. 1–5. doi: 10.1109/TMAG.2017.2663408.

[75] Y. Xu and R. Qu, "Hybrid multi-objective evolutionary algorithms based on decomposition for wireless sensor network coverage optimization," *Appl. Soft Comput.*, vol. 68, no. April, pp. 268–282, 2018, doi: 10.1016/j.asoc.2018.03.053.

[76] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer Heidelberg New York Dordrecht London, 2003. doi: 10.1162/evco.2004.12.2.269.

[77] M. Premkumar, P. Jangir, R. Sowmya, H. H. Alhelou, A. A. Heidari, and H. Chen,

"MOSMA: multi-objective slime mould algorithm based on elitist non-dominated sorting," *IEEE Access*, vol. 9, pp. 3229–3248, 2021, doi: 10.1109/ACCESS.2020.3047936.

[78] D. a. Van Veldhuizen and G. B. Lamont, "Evolutionary computation and convergence to a Pareto front," *Late Break. Pap. Genet. Program. 1998 Conf.*, pp. 221–228, 1998, [Online]. Available: https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=f329eb18a4549daa83fae28043d19b83fe8356fa

[79] N. Riquelme, "Performance metrics in multi-objective optimization," Latin American Computing Conference (CLEI), Arequipa, Peru, 2015, pp. 1-11, doi: 10.1109/CLEI.2015.7360024.

[80] Y. Liu, J. Wei, X. I. N. Li, and M. Li, "Generational distance indicator-based evolutionary algorithm with an improved Niching method for many-objective optimization problems," *IEEE Access*, vol. 7, pp. 63881–63891, 2019, doi: 10.1109/ACCESS.2019.2916634.

[81] D. A. Van Veldhuizen, "Multiobjective evolutionary algorithms: classifications, analyses, and new innovations". Department of Electrical and Computer Engineering, Graduate School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, Ohio," 1999. [Online]. Available: https://scholar.afit.edu/etd/5128

[82] C. A. C. Coello and M. R. Sierra, "A study of the parallelization of a coevolutionary multi-objective evolutionary algorithm," *Lect. Notes Artif. Intell. (Subseries Lect. Notes Comput. Sci.*, vol. 2972, pp. 688–697, 2004, doi: 10.1007/978-3-540-24694-7_71.

[83] Y. Sun, G. G. Yen, and Z. Yi, "IGD indicator-based evolutionary algorithm for many-objective optimization problems," *IEEE Trans. Evol. Comput.*, vol. 23, no. 2, pp. 173–187, 2019.

[84] E. Zitzler and L. Thiele, "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach," *IEEE Trans. Evol. Comput.*, vol. 3,

no. 4, pp. 257–271, 1999, doi: 10.1109/4235.797969.

[85] A. Auger, J. Bader, D. Brockhoff, and E. Zitzler, "Hypervolume-based multiobjective optimization: theoretical foundations and practical implications," *Theor. Comput. Sci.*, vol. 425, pp. 75–103, 2012, doi: 10.1016/j.tcs.2011.03.012.

[86] C. Audet, J. Bigeon, and D. Cartier, "Performance indicators in multiobjective optimization," *Eur. J. Oper. Res.*, vol. 292, no. 2, pp. 397–422, 2021, doi: 10.1016/j.ejor.2020.11.016.

[87] Y. N. Wang, L. H. Wu, and X. F. Yuan, "Multi-objective self-adaptive differential evolution with elitist archive and crowding entropy-based diversity measure," *Soft Comput.*, vol. 14, no. 3, pp. 193–209, 2010, doi: 10.1007/s00500-008-0394-9.

[88] O. Sch, X. Esquivel, A. Lara, and C. A. C. Coello, "Using the averaged hausdorff distance as a performance measure in evolutionary multiobjective optimization," in IEEE Transactions on Evolutionary Computation, vol. 16, no. 4, pp. 504-522, Aug. 2012, doi: 10.1109/TEVC.2011.2161872.

[89] A. H. Halim, I. Ismail, and S. Das, " *Performance assessment of the metaheuristic optimization algorithms: an exhaustive review*, " Artif Intell Rev, vol. 54, pp. 2323–2409, 2021, https://doi.org/10.1007/s10462-020-09906-6.

[90] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, Springer N., vol. 146. Springer New York Dordrecht Heidelberg London, 2010. doi: 10.1007/978-1-4614-1900-6.

[91] S. Poles, Y. Fu, and E. Rigoni, "The effect of initial population sampling on the convergence of multi-objective genetic algorithms," in *Multiobjective Programming and Goal Programming. Lecture Notes in Economics and Mathematical Systems,* 2009, vol. 618, pp. 123–133.

[92] M. Laumanns, "Analysis and applications of evolutionary multiobjective optimization algorithms," *THESIS Work , SWISS Fed. Inst. Technol. ZURICH*, no. 15251, 2003.

[93] H. Maaranen, K. Miettinen, and A. Penttinen, "Analysis and applications of

evolutionary multiobjective optimization algorithms," *J. Glob. Optim. Springer Sci. + Bus. Media B. V. 2006*, vol. 37, no. 3, pp. 405–436, 2007, doi: 10.1007/s10898-006-9056-6.

[94] C. Haubelt, G. Jurgen, and J. Teich, "Initial population construction for convergence improvement of MOEAs," *Evol. Multi-Criterion Optim. EMO 2005. Lect. Notes Comput. Sci.*, vol. 3410, pp. 191–205, 2005.

[95] A. Ebrahimi and Z. Ahmad, "A modified NSGA-II solution for a new multi-objective hub maximal covering problem under uncertain shipments," *J. Ind. Eng. Int. springer*, vol. 10, pp. 185–197, 2014, doi: 10.1007/s40092-014-0076-4.

[96] S. Ramesh, S. Kannan, and S. Baskar, "Application of modified NSGA-II algorithm to multi-objective reactive power planning," *Appl. Soft Comput. J.*, vol. 12, no. 2, pp. 741–753, 2012, doi: 10.1016/j.asoc.2011.09.015.

[97] F. Ye, W. Qi, and J. Xiao, "Research of Niching genetic algorithms for optimization in electromagnetics," *Procedia Eng.*, vol. 16, pp. 383–389, 2011, doi: 10.1016/j.proeng.2011.08.1099.

[98] P. C. Chang, S. H. Chen, C. Y. Fan, and C. L. Chan, "Genetic algorithm integrated with artificial chromosomes for multi-objective flowshop scheduling problems," *Appl. Math. Comput.*, vol. 205, no. 2, pp. 550–561, 2008, doi: 10.1016/j.amc.2008.05.027.

[99] N. J. D. S. Lima, C. J. A. Bastos-Filho, and D. R. B. Araújo, "Boolean operators to improve multi-objective evolutionary algorithms for designing optical networks," *J. Microwaves, Optoelectron. Electromagn. Appl.*, vol. 15, no. 4, pp. 319–332, 2016, doi: 10.1590/2179-10742016v15i4678.

[100] H. Ishibuchi and Y. Nojima, "Optimization of scalarizing functions through evolutionary multiobjective optimization," *Lect. Notes Comput. Sci. 4403 Evol. Multi-Criterion Optim. - EMO , Springer, Berlin*, pp. 51–65, 2007

[101] M. Li and H. Dankowicz, "Optimization with equality and inequality constraints using parameter continuation ," Applied Mathematics and Computation, vol. 375 pp. 1–27, 2020, https://doi.org/10.1016/j.amc.2020.125058.

[102] T. Weise, *Global optimization algorithms–theory and application*. e-Book, 2009, Jun 26, 361:p.153.

[103] J. Arora, "Numerical methods for constrained optimum design." (2012). 2017. doi: 10.1016/B978-0-12-800806-5/00012-3.

[104] W. Ning, B. Guo, Y. Yan, X. Wu, J. Wu, and D. Zhao, "Constrained multi-objective optimization using constrained non-dominated sorting combined with an improved hybrid multi-objective evolutionary algorithm," *Eng. Optim.*, vol. 49, no. 10, pp. 1645–1664, 2017, https://doi.org/10.1080/0305215X.2016.1271661

[105] Yi-Tung Kao and E. Zahara, "A hybrid genetic algorithm and particle swarm optimization for multimodal functions," *Appl. Soft Comput. Elsevier*, vol. 8, no. 2, pp. 849–857, 2008, doi: 10.1016/j.asoc.2007.07.002.

[106] A. Golchha and S. G. Qureshi, "Advance NSGA-II algorithm for solving MOO problems," *Int. J. Sci. Res.*, vol. 4, no. 9, pp. 1689–1691, 2015.

[107] M. Mitchell, *An introduction to genetic algorithms*. MIT Press, 1999.

[108] L. L. Cavalli-sforza, "Cultural versus biological inheritance: phenotypic transmission from parents to children phenotypes on children ' s phenotypes)," pp. 618–637, 1973.

Q. Dang, W. Gao, and M. Gong, "Multiobjective multitasking optimization assisted by multidirectional prediction method," *Complex Intell. Syst.*, vol. 8, no. 2, pp. 1663–1679, 2022, doi: 10.1007/s40747-021-00624-2.

[110] "Tamilselvi Selvaraj (2021). MATLAB code for constrained NSGA II - Dr.S.Baskar, S. Tamilselvi and P.R.Varshini (https://www.mathworks.com/matlabcentral/file exchange/49806-matlab-code-for-constrained-nsga-ii-dr-s- baskar-s-tamilselvi-and-p-r-varshini), MAT," p. 49806, 2021.

[111] M. Studniarski, R. Al-jawadi, and A. Younus, "An evolutionary optimization method based on scalarization for multi-objective problems," *Borzemski L., Świątek J., Wilimowska Z. Inf. Syst. Archit. Technol. Proc. 38th Int. Conf. Inf. Syst.*

Archit. Technol. – ISAT 2017. Adv. Intell. Syst. Comput. Spri, vol. 656, pp. 48–58, 2018, doi: 10.1007/978-3-319-67229-8.

[112] M. Li and X. Yao, "Quality evaluation of solution sets in multiobjective optimisation: a survey," ACM Computing Surveys (CSUR), vol. 52, no. 2

Article No.: 26, pp 1 - 38, 2019, https://doi.org/10.1145/3300148.

[113] E. Rahmo and M. Studniarski, "A new global scalarization method for multiobjective optimization with an arbitrary ordering cone," *Appl. Math.*, vol. 8, pp. 154–163, 2017, doi: 10.4236/am.2017.82013.

[114] F. H. Clarke, "Optimization and nonsmooth analysis," *John Wiley & Sons, New York*, 1983, book. http://epubs.siam.org/doi/abs/10.1137/1.9781611971309.fm

[115] K. Deb, "Comparison of multiobjective evolutionary algorithms: empirical results," *Evol. Comput.*, vol. 1994, no. 2, pp. 173–195, 2000.

[116] A. M. Gujarathi and B. V. Babu, "Improved strategies of multi-objective differential evolution (MODE) for multi-objective optimization," *Proc. 4th Indian Int. Conf. Artif. Intell. IICAI 2009*, pp. 933–948, 2009.

[117] P. Cao, Z. Fan, R. X. Gao, and J. Tang, "Solving configuration optimization problem with multiple hard constraints: an enhanced multi-objective simulated annealing approach," no. 860, 2017, [Online]. Available: http://arxiv.org/abs/1706.03141.