People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
Setif 1 University - Ferhat Abbas
Faculty of Sciences
Department of Computer Science

Bi-objective modeling and optimization by greedy stochastic algorithm

A thesis presented by:

Hayet DAHMRI

As a requirement to aim for the degree of
3^{rd} cycle Doctorate in computer science

Approved by supervisory committee:

Dr. Toumi Lyazid	Setif 1 University - Ferhat Abbas	**President**
Prof. Salim Bouamama	Setif 1 University - Ferhat Abbas	**Supervisor**
Prof. Farid Nouioua	University of Bordj Bou Arreridj	**Examinator**
Dr. Slimani Yacine	Setif 1 University - Ferhat Abbas	**Examinator**
Dr. Semchedine Moussa	Setif 1 University - Ferhat Abbas	**Examinator**
Dr. Balbal Samir	Setif 1 University - Ferhat Abbas	**Guest**

2024/2025

Abstract

Multi-objective optimization problems (MOPs) involve the simultaneous optimization of multiple, often conflicting objectives, and have wide-ranging applications in fields such as engineering, business, economics, and logistics. Most MOPs are classified as NP-Hard, meaning that finding an exact optimal solution is computationally expensive and impractical for large instances. In such cases, stochastic methods are preferred, as they offer near-optimal solutions within a reasonable time, as opposed to exact methods, which guarantee optimal solutions but require exponentially longer runtimes.

This thesis addresses a bi-objective optimization problem known as the Minimum Weight Minimum Connected Dominating Set (MWMCDS) problem. The objective is to minimize both the number of nodes (cardinality) and the total weight of the connected dominating set (CDS) in a given graph, a well-known challenge in graph theory.

To tackle this problem, three greedy stochastic algorithms are proposed. The first, Greedy Simulated Annealing (GSA), applies the simulated annealing technique with an aggregated objective function to guide the search process. The second, Improved NSGA-II (I-NSGA-II), is an enhanced version of the widely used NSGA-II algorithm, specifically adapted for multi-objective optimization. The third algorithm, Multi-objective Greedy Simulated Annealing (MGSA), introduces a new multi-objective adaptation of simulated annealing based on Pareto optimization. In all three approaches, tailored greedy heuristics are integrated to boost the efficiency of the solution process.

Experimental results, based on several performance metrics, demonstrate that the proposed algorithms outperform existing state-of-the-art methods, achieving superior results in terms of both solution quality and computational efficiency.

Keywords: multi-objective combinatorial optimization, stochastic algorithms, greedy heuristic, minimum weight minimum connected dominating set problem

Résumé

Les problèmes d'optimisation multi-objectif (POMs) nécessitent l'optimisation simultanée de divers objectifs souvent contradictoires. Ils ont de nombreuses applications dans différents domaines scientifiques, notamment l'ingénierie, le commerce, l'économie, la logistique, etc. La plupart des MOPs sont des problèmes NP-difficiles, il est donc trop coûteux en termes de calcul de trouver une solution optimale exacte s'il en existe une. Dans de telles situations, des méthodes stochastiques sont appliquées pour trouver une solution quasi-optimale dans un temps de calcul raisonnable, plutôt que des approches déterministes qui garantissent l'optimalité des solutions retournées mais dans un temps exponentiel.

Dans cette thèse, nous traitons le problème d'optimisation bi-objectif appelé problème de l'ensemble dominant connexe de cardinalité minimale et de poids minimale (MWMCDS), qui cherche à minimiser à la fois la cardinalité et le poids total du problème bien connu en théorie des graphes ; l'ensemble dominant connexe.

Trois algorithmes stochastiques gloutons sont proposés pour résoudre le problème mentionné. Le premier, GSA, est un recuit simulé standard qui utilise une fonction objective agrégée pour guider le processus de recherche. Le deuxième, I-NSGA-II, représente une version améliorée du célèbre algorithme NSGA-II dans le domaine de l'optimisation multi-objectifs. Le troisième, MGSA, est une nouvelle adaptation multiobjectif de l'algorithme de recuit simulé basée sur la technique de Pareto. Dans chacune de ces approches, une heuristique gloutonne est développée et utilisée pour améliorer l'efficacité de la résolution de problème. Les résultats expérimentaux basés sur plusieurs métriques de performance montrent que les algorithmes proposés surpassent les méthodes actuelles de pointe.

Mots-clés: optimisation combinatoire multi-objectifs, algorithmes stochastiques, heuristique gloutonne, problème d'ensemble dominant connexe de cardinalité minimale et de poids minimale

Acknowledgements

In the name of ALLAH, the Most Gracious and the Most Merciful. Alhamdulillah, I praise and thank Allah for giving me the strength and courage to complete this thesis.

First of all, I offer my sincerest gratitude to my supervisor, Pr. Salim Bouamama for his constant help and guidance. Without his careful remarks and valuable advice, this thesis would not have materialized.

Thanks are also due to the members of my dissertation committee for honoring me and accepting to judge my work.

As I would like to express my sincere thanks to all my teachers who taught me at various stages.

Dedication

To my beloved mother, for her unfailing encouragement and sacrifices throughout the study.

To my husband, I would like to thank him for energizing me with positivity and for his almost never-ending patience over the years.

To my brothers and sisters, a big thank you for their support and encouragement. May Allah bless you all.

To my wonderful daughter, the light of my life, a special thanks for always being there for me.

I dedicate this work to them, and to the memory of my dear father.

Publications

Hayet Dahmri and Salim Bouamama. *A Simulated Annealing-Based Multiobjective Algorithm for Minimum Weight Minimum Connected Dominating Set Problem*. pages 1–6. arXiv preprint, arXiv:2312.11527, 2023.

Hayet Dahmri and Salim Bouamama. *Improved NSGA-II for minimum weight minimum connected dominating set problem*. In International Symposium on Modelling and Implementation of Complex Systems, pages 248–261. Springer, 2020.

Hayet Dahmri, Salim Bouamama, and Samir Balbal. *A Greedy Simulated Annealing-Based Multiobjective Algorithm for Minimum Weight Minimum Connected Dominating Set Problem*. vol. 14, no. 5, pages 16686-16691. Engineering, Technology & Applied Science Research journal, 2024.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Overview

Many real-world problems require balancing multiple, often conflicting criteria. These are known as multi-objective optimization problems (MOPs). Unlike single-objective optimization, which seeks one optimal solution, MOPs aim to identify a set of solutions that represent trade-offs between different objectives. These solutions form the Pareto optimal set, where no single solution can improve one objective without worsening at least one other.

MOPs appear in various aspects of everyday life, with examples in fields such as engineering, business, economics, and logistics. Take buying a house, for instance: we might aim to minimize costs, the age of the house, and the distance to work or school, while maximizing property quality and neighborhood safety. On top of that, several constraints — such as budget, space requirements, access to public transport, and layout — further complicate the decision.

From this example, it's evident that finding an exact optimal solution, if it exists, would be computationally expensive and often impractical.

Solving multi-objective optimization problems (MOPs) has always been challenging due to their computational complexity and the vast solution space involved. In fact, many real-world MOPs are exponentially large, and a straightforward reduction from the knapsack problem shows that they are NP-hard to compute [1]. As a result, finding near-optimal solutions is often a practical and feasible approach for addressing these problems.

Over the past few decades, various methods have been developed to tackle MOPs, with many relying on heuristic approaches to provide approximate solutions within a reasonable computation time. A large number of these heuristics follow a greedy strategy. In addition to heuristics, metaheuristic algorithms have also gained prominence. These are stochastic, approximate methods that can efficiently find near-optimal solutions, unlike exact algorithms that guarantee optimality but require exponential time. Metaheuristics are particularly well-suited for global exploration, as they are able to navigate vast search spaces and identify promising regions in a reasonable computational time [2, 3]

This thesis presents greedy approximation approaches to solve an edge-weighted variant of the connected dominating set (CDS) problem, known as the minimum weight minimum connected dominating set (MWMCDS) problem. The objective is to minimize two criteria: the size of the CDS and its total weight. These concepts have practical applications in various domains, including wireless sensor networks, mobile ad hoc networks, and vehicular ad hoc networks [4, 5, 6, 7]. Additionally, other key applications are found in fields such as cancer therapy [8], sociology [9], and biology [10, 11].

1.2 Goals and contributions

The primary goal of this thesis is to develop novel algorithms to solve the multi-objective optimization problem MWMCDS by balancing the trade-off between two objectives: minimizing the size and total weight of the generated connected dominating set. The proposed methods are designed to deliver high-quality solutions within a reasonable computational time. The main contributions of this work are summarized as follows:

1. Greedy simulated annealing algorithm (GSA) [12]:

GSA is a standard simulated annealing algorithm enhanced by initializing with good starting solutions generated through a greedy heuristic or randomly. It explores neighboring solutions either through a greedy strategy or randomly. Additionally, GSA employs an effective temperature adjustment mechanism to guide the search. The performance of the proposed algorithm was evaluated and compared to a recent multi-objective genetic algorithm, MOGA [13], and the MCDS approach from [14]. The experimental results demonstrate the superiority of our method over the alternatives.

2. An improved non-dominated sorting genetic algorithm II (I-NSGA-II) [15]:

In this study, we introduce I-NSGA-II, an enhanced algorithm based on NSGA-II, specifically designed to address the Minimum Weight Minimum Connected Dominating Set (MWMCDS) problem. As part of the development process, three greedy heuristics were implemented, and the most effective one was selected to initialize the algorithm.

I-NSGA-II generates the initial population using either the selected greedy heuristic or random initialization. Offspring are created by applying either a

local search strategy or genetic operators, with equal probability. Fast non-dominated sorting and crowding distance mechanisms are employed to rank and select solutions for the next generation, ensuring diversity and convergence.

We compared the performance of I-NSGA-II and NSGA-II using multiple metrics, including Pareto front quality for the two objective functions, hypervolume indicator, C-metric, and execution time. The experimental results demonstrate that I-NSGA-II consistently outperforms NSGA-II, offering better convergence and computational efficiency.

3. A Multiobjective greedy simulated annealing algorithm (MGSA) [16]: MGSA is a multiobjective simulated annealing algorithm leveraging the Pareto optimization technique. It is initialized using a combined greedy heuristic that balances both weight reduction and cardinality minimization. Neighbors are generated either greedily or randomly, with equal probability. The algorithm also incorporates adaptive temperature control and solution acceptance mechanisms to enhance the search process.

After reaching the maximum number of iterations, the approximate Pareto set is further refined by eliminating redundant vertices. The performance of MGSA was compared against recent algorithms from the literature, namely MOGA [13], MCDS [14], I-NSGA-II [15], and GSA [12] on two types of datasets. The results obtained demonstrate the superiority of MGSA.

1.3 Thesis organization

The remainder of this thesis is organized as follows:

Chapter 2: Outlines the background of the thesis. It defines the optimization problem and presents its classes. Furthermore, it offers a comprehensive

overview of resolution methods, heuristics, and metaheuristics employed for addressing optimization problems.

Chapter 3: Devoted to discuss the multi-objective optimization. It defines the important concept, provides a detailed explanation of decision-making, presents some known multi-objective evolutionary algorithms, and describes well-known performance metrics used to compare multi-objective algorithms.

Chapter 4: Focuses on the minimum weight minimum connected dominating set problem. First, it recalls some basic definitions of graph theory, then presents the important dominating set variants and reports their complexity. After that, the MWMCDS problem is formally defined, and related works from the literature are discussed. Illustrative examples are provided throughout to clarify these concepts.

Chapters 5, 6, and 7: Each chapter presents in detail one of the three contributions we have developed. Specifically, Chapters 5, 6, and 7 describe the frameworks of the proposed approaches: GSA, I-NSGA-II, and MGSA, respectively. Pseudo-codes for the main algorithmic components are presented, accompanied by a detailed discussion of the experimental results.

Chapter 8: Finally, Chapter 8 concludes the thesis and states future directions in order to achieve further research studies.

Chapter 2

Background

2.1 Introduction

According to the Cambridge English Dictionary, optimization is defined as 'the act of making something as good as possible.' In mathematical terms, optimization involves finding the best solution(s) to a problem with one or more objectives. As a result, optimization problems can be categorized into two types: single-objective and multi-objective optimization problems.

To address these problems, numerous methods have been proposed in the literature, classified based on various criteria. Common classifications include enumerative methods, deterministic methods, and stochastic methods. Among these, metaheuristics — stochastic approaches — are extensively used to tackle most optimization problems and have proven effective in a wide range of applications.

This chapter is organized as follows: Sections 2 and 3 provide definitions of optimization problems and combinatorial optimization problems, respectively. Section 4 presents a classification of optimization problems. Section 5 reviews various solution methods, including enumerative, deterministic, and stochastic methods. Finally, Section 6 focuses on heuristics.

2.2 Definition of an optimization problem

A general single-objective optimization problem can be mathematically expressed as minimizing (or maximizing) an objective function f as follows [17, 18]:

$$\begin{cases} \text{optimize} & f(x) \\ \text{subject to} & \\ & x \in X \\ & g_j(x) \leq 0, \text{ for } j = 1, \ldots, l \\ & h_k(x) = 0, \text{ for } k = 1, \ldots, p \end{cases} \tag{2.1}$$

Here, $f(x)$, $g_j(x)$, and $h_k(x)$ are functions of the design vector $x = (x_1, \ldots, x_i, \ldots, x_n)^T$, where the components x_i are referred to as decision variables. X represents the search space. The functions $g_j(x)$ represent the l inequality constraints, while $h_k(x)$ represent the p equality constraints.

2.3 Definition of a combinatorial optimization problem

A Combinatorial Optimization Problem (COP) can be defined as an optimization problem where the search space consists of a finite set of feasible solutions, and an associated objective or cost function. Formally, a COP is represented by the pair $P = (S, f)$, where S is the discrete set of feasible solutions, and $f : S \to \mathbb{R}$ is the objective function to be optimized. The aim is to find a globally optimal solution $s^* \in S$, such that $f(s^*) \leq f(s)$ for all $s \in S$, assuming

a minimization problem. In many cases, finding such an optimal solution requires exploring large, complex search spaces, which is where metaheuristic approaches become especially useful due to their ability to efficiently explore and exploit these spaces.

2.4 Classification of optimization problems

Optimization problems can be classified into several categories based on different criteria. According to [19], the classification can be made with respect to the following factors:

- **Number of objective functions**: Optimization problems are classified as single-objective when there is only one objective function, or as multiobjective when involving multiple objective functions. In practice, optimization studies often utilize up to two objective functions; however, there are exceptions, such as in cases where three objectives are optimized, as seen in [20, 21].

- **Presence of constraints**: Optimization problems can be categorized into two types: constrained and unconstrained, based on the presence or absence of constraints that restrict the set of feasible solutions within the larger search space. In general, solving unconstrained problems is simpler, as there are no additional limitations to consider. However, in most practical applications, optimization problems are constrained, making them more challenging to solve due to the need to satisfy various conditions or requirements while searching for an optimal solution.

- **Function form**: The objective function in an optimization problem can be classified as either linear or nonlinear. If all constraints are linear, the problem is considered linearly constrained. On the other hand, when the functions involving the design variables are nonlinear, the problem is termed a nonlinear optimization problem. The distinction between linear and nonlinear plays a significant role in determining the complexity and methods used to solve the problem.

- **Types of design variables**: Design variables can be continuous (any real value with range), discrete (integer or specific values), or mixed-integer (a combination of both). Discrete optimization includes integer programming, where all variables are integers, and combinatorial optimization.

- **Uncertainty in values**: An optimization problem is classified as deterministic when the objective and constraint functions have explicitly defined and consistent values for a given set of design variables. On the other hand, a problem is considered stochastic if it involves uncertainty or noise in the design variables, objective functions, or constraints, causing variability in the results.

2.5 Resolution methods

To identify the Pareto-optimal set, numerous methods have been proposed in the literature, which can be categorized from various perspectives. In this thesis, we classify these methods into three main categories: enumerative methods, deterministic methods, and stochastic methods [22, 23].

2.5.1 Enumerative methods

Enumerative methods evaluate the objective function(s) systematically at every point in the search space, often referred to as brute-force or exhaustive search. These approaches are only practical for problems with a finite number of solutions, as exploring every candidate becomes unmanageable for larger problems. A key limitation is that the evaluation time grows with the number of options, making these methods inefficient for large solution spaces or when evaluations are computationally intensive [22].

2.5.2 Deterministic methods

Deterministic methods are defined by the absence of randomness, relying instead on the mathematical structure of the problem to generate a set of solutions that converge toward the global optimum. These approaches consistently yield the same output for a given input, as they follow a fixed sequence of steps, making them generally easier and faster to implement. Commonly used in nonlinear minimization problems, deterministic methods often involve iterative procedures that, after a set number of iterations, converge to the global optimum [24, 25].

2.5.3 Stochastic methods

Most optimization problems are challenging to solve using the aforementioned exploration methods, as the time complexity can become impractical or even unbounded, particularly for problems involving large or highly complex search spaces. Stochastic optimization methods are considered suitable and efficient methods that involve random variables for solving them [26]. For stochastic

problems, the random variables appear in the formulation of the optimization problem itself, which includes probabilistic objective functions or random constraints. Stochastic methods can be divided into two primary categories: *single-solution search approaches* and *population-based search approaches*. The first group returns a new feasible solution at each step during the search, one of the well-known examples of these approaches is the simulated annealing algorithm [27]. The second category involves exploring a set of solutions (population) simultaneously in the search space, aiming toward the optimal solution, though without guaranteeing its attainment. Evolutionary algorithms, such as genetic algorithms, are among the most well-known population-based search methods.

2.6 Heuristics

The term "heuristic" is derived from the ancient Greek word *heuriskein*, meaning the art of discovering new strategies for solving problems. Technically, heuristic is a method of problem-solving used to find approximate solutions when it's not feasible to exhaustively search through every possible solution due to computational constraints or the large size of the solution space. Heuristics enable a simple approach to traverse through a large space quickly without attaining an optimal solution; instead, the solution attained is approximate. A survey of the literature on heuristic classification reveals that the most widely recognized categories are local search heuristics, constructive heuristics, and metaheuristics [28].

2.6.1 Constructive heuristics

Constructive heuristics are generally the quickest approximate algorithms for solving combinatorial optimization problems [29]. They are straightforward, easy to implement, and do not require extensive computational tuning. These algorithms begin with an empty solution and progressively build upon it until a complete solution is formed. A well-known example of a constructive heuristic is the greedy algorithm.

It is essential to know the difference between constructive methods which extend empty solution until get a full solution, and local search techniques which refine a complete solution through local moves [29]. A pseudocode for a constructive heuristic is shown in Algorithm 1.

Algorithm 1 Constructive Heuristic Algorithm

1: $s \leftarrow \emptyset$
2: Determine $\Re(s)$
3: **while** $\Re(s)$ is not empty **do**
4: $\quad c \leftarrow$ SelectFrom$(\Re(s))$
5: $\quad s \leftarrow s \cup \{c\}$
6: $\quad$ Update $\Re(s)$
7: **end while**
8: **Output:** constructed solution s.

Referring to Algorithm 1, the process starts by identifying the set of possible extensions for each feasible solution s. From this, a set of solution components, $\Re(s)$, is generated to expand s. At each iteration, one of the potential extensions is selected, and this continues until $\Re(s)$ is empty, indicating either that s is a complete solution or that it is a partial solution that cannot be extended further into a feasible one.

2.6.1.1 Greedy heuristics

Greedy heuristics are a type of constructive heuristic that use a weighting function in the procedure SelectFrom $\mathfrak{R}(s^p)$ (see Algorithm 1). At each step, a locally optimal solution is selected and then used as input for the next stage. Although a greedy algorithm does not guarantee an optimal solution, it can efficiently provide a close approximation. For instance, in the traveling salesman problem, which has high computational complexity, a greedy approach involves visiting the nearest unvisited city at each step. While this method may not yield the best solution, it completes in a reasonable number of steps, making it practical for problems where finding the exact optimal solution would require an excessive number of steps.

2.6.2 Local search heuristics

Local search methods have proven highly effective in solving complex combinatorial optimization problems. This strategy involves making small, incremental changes to improve the current solution by exploring its neighborhood. The current solution is replaced by the best neighboring solution, and the search continues from there until no further improvement can be found, indicating that a local minimum has been reached. Formally, the neighborhood is defined as [29]:

A neighborhood structure is a function $N : S \to 2^S$, which assigns to each $s \in S$ a set of neighbors $N(s) \subseteq S$, where $N(s)$ is the neighborhood of s.

Neighborhood structures are typically defined by the specific modifications applied to a solution s to generate all its neighbors. The process of applying these modifications, producing a neighbor $s' \in N(s)$, is known as a move.

The most basic form of local search is iterative improvement. This begins with an initial solution, followed by selecting an appropriate neighborhood to replace the current solution. The process continues until a local optimum is found. Algorithm 2 outlines the general local search algorithm.

Algorithm 2 Generic local search algorithm

```
1: input: Initial solution s_0
2: Output: current_solution (local optima)
3: current_solution ← s_0
4: while Termination condition not satisfied do
5:     Generate N(current_solution)
6:     s' ← SelectNeighbor(N(current_solution))
7:     if f(s') < f(current_solution) then
8:         current_solution ← s'
9:     end if
10: end while
```

2.6.2.1 Selection of the neighborhood

To enhance the selection process for identifying a suitable neighbor, several strategies are commonly employed in the literature, including first improvement, best improvement, and random selection strategies [30].

- **First Improvement**: This approach involves generating neighboring solutions incrementally. The first neighbor that offers a lower cost than the current solution is chosen. In this method, the neighborhood is explored in a fixed order, allowing for a quick assessment.

- **Best Improvement**: This method conducts a comprehensive evaluation of all neighboring solutions to find the one with the lowest cost, making it particularly effective for minimization tasks. However, this thorough exploration can be computationally demanding, particularly in larger neighborhoods.

- **Random Selection**: In contrast to the previous strategies, this method selects a neighboring solution at random. The aim is to improve the current solution's quality based on the associated cost, introducing an element of randomness into the search process.

To optimize both the quality of solutions and search efficiency, it may be beneficial to implement the first improvement strategy when starting with a randomly generated solution, while employing the best improvement strategy when the initial solution is derived through a greedy method.

2.6.2.2 Escaping from Local Optima

Local search algorithms can be effective when the search space contains few local optima or when the local optima exhibits similar qualities. However, a significant limitation of these algorithms is their tendency to converge on a local optimum, potentially leading to suboptimal solutions (see Figure 2.1).

Additionally, the algorithm's success can heavily depend on the initial solution chosen. To address these challenges, various alternative algorithms were developed in the 1980s aimed at enhancing search performance, as illustrated in Figure 2.2.

2.6.3 Metaheuristics

Metaheuristic is a term introduced by Glover in [32] to design a new kind of algorithm devised to tackle difficult optimization problems. The word metaheuristic consists of two Greek words, meta which means "beyond" or "in the sense of an upper level", and heuristic which means "to find". Thus, metaheuristics can be viewed as approaches that combine basic heuristic techniques

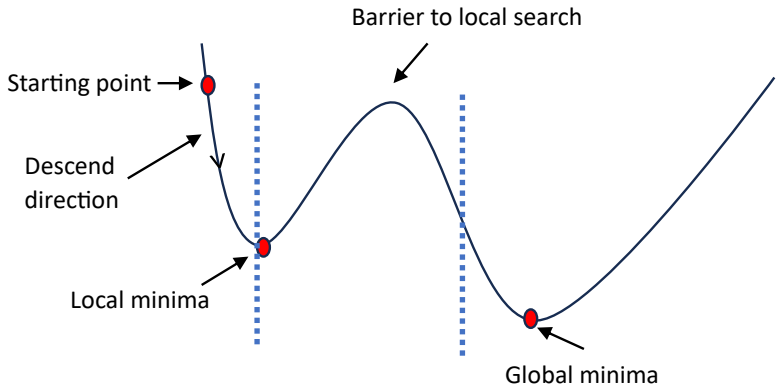


FIGURE 2.1: Example of a problem with local minima Ref[31].

in higher-level frameworks to solve combinatorial optimization problems efficiently and effectively through iterative attempts.

Metaheuristics are approximate optimization techniques designed to find high-quality solutions for a variety of computationally complex problems that are inefficient to solve with exact methods. Unlike heuristics, which are problem-specific and often tailored to a particular class of problems, metaheuristics are more general-purpose and flexible, making them applicable to a wide range of problems. Their primary aim is to effectively escape local minima, allowing further exploration of the search space and the discovery of potentially better solutions. While they do not guarantee finding a global optimum, metaheuristics typically yield near-optimal solutions at relatively low computational costs [3, 33, 34].

Successful metaheuristics depend on two key attributes: diversification, which focuses on exploring the search space, and intensification, which emphasizes exploiting the best solutions discovered. Diversification aims to uncover regions with potential near-optimal solutions, while intensification hones on the current top candidates, refining the search for even better results. Achieving a

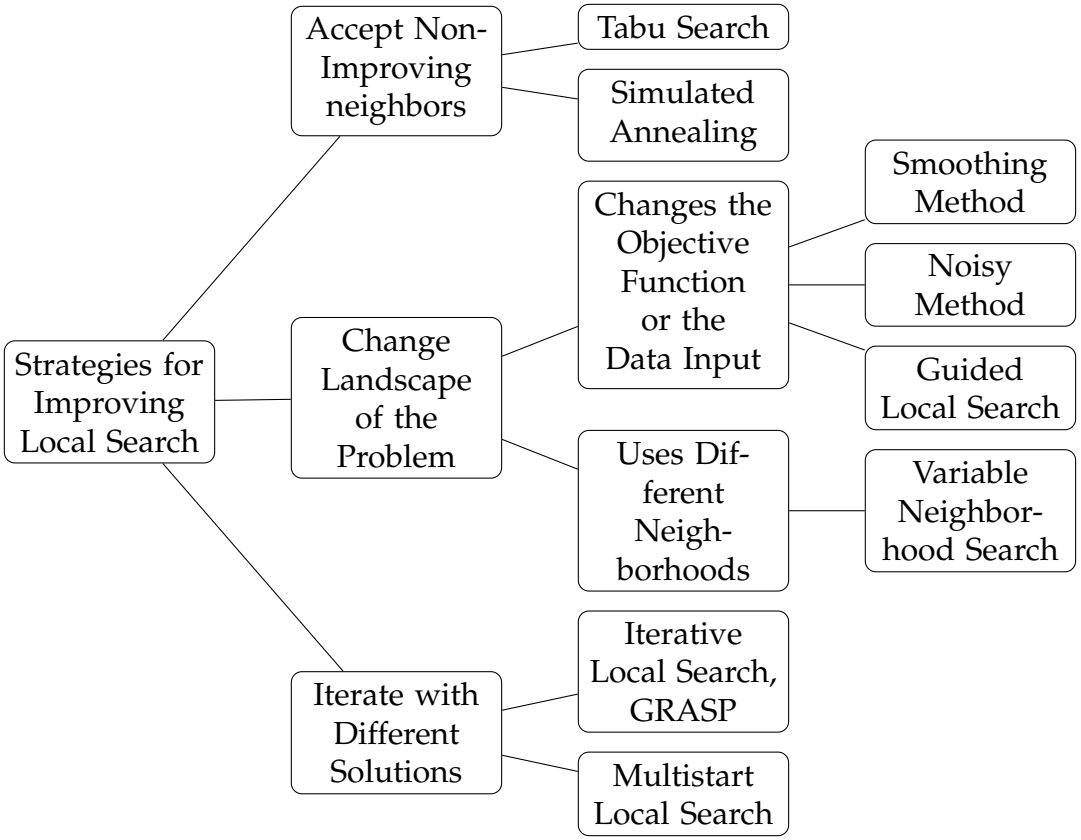


FIGURE 2.2: Family of strategies for escaping from local optima Ref[30].

good balance between these two strategies is essential for increasing the likelihood of reaching global optimality [35].

Some of the more popular research methods can be distributed on the diversification-intensification spectrum line as shown in Figure 2.3.

The figure shows that random search prioritizes diversification by generating random solutions throughout the search space without using search memory. In contrast, local search focuses on intensification, selecting the best solution that improves upon the current one. Population-based and single-solution-based methods fall between these two approaches. Single-solution-based approaches typically emphasize intensification, while population-based approaches place a stronger focus on diversification [29, 30].

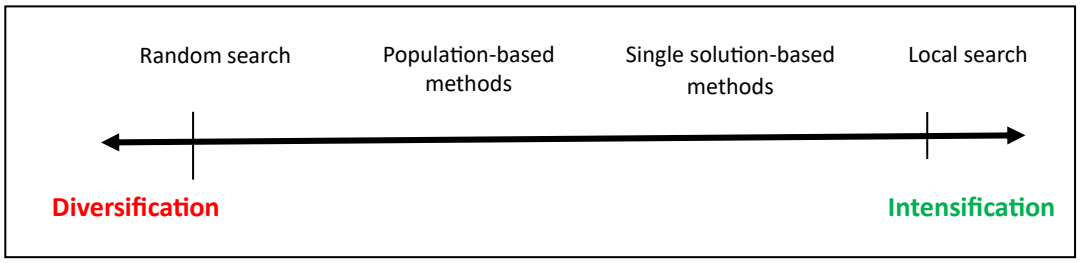


FIGURE 2.3: Balancing the diversification and intensification spectrum in metaheuristics Ref [30].

2.6.3.1 Classification of metaheuristics

There are several ways to categorize metaheuristics, in this section, we present the most common ones [29, 30, 35].

- **Nature-inspired versus non-nature inspired**: Many algorithms are designed to mimic natural processes in order to efficiently find high-quality solutions for combinatorial optimization problems. Nature-inspired algorithms, such as neural networks, genetic algorithms, simulated annealing, ant colony optimization, and bacterial foraging optimization, are based on natural phenomena. On the other hand, non-nature inspired algorithms include techniques like tabu search, variable neighborhood search, and iterated greedy algorithm.

- **Single-solution search versus population-based search**: The key distinction between these methods is the number of candidate solutions managed at once. Single-solution-based methods, or trajectory-based methods, concentrate on one solution at a time, exploring the search space by following specific paths. Examples include tabu search, iterated local search, simulated annealing, and variable neighborhood search. In

contrast, population-based algorithms evolve a set of solutions simultaneously, such as in evolutionary algorithms, ant colony optimization, particle swarm optimization, bee colony optimization, and artificial immune systems.

- **One versus various neighborhood structures**: The majority of metaheuristics work on a single neighborhood structure, maintaining a stable fitness landscape topology during the search. However, this approach may constrain the diversification of solutions. To address this limitation, some metaheuristics adopt a set of neighborhood structures by swapping between different fitness landscapes, examples including variable neighborhood search and iterated local search.

- **Memory usage versus memoryless methods**: A key characteristic in classifying metaheuristics is whether they utilize memory. Memoryless methods, such as local search, simulated annealing, and greedy randomized adaptive search procedure, do not store or retrieve search history, relying solely on the current solution and search trajectory for decision-making during the process. Other metaheuristics like tabu search incorporate a memory that contains valuable information such as visited elite solutions etc. Two types of memories exist, short-term memory which prevents revisiting recently discovered solutions and avoids recycling, and long-term memory which helps to ensure diversification and intensification aspects.

- **Deterministic versus stochastic**: Deterministic methods make decisions based on fixed rules, thus for the same initial solution, the same final solution will be obtained. For instance local search and tabu search. On the other hand, stochastic metaheuristics apply some random rules leading to

different final solutions from the same initial solution. Examples include simulated annealing, evolutionary algorithms, etc.

- **lIterative versus greedy**: Iterative algorithms begin with a complete solution and refine it through successive iterations. In contrast, greedy algorithms start with an empty solution and progressively build upon it by adding elements according to predefined criteria until a full solution is reached.

2.6.3.2 Simulated annealing

Simulated annealing derives its name from the physical annealing process of solids, where a crystalline material is heated to its melting point and then slowly cooled to achieve the optimal crystal lattice configuration, minimizing defects and reaching its lowest energy state. If the cooling is gradual, the result is a solid with high structural integrity. Simulated annealing draws an analogy between this thermodynamic process and the search for global minima in optimization problems [36].

Simulated annealing is a widely studied local search metaheuristic, particularly effective for discrete optimization problems and, to a lesser extent, continuous ones. One of its key features is its ability to escape local optima by allowing hill-climbing moves (i.e., moves that temporarily worsen the objective function) in the search for a global optimum. Its ease of implementation, convergence properties, and use of hill-climbing moves have contributed to its popularity as an optimization technique that can [37]:

(a) Handle cost functions with arbitrary levels of nonlinearity, discontinuity, and stochasticity.

(b) Manage various boundary conditions and constraints imposed on these cost functions.

(c) Be relatively simple to implement with minimal coding compared to other nonlinear optimization algorithms.

(d) Offer statistical assurances that the global optimum can be reached.

Survey articles providing a comprehensive overview of the theoretical advancements and practical applications of simulated annealing include [38, 39, 40, 41, 42, 43]. Notable books dedicated solely to this topic include [44, 45].

The basic iteration

At each iteration of the simulated annealing process, the algorithm evaluates a neighboring solution s' based on the current solution s. The system then probabilistically determines whether to move to this new solution s' or remain at the current state s. If the neighboring solution s' improves the objective function, it is accepted outright. However, if s' results in a worse outcome, the algorithm may still accept it with a certain probability, which decreases as the system's temperature cools. These probabilities eventually guide the system towards lower energy states. Typically, this process is repeated until the system achieves a sufficient state for the application or until the allocated computational resources are depleted.

Exploring Neighboring Solutions

In optimization, neighboring solutions are derived by making slight modifications to a given solution. For example, in the traveling salesman problem, where the solution is represented as a permutation of cities, a neighboring solution is created by swapping the positions of two cities. These small changes,

or "moves," generate a new set of potential solutions that are close to the original one. The idea is that these moves allow for gradual improvements, such as finding more efficient city connections in the traveling salesman scenario.

However, simply moving from one better neighbor to the next may only lead to a local optimum rather than the global best solution. The simulated annealing algorithm overcomes this by not only favoring better neighbors but also occasionally accepting worse ones. This strategy helps the algorithm explore the broader solution space and escape local optima, increasing the chance of finding the global optimum if given enough time.

Metropolis acceptance criterion

The probability of transitioning from the current solution s to a neighboring solution s' is determined by the Metropolis acceptance criterion [46]. The objective is to minimize the energy (or objective function) of the system, where solutions with lower energy are considered better. The candidate solution s' is accepted with a probability given by the function $P(f(s), f(s'), t)$, which depends on the objective function values of the current and candidate solutions, as well as the temperature t.

$$P(f(s), f(s'), t_i) = \begin{cases} \exp\left[-\frac{f(s')-f(s)}{t_i}\right] & \text{if } f(s') - f(s) > 0 \\ 1 & \text{if } f(s') - f(s) \leq 0 \end{cases}.$$

Let t_i represent the temperature parameter at iteration i, which controls the probability of accepting worse solutions in the simulated annealing process such that

$$t_i > 0 \text{ for all } i \text{ and } \lim_{i \to \infty} t_i = 0$$

Note that when i increases, t_i typically decreases, allowing the algorithm to

focus more on refining the current solution while limiting the acceptance of less optimal neighbors. The acceptance probability drives the search in simulated annealing. A slow temperature decrease allows the system to reach equilibrium at each iteration.

Change temperature

The temperature controls how sensitive the solution s is to changes in the system's energy. The algorithm begins with a high initial temperature t_0, which decreases according to a user-defined annealing schedule until it reaches $t_f = 0$. This process enables broad exploration of the search space, gradually narrowing to regions of lower energy, akin to steepest descent. While the probability of converging to a global optimum approaches 1 as the annealing schedule increases [47], in practice, the time required often exceeds that of an exhaustive search.

Statement of the algorithm

Algorithm 3 describes the general process of simulated annealing algorithm. It iterates $M_0 + M_1 + ... + M_i$ times, where i represents the value for t_i that satisfies specific stopping criteria (e.g., reaching a predetermined total number of iterations or finding a solution of desired quality). In addition, if $M_i = 1$ for all i, then the temperature changes at each iteration.

Single-objective Versus Multi-objective Problems

Originally developed for single-objective combinatorial optimization, simulated annealing has since been adapted for multi-objective problems [43]. Multi-objective simulated annealing is easy to implement and can generate a Pareto-optimal set in one run with minimal computational cost. Additionally, its performance remains robust regardless of the shape of the Pareto front, which often

Algorithm 3 Simulated Annealing Algorithm

Input: Initial solution $s \in S$, Temperature change counter $k = 0$, Temperature cooling schedule t_k, Initial temperature $T = t_0 \geq 0$, Repetition schedule M_k.
repeat
  Set repetition counter $m = 0$.
  repeat
    Generate a solution $s' \in N(S)$.
    Calculate $\Delta_{s,s'} = f(s') - f(s)$.
    if $\Delta_{s,s'} \leq 0$ **then**
      $s \leftarrow s'$.
    else
      $s \leftarrow s'$ with probability $\exp\left(-\frac{\Delta_{s,s'}}{t_k}\right)$.
    end if
    $m \leftarrow m + 1$.
  until $m = M_k$
  Update temperature: $T \leftarrow t_k(T, k)$.
  $k \leftarrow k + 1$.
until stopping criterion is met.
Output: Optimized solution s^*.

poses challenges for mathematical programming methods.

Serafini [48] first introduced the multi-objective simulated annealing (MOSA) algorithm by modifying the decision acceptance criteria. Since then, various criteria have been explored to improve the acceptance of non-dominated solutions. A selection rule was proposed to focus exploration on these solutions. Ulungu et *al.* [49] later developed an alternative MOSA, followed by a comprehensive version [50], which uses a weighted aggregation function for evaluation. While the algorithm operates with a single current solution, it maintains a record of all non-dominated solutions found during the search.

Suppapitnerm and Parkes [51] introduced SMOSA, a simulated annealing-based approach for multi-objective problems. The algorithm explores a single

solution per iteration, adjusting the temperature adaptively based on the objective function values. All non-dominated solutions are stored. A novel acceptance probability is proposed, using an annealing schedule with separate temperatures for each objective. If a solution is potentially Pareto-optimal, it is accepted; otherwise, a multi-objective acceptance rule is applied. For further examples of multi-objective simulated annealing algorithms, see [52, 53, 54, 55, 56].

2.6.3.3 Genetic algorithm

The genetic algorithm is an optimization and research method inspired by genetics and natural selection and evolution principles, in which the fittest individuals are selected to produce offspring of the next generation. The genetic algorithm (GA) was initially introduced by John Holland in 1975, since then several changes have been made to this formulation.

In the process of natural selection, the most suitable individuals from a population are chosen to generate offspring (next generation) that inherit the characteristics of their parents. If the parents have better fitness, their offspring will outperform their parents and have a better chance of survival. This process keeps repeating itself and will eventually find the generation with the fittest individuals. The genetic algorithm translates this concept to problem-solving scenarios, where a number of solutions are evaluated, and the best among them is selected. Six phases are considered: initial population, fitness function, selection, crossover, mutation, and termination.

Initial population

The genetic algorithm starts with a set of individuals called a population. Each individual represents a solution to the problem to solve, and it is characterized

by a set of parameters (variables) known as genes. The genes are linked in a chain (string), forming a chromosome (solution). Normally, an individual's genetic set is represented by a string in alphabetical terms wherein binary values (a string of 1 and 0) are used.

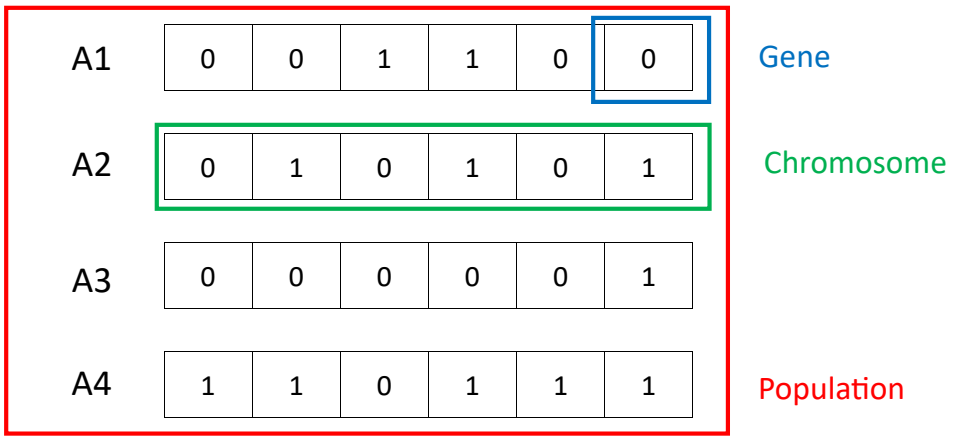


FIGURE 2.4: Population, chromosome, and gene.

Fitness function

The fitness function serves as the objective function to be optimized, it provides the mechanism for evaluating the individuals by assigning a fitness score to each, which guides the selection phase to decide whether an individual will be selected for the next generation.

Selection

Selection shapes the natural survival-of-the-fittest mechanism: the fittest solutions survive, while the weakest die. Two pairs of individuals are chosen for reproduction based on their fitness scores. A commonly used method for selection is the roulette-wheel method, where the probability of selecting a particular individual is proportional to its fitness score.

Fig. 2.5 illustrates a simple example of roulette-wheel selection (RWS) with

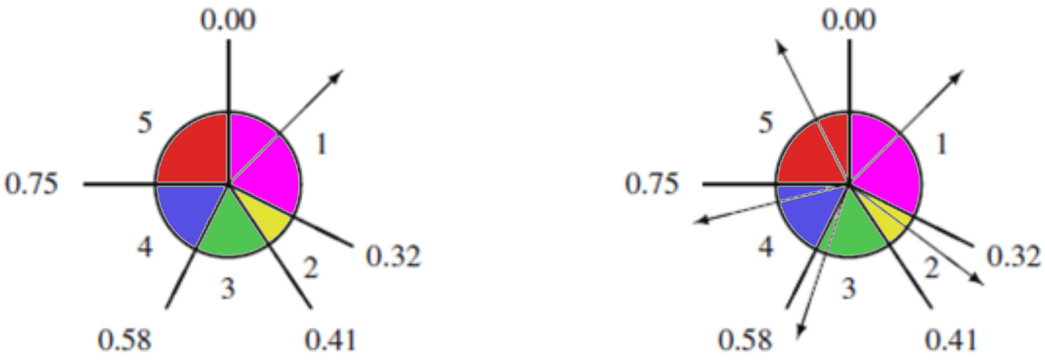


FIGURE 2.5: Example of roulette-wheel selection (RWS).

a population of five solutions. Each individual's probability of selection is proportional to the area of a sector of the roulette wheel. The numbers labeled on the wheel spokes represent cumulative probabilities for use by the pseudo-random number generator, which are used to choose solutions for reproduction. For example, the number 0.13 selects solution 1, and 0.68 selects solution 4. On the left side, the standard roulette wheel selection with a single pointer to spin five times is depicted. On the right side, the stochastic universal selection (SUS) is shown, which uses five linked equidistant pointers, resulting in five selections with one spin. Determining the correct number for a specified pseudo-random number r involves searching an array of values that bracket r. For a population size M, this method can be implemented in $O(\log M)$ time. Nevertheless, the method is characterized by high stochastic variability, which can lead to strong divergences of the actual number of times the solution s is chosen, N_s, and its expected value, $E[N_s]$. In order to eliminate this effect, sampling without substitution can be used to ensure that at least the integer part of $E[N_s]$ is approached and the non-integer parts are evenly distributed over the random sampling. Baker's Stochastic Universal Selection (SUS) [57] provides

an effective implementation of this concept, as demonstrated by Hancock's empirical study [58], showcasing its superiority in practice.

Crossover

In the crossover phase, parents swap some of their genes to generate new combinations. An example of a one-point crossover is provided below. Given the parents $A1$ and $A2$ with a crossover point at position 3, offspring $A5$ and $A6$ are formed by interchanging the parental genes until the crossover point is reached.

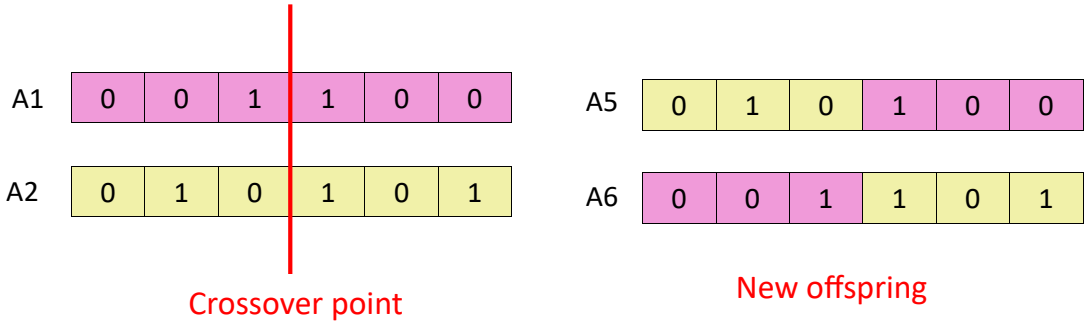


FIGURE 2.6: Crossover operation.

Mutation

In certain newly generated offspring, there is a chance that some of their genes can mutate with a low random probability. This means that a gene (or a subset of genes) is selected randomly and the value of its allele will change (changing from 0 to 1 or vice versa). Mutation serves to maintain diversity in the population and avoid premature convergence.

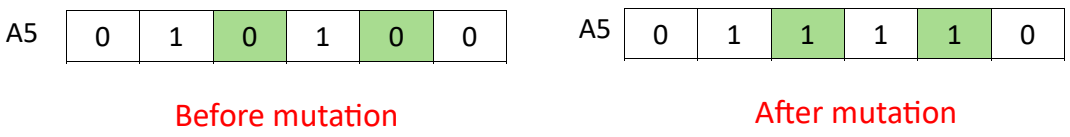


FIGURE 2.7: Mutation operation.

Termination

Genetic Algorithm is a stochastic search method that can in principle run indefinitely, to terminate its execution, a stopping criterion needs to be specified. Common approaches include setting a limit on the number of generations, or the computer clock time, or monitoring population diversity and stopping when it falls below a certain threshold.

Statement of the algorithm

Algorithm 4 outlines the process of a genetic algorithm. It starts with the initialization phase and then repeats the selection, crossover, and mutation operations until a stopping criterion is met. Ultimately, the algorithm outputs the best solution found.

Algorithm 4 Genetic Algorithm

input: Population size N
$population \leftarrow$ INITIALIZE(N)
repeat
    for each $individual$ **in** $population$ **do**
        EVALUATE$(individual)$
    end for
    $parents \leftarrow$ SELECTION$(population)$
    $offspring \leftarrow$ CROSSOVER$(parents)$
    MUTATION$(offspring)$
    $population \leftarrow$ REPLACEMENT$(population, offspring)$
until stopping criterion is met
Output: Optimized solution $individual^*$.

Single-objective Versus Multi-objective Problems

Genetic algorithms are suitable for multiobjective optimization because they operate on a population of individuals and therefore they capture the whole Pareto front in a single run. Furthermore, they require minimal prior knowledge of the problem compared to other classical methods. As a result, numerous approaches and variations of genetic algorithms that deal with more than

one objective have been published in the technical literature.

Schaffer [59] was the first that introduce a multiobjective genetic algorithm named VEGA (Vector Evaluating Genetic Algorithm). VEGA uses GA's selection technique to generate non-dominated individuals, where each objective is identified as the selection indicator for a portion of the population. However, this approach overlooks the concept of Pareto optimization and it results in poor coverage of the Pareto frontier.

Non-dominated sorting was introduced by Goldberg [60] to categorize the population according to Pareto optimality. Initially, non-dominated individuals are identified, assigned a rank of r (where r is the iteration number), and eliminated from the population. This step is repeated with the remaining individuals until the whole population has been ranked.

Many other multiobjective genetic algorithms were proposed including [61, 62, 63, 64]. In general, these algorithms differ from each other in the following points [65, 66, 67]: the fitness assignment procedure, the approach to maintaining the diversity of solutions, and whether Pareto solutions are stored in a different set than the population. Fonseca and Fleming [61] have divided multiobjective genetic algorithms into non-Pareto and Pareto-based approaches, For an overview of genetic algorithms in multiobjective optimization, see [68].

2.7 Conclusion

In this chapter, we provide a thorough review of optimization problems and the various approaches used to address them. We started by establishing fundamental definitions for optimization and combinatorial optimization problems, followed by an examination of their classification based on distinct attributes.

We then looked at the three main resolution methods: the enumeration methods, deterministic methods, and stochastic methods. Additionally, the chapter covered heuristics extensively with particular emphasis on those used in our research work.

Chapter 3

Multi-objective optimization

3.1 Introduction

Multi-objective optimization (MOP) differs from traditional single-objective optimization, which seeks a single optimal solution, by aiming to identify a set of solutions that balance multiple, often conflicting objectives. This complexity arises from the inherent trade-offs between objectives, necessitating advanced strategies to effectively explore the solution space. These strategies produce a set of non-dominated solutions, which must be evaluated using specialized performance metrics. This chapter provides a comprehensive overview of MOP, discussing its definition, key concepts like Pareto dominance and optimality, decision-making techniques, and leading evolutionary algorithms for addressing MOPs. Additionally, the chapter outlines the most commonly used performance metrics to assess the quality of these solutions.

3.2 Definition of a multi-objective optimization problem

A multi-objective optimization problem seeks to optimize several conflicting objectives simultaneously. Rather than focusing on a single best solution, the goal is to find a set of solutions that represent trade-offs among the objectives, such as minimizing cost while maximizing quality. These solutions are known as Pareto-optimal solutions, named after the economist Vilfredo Pareto [69], and they reflect different compromises where no objective can be improved without worsening another.

Formally, a multi-objective optimization problem is defined as follows [70, 18, 71]: :

$$\begin{aligned} \textbf{Optimize} \quad & F(x) = (f_1(x), \ldots, f_m(x))^T, x \in \Omega \\ \textbf{subject to} \quad & g_j(x) \leq 0, \quad j = 1, \ldots, l, \\ & h_k(x) = 0, \quad k = 1, \ldots, p. \end{aligned} \tag{3.1}$$

Here, F represents the vector of m objective functions to be optimized, while $x = (x_1, \ldots, x_n) \in \Omega$ is the decision variable vector, with Ω being the feasible set. The functions $g_j(x)$ correspond to the l inequality constraints, and $h_k(x)$ represent the p equality constraints. Each decision variable x_i is typically bounded by lower and upper limits, $x_{li} \leq x_i \leq x_{ui}$, for $i = 1, \ldots, n$.

Figure 3.1 presents a multi-objective optimization scenario involving three decision variables and two objective functions. The feasible search space is mapped onto the corresponding objective space.

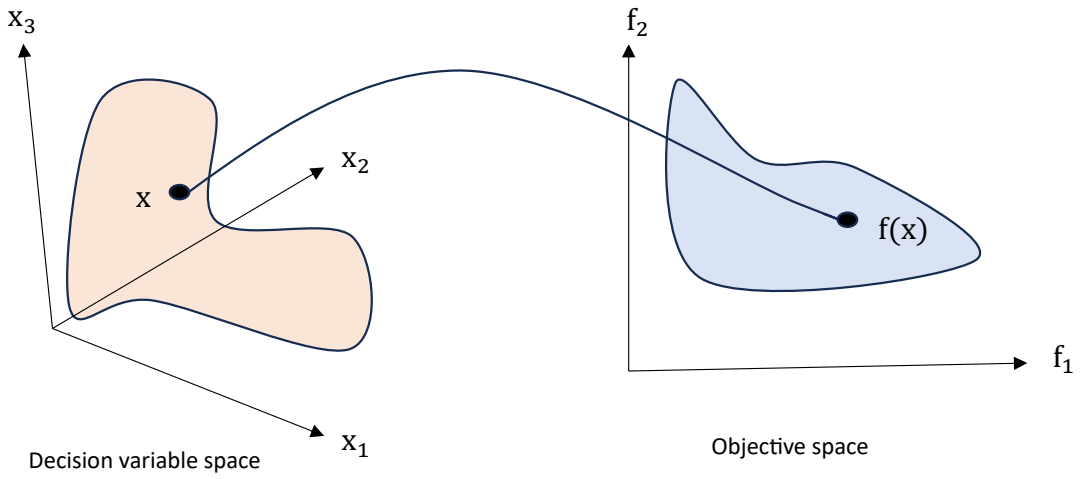


FIGURE 3.1: A multi-objective optimization problem having three variables and two objectives.

3.3 Pareto dominance and optimality

The primary aim of multi-objective optimization is to seek an optimal solution for the given problem. This involves finding solutions that optimize multiple, often conflicting objectives simultaneously. To overcome this challenge, most multi-objective optimization algorithms utilize the concept of Pareto dominance. This principle compares two solutions to assess whether one outperforms (dominates) the other. Below, we provide definitions of the terminology pertinent to Pareto dominance and optimality. For these definitions, we refer to [72].

Definition 3.1. Pareto Dominance

In a multi-objective optimization problem, given two feasible solutions x_a and x_b, we say that x_a *dominates* x_b (denoted as $x_a \preceq x_b$) if the following conditions hold:

$$f_i(x_a) \leq f_i(x_b) \quad \forall i \in \{1, \ldots, m\}, \quad \text{and} \quad \exists k \in \{1, \ldots, m\} \text{ such that } f_k(x_a) < f_k(x_b)$$

Definition 3.2. Pareto-Optimal Solution

A solution x^* is considered *Pareto-optimal* if no other solution exists that dominates it:

$$\nexists x \in \Omega \text{ such that } x \preceq x^*$$

Definition 3.3. Pareto-Optimal Set

The *Pareto-optimal set* X^* is the set of all solutions that are not dominated by any other feasible solutions. This set is defined as:

$$X^* = \{x^* \in \Omega \mid \nexists x \in \Omega \text{ such that } f(x) \preceq f(x^*)\}$$

Definition 3.4. Pareto-Optimal Front

The *Pareto-optimal front*, denoted by $\mathcal{F}^*$, refers to the set of all objective function values corresponding to the Pareto-optimal solutions. It is defined as:

$$\mathcal{F}^* = \left\{ f(x^*) = (f_1(x^*), \ldots, f_m(x^*))^{\mathrm{T}} \mid x^* \in X^* \right\}$$

Figure 3.2 illustrates the dominance relationship between solutions and Pareto optimality, where both objectives are to be minimized.

3.4 Decision making

The optimal Pareto set in most multi-objective optimization problems consists of a vast, or even infinite, number of solutions. However, in practice, only a single solution is usually selected from this set. The Decision Maker (DM) is the person tasked with selecting the most suitable solution, where their preferences are taken into consideration in order to determine a total order among the Pareto set's elements. There are several ways to include DM preferences.

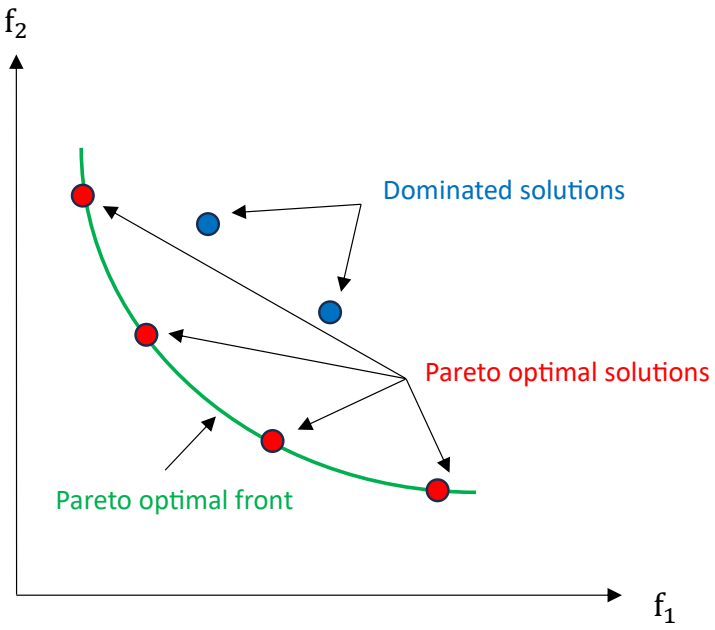


FIGURE 3.2: Illustration of the Pareto optimality and dominance relations between solutions.

The decision maker (DM) might, for instance, rank the goals by their importance. Alternatively, they could sample a section of the Pareto front and select a solution from that subset.

Methods for solving multi-objective problems are typically categorized based on when the DM provides preference information. As a result, various decision support techniques have been developed to incorporate DM preferences into the optimization process [73]. The following classification illustrates these methods:

3.4.1 Priori methods

Priori techniques are widely recognized for their broad applicability and simplicity. These methods combine multiple objectives into a single objective function, using various aggregation techniques such as the weighted sum, fuzzy

integrals, and Tchebychev functions, as discussed in the literature [18]. In the weighted sum approach, the decision maker assigns priorities to each criterion to create a single objective function. The optimization process then focuses on finding the best solution for this combined objective.

3.4.2 Posteriori methods

Posterior methods aim to identify Pareto-optimal solutions without requiring direct input from the decision maker (DM) during the optimization process. As a result, these methods present the full set of Pareto solutions, allowing the DM to select the most suitable option afterward. By eliminating the need to model the DM's preferences or have prior knowledge of the problem, these techniques offer flexibility. However, the large number of solutions generated can make it challenging for the DM to effectively analyze the optimal Pareto set.

3.4.3 Interactive methods

Interactive methods are designed to actively involve the decision maker (DM) throughout the entire optimization process. The DM engages with the method at each step, clearly expressing preferences, and refining them as the process continues. This iterative approach proceeds until the DM is satisfied with the solution.

Posterior methods are often considered the most effective for obtaining a set of optimal solutions, as the decision is made after the optimization process has been completed. This allows the DM to make an informed choice based on the full range of available alternatives [26].

Figure 3.3 presents a categorization based on the DM preference of various techniques used in multi-objective optimization.

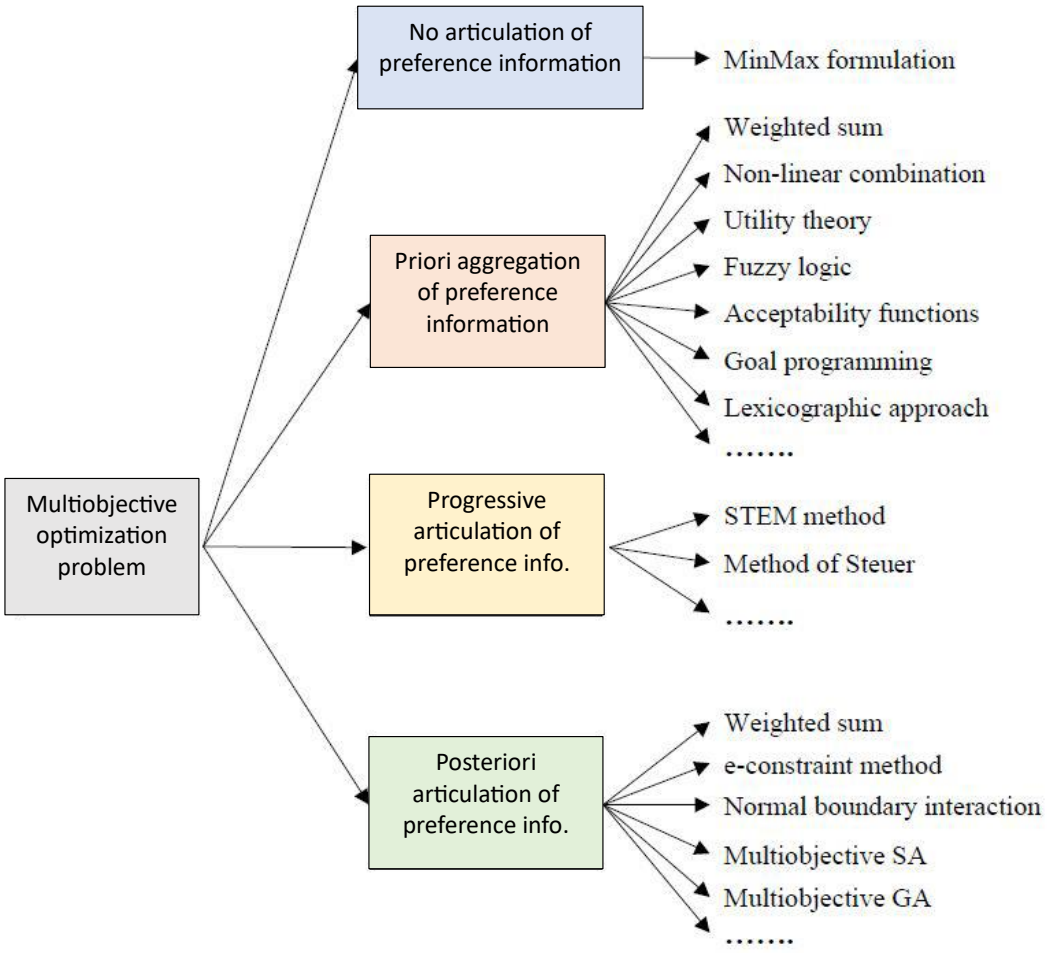


FIGURE 3.3: A classification of some methods for multi-objective optimization ref [74].

3.5 Multi-Objective Evolutionary Algorithms

When solving an MOP, the objective is not only to find the Pareto set but also to ensure that its solutions are well-distributed. Evolutionary Algorithms (EA) are metaheuristics that have proven highly effective in solving MOPs by discovering a diverse set of Pareto optimal solutions within a single run [71, 18, 75]. Over the years, there has been a growing interest in the application of EAs for MOPs, leading to the development of multi-objective evolutionary algorithms (MOEAs) [59, 76], including the Multi-Objective Genetic Algorithm

(MOGA) [77]; the Niched Pareto Genetic Algorithm [62]; the Non-dominated Sorting Genetic Algorithm (NSGA) [63], along with its successors NSGA-II [78] and NSGA-III [79]; the Strength Pareto Evolutionary Algorithm (SPEA) [80] and its improved version SPEA2 [81]; and the Multi-Objective Evolutionary Algorithm based on Decomposition (MOEA/D) [82]. This section will delve into a detailed presentation of the popular MOEAs.

3.5.1 Non-dominated sorting genetic algorithm II (NSGA-II)

The Non-dominated Sorting Genetic Algorithm II (NSGA-II), developed by Deb et *al.* in 2002 [78], is an improved iteration of the original Non-dominated Sorting Genetic Algorithm (NSGA), introduced by Srinivas and Deb in 1994 [63].

The fundamental framework of NSGA-II operates as follows: First, it employs Pareto dominance to classify individuals within the parent population based on their non-domination levels. Next, evolutionary operators — selection, crossover, and mutation — are applied to create an offspring population of the same size as the parent population. This combined population (parents and offspring) is then sorted into fronts according to the dominance ranks of the individuals. Finally, the selection of individuals for the next generation is conducted from this combined population, considering both the ranked fronts and the crowding distance. This mechanism is applied to preserve diversity within the population.

NSGA-II procedure is characterized by three key elements: using an elitist principle, emphasizing non-dominated solutions, and implementing an explicit mechanism to preserve diversity.

Figure 3.4 presents a diagram illustrating the workflow of the NSGA-II algorithm, and algorithm 5 outlines its pseudo-code.

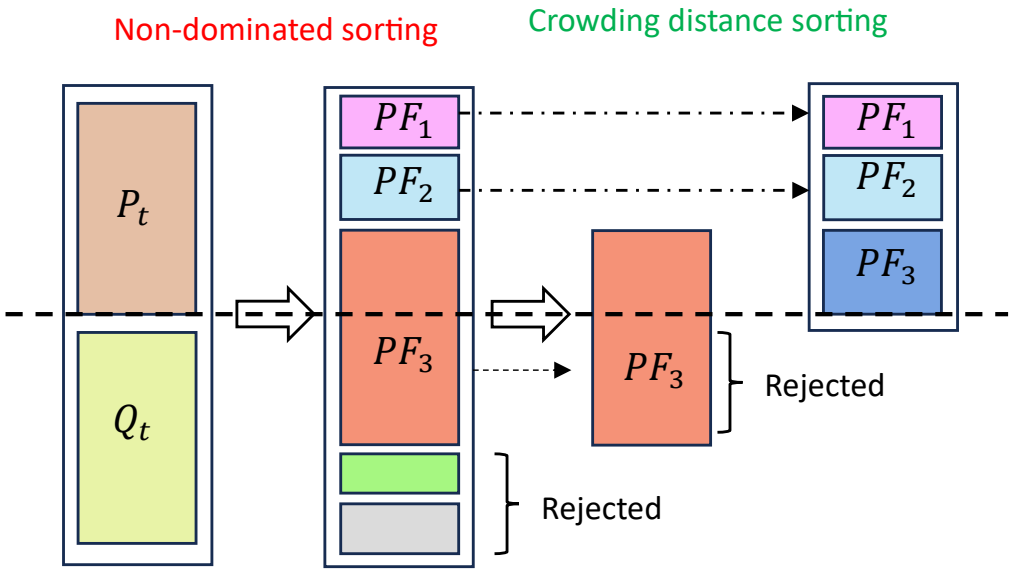


FIGURE 3.4: Operating principle of NSGA-II algorithm.

Algorithm 5 Pseudo code of the NSGA-II algorithm

Initialize population P_0 with N individuals
Evaluate fitness of each individual in P_0
for generation $t = 0$ to $T - 1$ **do**
    Generate offspring population Q_t from P_t using crossover and mutation
    Evaluate fitness of each individual in Q_t
    Combine populations: $R_t = P_t \cup Q_t$
    Perform non-dominated sorting on R_t to identify Pareto fronts F_i
    Initialize new population $P_{t+1} = \emptyset$
    Initialize front index $i = 1$
    while $|P_{t+1}| + |F_i| \leq N$ **do**
        $P_{t+1} = P_{t+1} \cup F_i$
        $i = i + 1$
    end while
    Sort front F_i by crowding distance
    Add the most widely spread individuals from F_i to P_{t+1} until $|P_{t+1}| = N$
end for
Output: the final population P_T as the set of Pareto-optimal solutions

3.5.2 Strength Pareto Evolutionary Algorithm 2 (SPEA2)

The Strength Pareto Evolutionary Algorithm 2 (SPEA2) is a multi-objective optimization algorithm developed by Zitzler et al. [81] as a revised version of its predecessor SPEA. SPEA2 seeks to effectively identify solutions that optimize multiple objectives simultaneously. It works as follows: each candidate solution is evaluated based on its "strength", or how many solutions it dominates, and its "raw fitness", or how closely it resembles other solutions. These evaluations are employed to create and maintain an external archive of non-dominated solutions. The algorithm iteratively evolves a population of candidate solutions using genetic operators, including selection, crossover, and mutation.

SPEA2 distinguishes itself from its predecessor by utilizing a refined fitness assignment strategy, which incorporates a nearest neighbor density estimation technique to enhance search efficiency. Additionally, it implements an improved archive truncation method to preserve boundary solutions. The pseudo-code for the SPEA2 algorithm is shown in Algorithm 6.

Algorithm 6 Pseudo code of the SPEA2 Algorithm

1: Initialize population P with random solutions
2: Calculate raw fitness for each solution in P
3: Assign strength to each solution in P based on the number of solutions it dominates
4: Construct the archive A by selecting nondominated solutions from $P \cup A$
5: **while** termination condition not met **do**
6: Generate offspring population Q through variation and recombination of solutions in P
7: Calculate raw fitness and strength for each solution in Q
8: Update the environmental fitness of solutions in $P \cup Q$
9: Select the best solutions from $P \cup Q$ to form the next generation
10: **end while**

3.5.3 Pareto Archived Evolution Strategy (PAES)

PAES (Pareto Archived Evolution Strategy) is a (1+1) evolution strategy originally introduced by Knowles et *al.* [83]. This method incorporates local search techniques with a population size of one, leveraging a reference archive of previously discovered solutions to enhance its performance. The archive plays a crucial role in determining the approximate dominance ranking of the current solution vector.

The PAES process can be summarized as follows: Initially, a single individual is generated to serve as the parent for producing new solutions. A mutation operator is then applied to this parent individual to create an offspring. The algorithm evaluates whether the offspring should be added to the archive and decides which solution will become the parent for the next generation. Non-dominated solutions identified during the search are stored in an external archive [18, 84]. This iterative process helps in maintaining a diverse set of high-quality solutions.

The pseudo-code for PAES is provided in Algorithm 7, illustrating the step-by-step operation of the strategy.

3.5.4 The multi-objective evolutionary algorithm based on decomposition (MOEA/D)

MOEA/D (Multi-Objective Evolutionary Algorithm based on Decomposition), proposed by Zhang et al. [82], decomposes multi-objective optimization problems (MOPs) into a number of single-objective problems (SOPs). This decomposition involves linear or nonlinear weighted aggregations of the original MOP objectives. Various aggregation methods can be utilized, including the weighted

Algorithm 7 Pseudo code of the PAES Algorithm

```
Initialize archive A with a random solution
Set iteration counter t ← 0
while termination condition not met do
    Generate a new solution s′ by perturbing a solution from A
    if s′ is dominated by any solution in A then
        Discard s′
    else if s′ dominates any solution in A then
        Replace dominated solutions in A with s′
    else
        Add s′ to A
    end if
    Increment iteration counter t ← t + 1
end while
Output: Archive A containing the approximated Pareto front
```

sum approach, the Tchebycheff approach, and the penalty boundary intersection method.

Once the SOPs are established, MOEA/D creates neighborhood relationships among them based on their aggregation weight vectors. To determine these relationships, the algorithm calculates the distances between the aggregation vectors; SOPs with smaller distances between their vectors are considered closer by MOEA/D. This structure allows MOEA/D to optimize each SOP using information from its neighboring problems, enhancing the search process.

The pseudo-code for MOEA/D is illustrated in Figure 8, which outlines the algorithm's step-by-step execution.

3.6 Performance metrics

In single-objective optimization problems, assessing the quality of a solution is relatively straightforward: for minimization problems, a smaller value of the objective function indicates a better solution. However, evaluating the quality

Algorithm 8 Pseudo code of the MOEA/D algorithm

```
Initialize the weight vectors λ_1, λ_2, ..., λ_μ uniformly in the objective space
Randomly initialize a population P of size μ
Set iteration counter t ← 0
while termination condition not met do
    for i ← 1 to μ do
        Select neighbors of solution x_i using a neighborhood selection strategy
        Decompose the objectives for each solution and its neighbors
        Update the neighbor solutions using a local search or variation operator
    end for
    Update the population P with the improved solutions
    Update the external archive if necessary
    Increment iteration counter t ← t + 1
end while
```

of a Pareto set in multi-objective optimization (MOP) is more complex. Key requirements for an effective MOP strategy include [85]:

1. **Convergence**: The obtained solution set should closely approximate the true Pareto front.

2. **Diversity**: The strategy should maintain a diverse set of solutions.

The first requirement ensures that the solutions are near-optimal, while the second guarantees a broad spectrum of trade-off solutions. These two functionalities cannot be adequately captured by a single performance indicator, prompting the development of various metrics in the literature [86].

In the following sections, we will present the most commonly used performance metrics in this context.

3.6.1 Hypervolume (HV)

One of the most prominent indicators in multi-objective optimization is the hypervolume, also referred to as the S-metric or Lebesgue measure. Initially

proposed by Zitzler and Thiele in [80], the hypervolume indicator is characterized by its robust mathematical properties, enabling it to effectively capture both dominance and distribution characteristics of Pareto fronts without requiring explicit knowledge of the true Pareto front. Empirical investigations [87, 88] have demonstrated its superior performance relative to alternative metrics, contributing to its widespread adoption within the evolutionary computation community [89].

The hypervolume indicator quantitatively assesses the volume of the objective space that is dominated by the approximation of the Pareto front, denoted as Y_N, which is constrained by a reference objective vector $r \in \mathbb{R}^m$. This relationship is defined such that for all $y \in Y_N, y \leq r$. The hypervolume is mathematically represented as:

$$HV\left(Y_N; r\right) = \lambda_m \left(\bigcup_{y \in Y_N} [y, r] \right)$$

where λ_m signifies the m-dimensional Lebesgue measure. A higher value of the hypervolume indicator correlates with enhanced convergence towards the Pareto front and increased diversity among the solutions. Figure 3.5 illustrates this concept in the context of a bi-objective scenario $(m = 2)$.

3.6.2 C-metric (C)

The C-metric, also known as the coverage of two sets, is a binary performance indicator introduced in [80]. It evaluates the performance of two non-dominated sets by computing the proportion of points in one Pareto front approximation that are weakly dominated by points in the other set.

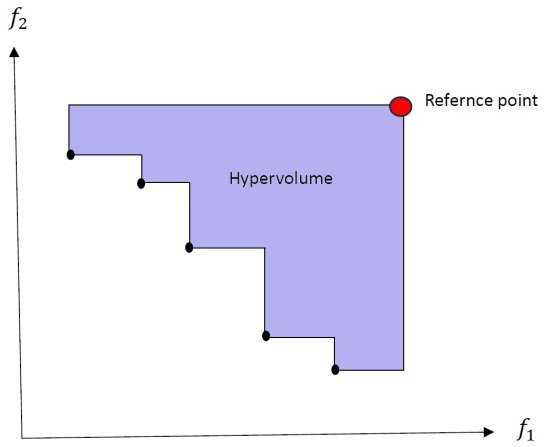


FIGURE 3.5: Illustration of the hypervolume indicator for a bi-objective problem.

Let Y_N^1 and Y_N^2 represent two approximations of the Pareto front. The C-metric quantifies the proportion of points in Y_N^2 that are weakly dominated by points in Y_N^1. This binary indicator maps the ordered pair (Y_N^1, Y_N^2) to the interval $[0, 1]$ and is defined as follows:

$$C\left(Y_N^1, Y_N^2\right) = \frac{|\{y^2 \in Y_N^2 \,|\, \exists y^1 \in Y_N^1 \text{ such that } y^1 \leq y^2\}|}{|Y_N^2|}.$$

If $C\left(Y_N^1, Y_N^2\right) = 1$, this indicates that all elements of Y_N^2 are either dominated by or equal to elements of Y_N^1. Conversely, if $C\left(Y_N^1, Y_N^2\right) = 0$, none of the elements of Y_N^2 are weakly dominated by elements of Y_N^1.

It is crucial to compute both orderings, as it is not guaranteed that $C\left(Y_N^1, Y_N^2\right)$ is equal to $1 - C\left(Y_N^2, Y_N^1\right)$. Moreover, $C(Y_N^1, Y_N^2) > C(Y_N^2, Y_N^1)$ implies that Y_N^1 contains superior solutions compared to Y_N^2.

3.6.3 Generational Distance (GD)

The Generational Distance (GD) [76] is a performance metric that quantifies the average distance between the solutions in an approximated Pareto front and

their nearest counterparts in the true Pareto front. The GD is defined mathematically as follows:

$$GD\left(Y_N; Y_P\right) = \frac{1}{|Y_N|}\left(\sum_{y^1 \in Y_N} \min_{y^2 \in Y_P} \left\|y^1 - y^2\right\|^p\right)^{\frac{1}{p}}$$

where $|Y_N|$ denotes the number of points in the approximated Pareto front Y_N, and $Y_P \subseteq \mathcal{Y}_P$ represents a discrete approximation of the true Pareto front. Typically, the parameter p is set to 2, corresponding to the Euclidean distance.

A lower GD value indicates better performance, as it signifies that the solutions within the approximated Pareto front are closer to the true Pareto front, thereby reflecting enhanced convergence quality.

3.6.4 Spacing (SP)

The spacing metric (SP) [90] quantifies the variation in distances between consecutive elements of a Pareto front approximation. This indicator is computed as follows:

$$SP\left(Y_N\right) = \sqrt{\frac{1}{|Y_N| - 1}\sum_{j=1}^{|Y_N|}\left(\bar{d} - d^1\left(y^j, Y_N \backslash \{y^j\}\right)\right)^2}$$

where Y_N denotes the Pareto front approximation. The term $d^1\left(y^j, Y_N \backslash \{y^j\}\right) = \min_{y \in Y_N \backslash \{y^j\}} \|y - y^j\|_1$ represents the l_1 distance from the point $y^j \in Y_N$ to the set $Y_N \backslash \{y^j\}$, and $\bar{d}$ is the mean of all $d^1\left(y^j, Y_N \backslash \{y^j\}\right)$ for $j = 1, 2, \ldots, |Y_N|$.

A lower value of the spacing metric indicates better performance, suggesting that the algorithm produces a more consistent and evenly distributed set of solutions.

3.7 Conclusion

This chapter focuses on multi-objective optimization (MOO), beginning with a clear definition of the field and an introduction to the concept of Pareto dominance, which is fundamental for identifying optimal solutions in multi-objective problems. Subsequent sections explore various decision-making approaches, including a priori, posteriori, and interactive methods, highlighting their respective strengths and applications.

The chapter further investigates Multi-Objective Evolutionary Algorithms (MOEAs), which have garnered significant interest due to their effectiveness in exploring the solution space and identifying diverse sets of non-dominated solutions. Prominent MOEAs, such as NSGA-II, SPEA-II, PAES, and MOEA/D, are examined in detail, discussing their mechanisms and comparative advantages.

Finally, the chapter concludes with an overview of performance metrics commonly employed to evaluate the quality of solutions generated by multi-objective optimization algorithms, providing a comprehensive framework for assessing their effectiveness.

Chapter 4

Minimum weight minimum connected dominating set (MWMCDS) Problem

4.1 Introduction

The study of dominating sets and their variants is crucial due to their wide-ranging applications and the fundamental challenges they present in graph theory. These concepts not only solve practical problems in network design, social networks, and biological systems but also drive theoretical research and algorithm development in combinatorial optimization. Minimum weight minimum connected dominating is a bi-objective optimization problem that seeks to optimize both the size and the weight of the generated connected dominating set. In this chapter, we first introduce some basic definitions in graph theory which are necessary to understand the rest of this thesis. Then, we describe the dominating sets and their variants. After that, we delve into the MWMCDS problem, providing a formal problem statement and illustrating the problem with a detailed example and we conclude by mentioning related works.

4.2 Preliminaries on graphs

For a given graph $G = (V, E)$ consists of two sets V and E. The elements of V are called *vertices* (or *nodes*), and the elements of E are called *edges* (or *lines*). Each edge has a set of one or two vertices associated with it, which are called *endpoints*. An edge is said to join its endpoints.

In the remainder of this section, we provide the basic definitions related to graphs, accompanied by illustrative examples. For basic concepts in graphs, we refer to [91, 92, 93, 94].

Definition 4.1: If vertex v is endpoint of edge e, then v is said to be *incident* on e, and e is incident on v.

Definition 4.2: A vertex u is *adjacent* to vertex v if they are joined by an edge.

Definition 4.3: Two distinct adjacent vertices can be referred to as *neighbors*, denoted $N(V)$, such that:

$$N(v) = \{u \mid (v, u) \in E \vee (u, v) \in E\}$$

Definition 4.4: *Adjacent edges* are edges that have an endpoint in common.

Definition 4.5: A *multi-edge* is a collection of two or more edges having identical endpoints.

Definition 4.6: A *self-loop* is an edge that connects a vertex to itself.

Example

Figure 4.1 shows an example of a graph $G = (V, E)$, G has vertices set $V = \{1, 2, 3, 4\}$ and edges set $E = \{a, b, c, d, e, f\}$. The edge f has two endpoints, that is 2 and 3. 2 and 3 are called $neighbors$, because they are joined by the edge f. a and e are adjacent edges because they have an endpoint in common (vertex 2). The set $\{a, b\}$ is a multi-edge with endpoints 1 and 2. Edge c is a self-loop.

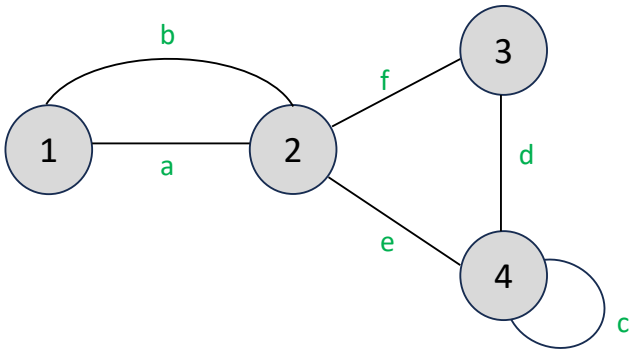


FIGURE 4.1: A graph.

4.2.1 Simple graphs

The majority of theoretical graph theory focuses on simple graphs. This is partially due to the fact that many problems regarding general graphs can be reduced into problems about simple graphs.

Definition 4.7: A *simple graph* is a graph without self-loops or multi-edges.

Definition 4.8: The *degree* of a vertex v in a simple graph G, denoted $deg(v)$, is the number of neighbors of v in G, $deg(v) = |N(v)|$.

Definition 4.9: An *isolated vertex* in a graph is a vertex of degree zero.

Definition 4.10: A *walk* in a simple graph G is a sequence of vertices: $W = v_0, v_1, \ldots, v_n$ such that for $j = 1, \ldots, n$, the vertices v_{j-1} and vj are adjacent.

Definition 4.11: A graph G is connected if there is a walk between every pair of vertices.

Example

Figure 4.2 illustrates a simple graph $G = (V, E)$ with vertices set $V = \{1, 2, 3, 4\}$ and edges set $E = \{a, b, c, d\}$. Degrees of vertices are: $deg(1) = 1$, $deg(2) = 3$, $deg(3) = 2$, $deg(4) = 2$. $W = 1, 2, 4, 3$ is a walk. G is connected because, between every pair of vertices, there is a walk.

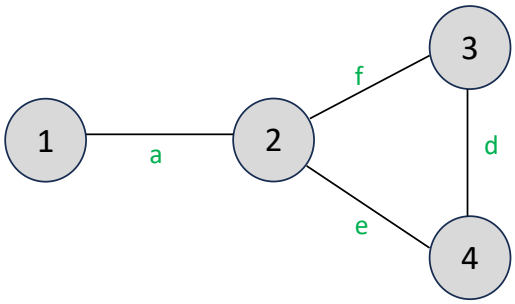


FIGURE 4.2: A simple graph.

4.2.2 Directed / undirected graphs

Depending on the nature of the graph edges, we can categorize graphs into two types: directed and undirected.

Definition 4.12: If the pairs $(u, v) \in E$ are ordered pairs, then G is called a *directed graph* (also known as *digraph*).

Definition 4.13: If the pairs $(u, v) \in E$ and $(v, u) \in E$ denote the same edge, meaning each edge between two vertices creates a connection in two opposite senses at one, then G is called *undirected graph*.

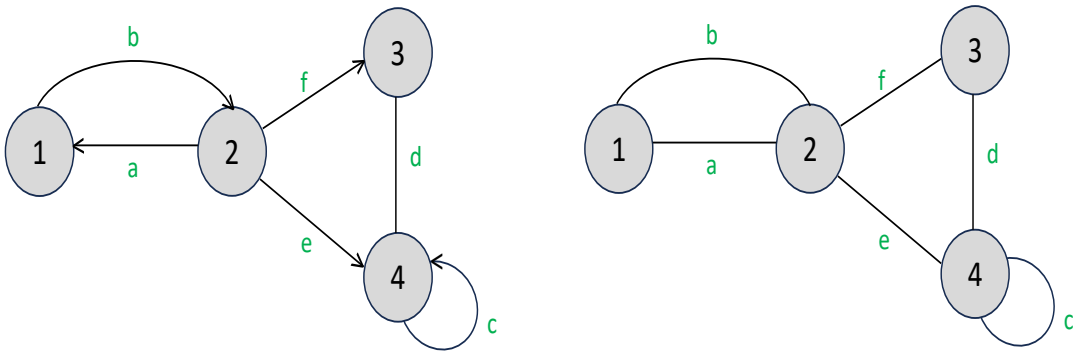
Example

In Figure 4.3 an example of a directed graph is shown on the left, and an undirected graph on the right.

4.2.3 Subgraphs

Definition 4.14: A *subgraph* of a graph G is a graph H such that $V_H \subset V_G$ and $E_H \subset E_G$.

Definition 4.15: In a graph G, the *induced subgraph* on a set of vertices $W = w_1, \ldots, w_n$, denoted $G(W)$, has W as its vertex set, and it contains every edge of



(a) Directed graph (b) Undirected graph

FIGURE 4.3: A directed \undirected graph.

G whose endpoints are in W. That is,

$V(G(W)) = W$ and $E(G(W)) = \{e \in E(G) \mid$ the endpoints of edge e are in $W\}$

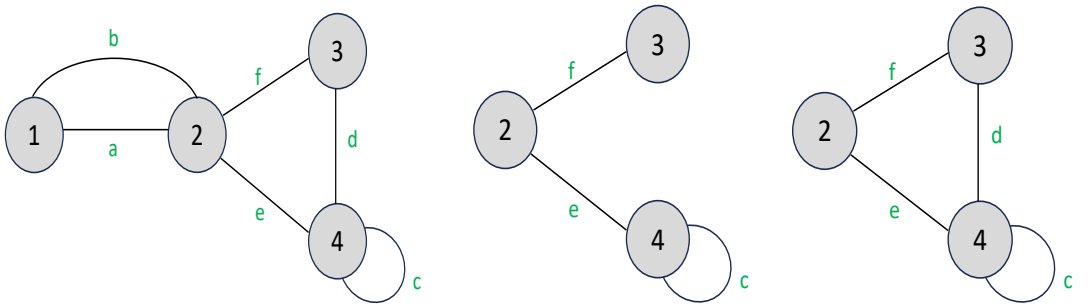
Example

Figure 4.4 illustrates an undirected graph $G = (V, E)$ with the vertex set $V = \{1, 2, 3, 4\}$ and the edge set $E = \{a, b, c, d, e, f\}$. A subgraph of G, denoted as H, has the vertex set $V_H = \{2, 3, 4\} \subseteq V$ and the edge set $E_H = \{c, e, f\} \subseteq E$. An induced subgraph of G, denoted as J, has the vertex set $V_J = \{2, 3, 4\} \subseteq V$ and the edge set $E_J = \{c, e, f, d\} \subseteq E$, which includes every edge in G whose endpoints are in V_J.

4.3 Dominating set and its variants

Given a simple undirected graph $G = (V, E)$, where V is the set of vertices and E is the set of edges.

Definition 4.16: A *dominating set* (DS) is a subset D of V such that each vertex



(a) Directed graph G (b) Subgraph of G (c) Induce subgraph of G

FIGURE 4.4: An undirected graph G, a subgraph of G , and an induced subgraph of G.

not in D has at least one neighbour in D. Vertices in the DS are called *dominators*, and those out of it are called *dominatees*.

Definition 4.17: A *connected dominating set* (CDS) is a connected induced subgraph by D, where D represents a DS.

Definition 4.18: A *minimum connected dominating set* (MCDS) is a CDS of minimum cardinality (size).

Definition 4.19: If G is a vertex-weighted graph, a *minimum weight connected dominating set* (MWCDS) is a CDS with minimum total weight.

Example

Figure 4.5 gives an illustrative example of a dominating set and its variants on a simple undirected graph with 7 vertices and 9 edges. Dominators are highlighted with a black background and white lettering.

4.3.1 Complexity

From the view of complexity theory, the DS problem is a classic NP-hard problem, and cannot be approximated with a constant ratio under the assumption $P \neq NP$ [95]. Its considered variants are generally NP-hard [96, 97].

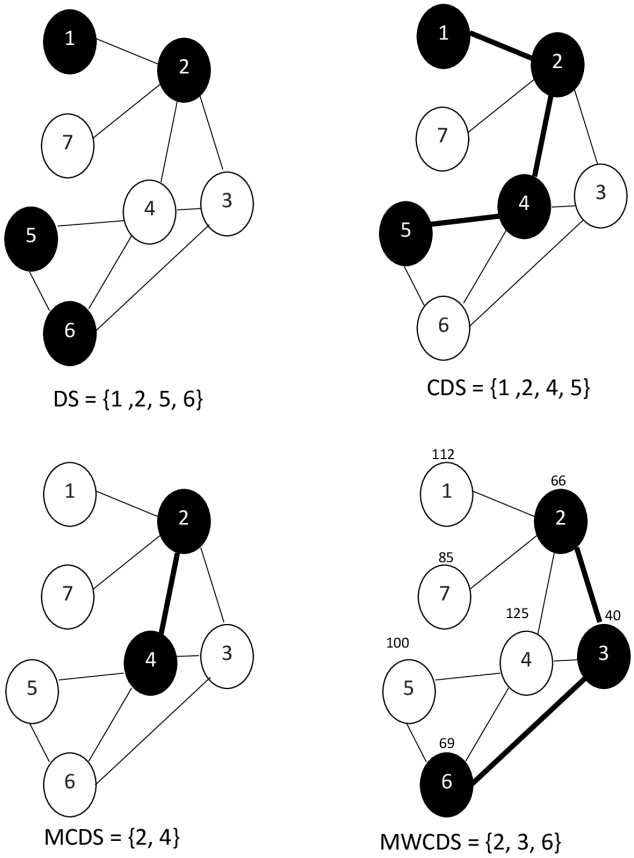


FIGURE 4.5: Example of DS (Dominating Set), CDS (Connected DS), MCDS (Minimum CDS) and MWCDS (Minimum weighted CDS) in a simple undirected graph.

4.4 Minimum weight minimum connected dominating set (MWMCDS) Problem

MWMCDS is a new bi-objective combinatorial optimization problem variant of CDS defined firstly by Rengaswamy et *al.* in 2017 [13]. It seeks to minimize simultaneously the cardinality of the connected dominating set and its total edge weight. The authors in [13] proposed a multiobjective genetic algorithm labeled MOGA based on the scalarization model and built upon a fitness model to deal

with this problem, they used a data transfer model to analyze performance difference between an MCDS and an MWMCDS, and their experiments show that using an MWMCDS instead of MCDS gives better results.

4.4.1 Problem statement

Given an undirected edge-weighted graph $G = (V, E, w)$, where V and E represent the set of vertices and the set of edges respectively, and $w : E \mapsto R^+$ is a function that assigns a positive weight value to all edges in E. In the rest of this thesis, $|v|$ is denoted by n, and $|E|$ by m. The goal in the MWMCDS problem is to find a connected dominating set $D \subseteq V$ in which the cardinality and the total weight are to be minimized together. Hence, we can formulate this problem as follows:

minimize $\{F_1(D), F_2(D)\}$

subject to $\forall\, v \in V \backslash D : N(v) \cap D \neq \emptyset$,
$D \subseteq V$,
$G(D)$ is connected.

In the above definition, we look for a connected dominating set $D \subseteq V$ (a candidate solution) in which two objective functions are simultaneously minimized. Let $|D|$ represent the cardinally of D. The first objective function $F_1(D) := |D|$, named as the cardinality objective function, intends to minimize the size of the candidate solution while the second objective function $F_2(D)$ named as the weight objective function, intends to minimize its total weight. $F_2(D)$ is calculated as follows:

$$F_2(D) := F_{2a} + F_{2b} \tag{4.1}$$

$$F_{2a} = \sum_{((u,v)\in E)\wedge(u\in D\wedge v\in D)} w_{(u,v)} \tag{4.2}$$

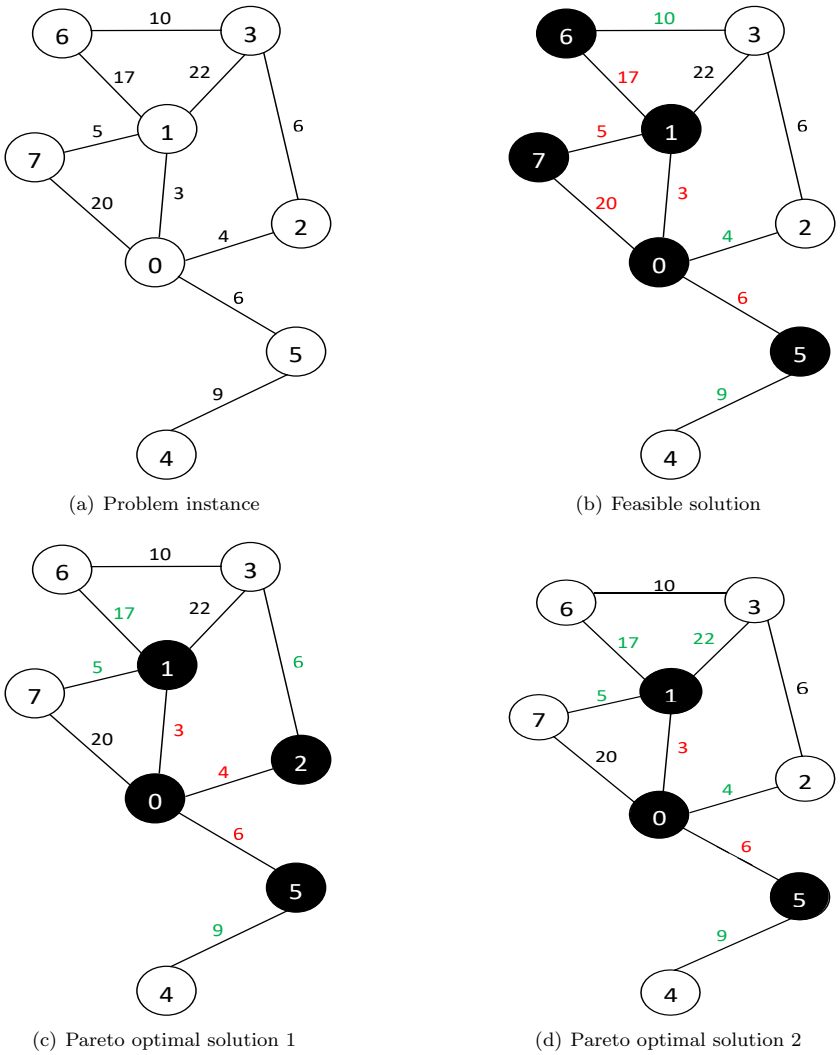
$$F_{2b} = \sum_{(u\in V\setminus D)} \mathbf{min}\{w_{(u,v)} \mid (u,v) \in E \wedge v \in D\} \tag{4.3}$$

As indicated in the previous formulas, F_{2a} counts the sum of all weights of edges that connect two vertices in D, and F_{2b} takes for each vertex v in $V \setminus D$ the minimum weight from all weights of edges that connect v with a vertex in D and then aggregates them. $F_2(D)$ is the sum of F_{2a} and F_{2b}.

4.4.2 Graphical example

Figure 4.6 gives a graphical example of the MWMCDS problem. In particular, Figure 4.6.a shows a simple undirected edge-weighted graph that contains 8 vertices and 10 edges. The labels printed within the vertices represent their ID, and those printed next to the edges are their weights. The solutions shown in Figure 4.6.b represents a feasible solution, and Figures 4.6.c and 4.6.d correspond to Pareto optimal solutions. The black vertices are the vertices that compose the candidate solution D, the red weights are the weights that F_{2a} takes into account, and the green weights are those that F_{2b} considers.

The feasible solution shown in Figure 4.6.b, $D = \{0, 1, 5, 6, 7\}$, has the objective function values $F_1(D) = 5$ and $F_2(D) = 74$ ($F_{2a} = 51$, $F_{2b} = 23$). The Pareto optimal solution 1, shown in Figure 4.6.c, with $D = \{0, 1, 2, 5\}$, has the objective function values $F_1(D) = 4$ and $F_2(D) = 50$ ($F_{2a} = 13$, $F_{2b} = 37$). The Pareto optimal solution 2, shown in Figure 4.6.d, with $D = \{0, 1, 5\}$, has the objective function values $F_1(D) = 3$ and $F_2(D) = 66$ ($F_{2a} = 9$, $F_{2b} = 57$).



(a) Problem instance

(b) Feasible solution

(c) Pareto optimal solution 1

(d) Pareto optimal solution 2

FIGURE 4.6: An illustrative example of the MWMCDS problem.

4.4.3 Related work

Due to the difficulty and the potential benefits of solving DS and its variants, considerable work has been conducted in this regard. Most of them are based on metaheuristic algorithms, which are approximate approaches that can find reasonably near-optimal solutions in an acceptable computation time, rather than exact algorithms that guarantee the optimality of the returned solutions but in an exponential time.

As an example of such approaches, Morgan and Grout [98] introduced the first metaheuristic to deal with the MCDS problem. The latter combines tabu search and simulated annealing algorithm. Jovanovic et *al.* [99] proposed an ant colony optimization algorithm (ACO) with greedy heuristics. Recently, Li et *al.* [100] presented a greedy randomized adaptive search procedure (GRASP) that incorporates a tabu search as a local enhancement process. Besides, Wu et *al.* [101] developed a tabu search procedure (RSN-TS) based on a restricted swap-based neighborhood. The authors conducted a considerable number of experimental tests to show that RSN-TS outperforms GRASP and ACO both in terms of solution quality and computation time. Later, Hedar et *al.* [102] implemented two methods for solving the MCDS problem. The first one is a memetic algorithm and the second one is a simulated annealing. The performance of both approaches when applied to the MCDS problem on common benchmark instances is better than ACO and GRASP but less than RSN-TS based on results reported in the literature.

However, few researches have been carried out on the MWCDS problem. Ambühl et *al* [103] developed an approximation algorithm to solve this problem. A hybrid genetic algorithm (HGA) and a population-based iterated greedy (PBIG) algorithm were proposed in [104]. Bouamama et *al.* [105] developed a hybrid ant colony optimization approach combined with a reduced variable neighborhood search (ACO-RVNS) to solve both MCDS and MWCDS. In this algorithm, MCDS is treated as MWCDS with a unit weight of one to every vertex of the input graphs. It was demonstrated that ACO-RVNS outperforms RSN-TS, PBIG, and HGA across all available benchmark sets, especially for large problem instances. In [106] a restart local search algorithm with the tabu method (RLS-Tabu) is proposed to solve the MWCDS problem. The authors

present two strategies in the neighborhood search procedure for appropriately eliminating vertices: a greedy and random strategy, and multiple deletion strategy. More recently, authors in [107] developed a local search algorithm called CVPLS to tackle the MWCSD problem.

One should mention that all previous approaches have something in common: they optimize only a single objective function such as minimizing the size of the connected dominating set (CDS) and minimizing its total weight for MCDS and MWCDS, respectively. To the best of our knowledge, the first approach in the literature that considered these two objectives together was presented in [13]. The authors of this study first defined the minimum weight minimum connected dominating set problem (MWMCDS) of which the aim is to minimize simultaneously the size and the total weight of the generated CDS. Then, they proposed a multiobjective genetic algorithm (MOGA) based on a scalarization model to deal with the MWMCDS problem. They compare MOGA with an MCDS produced in [14]. The analysis by comparison with the MCDS algorithm indicates significant performance by MOGA.

MWMCDS problem was then exclusively addressed in our works, which are listed below and thoroughly detailed in the next chapters.

- A greedy simulated annealing algorithm (GSA) [12] based on the scalarization concept. The experimental results indicate that GSA performs better than MOGA and MCDS.

- An improved Pareto genetic algorithm based on NSGA-II named I-NSGA-II [15]. A comparison of this approach against NSGA-II demonstrates its superiority.

- A multiobjective greedy simulated annealing algorithm (MGSA) [16] based on Pareto optimality concept. Comparative analysis against MCDS, MOGA, GSA, and I-NSGA-II demonstrates the effectiveness of MGSA.

4.5 Conclusion

Throughout this chapter, we have explored the Minimum Weight Minimum Connected Dominating Set (MWMCDS) problem. We started by introducing preliminaries and basic definitions of graph theory which are essential for comprehending the rest of this thesis. These definitions include simple graphs, directed/undirected graphs, and subgraphs. Illustrating examples are also given. We then delved into the dominating set (DS) and its variants. After that, we presented the MWMCDS problem which is an extension of DS. We outline its problem statement, provide a detailed example, and survey related work.

Chapter 5

Greedy Simulated Annealing (GSA) Algorithm for MWMCDSP

5.1 Introduction

In this chapter, we introduce a novel technique termed the Greedy Simulated Annealing (GSA) algorithm to address the MWMCDSP. The GSA algorithm improves the standard Simulated Annealing (SA) approach by introducing a greedy heuristic for generating initial solutions and neighbor candidates, hence enhancing the overall solution quality and computational efficiency. Additionally, GSA employs an effective temperature adjustment mechanism to guide the search. This chapter details the design of the GSA algorithm, its components, and its effectiveness compared to existing approaches.

5.2 Greedy heuristic

All the greedy heuristics used in this thesis follow the same process, the difference lies in the selection method.

5.2.1 General process

A feasible solution is greedily constructed as follows. For a given partial feasible solution S, all vertices in V can be partitioned into three disjoint subsets with respect to S: vertices in S (named as BLACK vertices), vertices that are not in S but are dominated by BLACK vertices (called GRAY vertices), and vertices that are not in S and not dominated by BLACK vertices (referred to as WHITE vertices). The initial vertex is selected from WHITE vertices, and subsequent vertices are chosen from GRAY vertices in order to establish the connectivity of DS. The heuristics terminate when there is no WHITE vertex left and all BLACK vertices are connected to each other.

5.2.2 Selection method

Let $d_s(v)$ represents the number of WHITE neighbors of vertex v (current degree with respect to S).
The first vertex v^{first} is chosen as follows:

$$v^{first} \leftarrow \textbf{argmax}\{d_S(v) \mid v \in V\}. \tag{5.1}$$

The remaining vertices to be placed in S are chosen as follow:

$$v^* \leftarrow \textbf{argmax}\{d_S(v) \mid (v \in V \setminus S) \ \wedge \ (\text{color}(v) = \textbf{GRAY})\}. \tag{5.2}$$

5.3 Representation

A problem instance is mapped to an edge-weighted graph $G(V, E, w)$ where each vertex v in V is represented by a unique integer number from $\{0, 1, \cdots, n-$

$1\}$, where n denotes the size of V, that is, $n = |V|$. A candidate solution S is coded by a vector of fixed length n of which each element can take values 0 or 1 depending on if the corresponding vertex belongs to S or not.

5.4 Aggregated objective function

Given a candidate solution S, with respect to the objective functions F_1 and F_2 defined in Section 4.4.1, the sum-weighted aggregated objective function can be defined as:

$$F := \alpha \times F_1' + \beta \times F_2' \tag{5.3}$$

Where $\alpha \in [0, 1]$ and $\beta \in [0, 1]$ are parameters such that $\alpha + \beta = 1$ and $\alpha = \beta$. In addition, F_1' and F_2' are the normalized values of F_1 and F_2, based on the total number of vertices and total weight respectively.

5.5 GSA framework

Simulated Annealing (SA) [27] is a well-known metaheuristic approach that has been applied successfully to a large number of combinatorial optimization problems. SA is both a single-solution-based algorithm and exploration-oriented (see Section 2.6.3.2).

Our algorithm named GSA is a standard SA algorithm improved by starting with a good initial solution based on a greedy heuristic and generating neighbors either greedily or randomly. Moreover, GSA uses a useful calculation method for changing temperature. These improvements ensure computational efficiency and improve the quality of obtained solutions. A high-level description of the GSA algorithm is illustrated in Alg. 9.

Algorithm 9 GSA for the MWMCDS problem

input: $A\ problem\ instance\ (G, V, E, w),\ and\ parameters$ *sol_size* , $k\ and\ T_0$
$\varphi \leftarrow$ generate_initial_solution(sol_size) //see Alg. 10
$Temperature \leftarrow T_0$
$S^{best} \leftarrow argmin\{F(S)|\ S \in \varphi\}$ //see (5.3) for the definition of F.
$S \leftarrow S^{best}$
while termination condition not satisfied **do**
    $p \leftarrow random\ number\ uniformly\ distributed\ over\ [0, 1]$
    if $p > 0.5$ **then**
        $S' \leftarrow$ neighbor_greedy(S) //see Alg. 13
    else
        $S \leftarrow pick\ a\ random\ solution\ from\ \varphi$
        $S' \leftarrow$ neighbor_random(S)
    end if
    if $F(S') < F(S^{best})$ **then**
        $S^{best} \leftarrow S'$
    end if
    if accept(Temperature,S,S') **then**
        $S \leftarrow S'$
    end if
    $Temperature \leftarrow$ change_temperature(Temperature,k)
end while
output: $\{S^{best}, F_1(S^{best}), F_2(S^{best})\}$

5.5.1 Initial solutions

The initial solutions set is created as given in Algorithm 10, where $sol\_size = 100$. The first solution is generated using the greedy heuristic, which involves selecting vertices based on their degree, and subsequent solutions are generated at random. This process ensures a diverse initial set of solutions and offers a balanced starting point for further optimization.

5.5.2 Neighbors generation

The neighbor of a candidate solution is obtained either greedily using procedure neighbor_greedy() (see Alg. 13) or randomly using procedure neighbor_random() with respect to a probability distribution p. In this context, both

Algorithm 10 generate_initial_solution (sol_size)

input: sol_size
$\varphi \leftarrow \emptyset$
$S \leftarrow$ generate_greedy() //see Alg. 11
$\varphi \leftarrow \varphi \cup \{S\}$
for $i \leftarrow 2$ *to* sol_size **do**
    $S \leftarrow$ generate_random() //see Alg. 12
    $\varphi \leftarrow \varphi \cup \{S\}$
end for
output: $\varphi = \{S_1, S_2, \ldots, S_{\text{sol\_size}}\}$

Algorithm 11 generate_greedy()

$S \leftarrow \emptyset$
$v^{first} \leftarrow$ **argmax**$\{d_s(v) \mid v \in V\}$
$v^{first}.color \leftarrow GRAY$
for all $v \in V \setminus \{v^{first}\}$ **do**
    $v.color \leftarrow WHITE$
end for
repeat
    $v^* \leftarrow$ **argmax**$\{d_s(v) \mid (v \in V \setminus S) \ \wedge \ (color(v) = GRAY)\}$
    $v^*.color \leftarrow BLACK$
    $S \leftarrow S \cup \{v^*\}$
    for all $v \in N(v^*) \setminus S$ **do**
        $v.color \leftarrow GRAY$
    end for
until (All vertices are colored either BLACK or GRAY)
output: S

procedures follow the same basic steps except that the last chooses the input solution and the dominator vertices randomly.

5.5.3 Acceptance criterion

The candidate solution S' is accepted as the current solution if it outperforms the incumbent solution S, otherwise, it may be accepted or rejected depending on the Metropolis condition (for further details on the Metropolis condition, see 2.6.3.2.1).

Algorithm 12 generate_random()

$S \leftarrow \emptyset$
$v^{first} \leftarrow pick\ a\ random\ vertex\ from\ V$
$v^{first}.color \leftarrow GRAY$
for all $v \in V \setminus \{v^{first}\}$ **do**
    $v.color \leftarrow WHITE$
end for
repeat
    $v^* \leftarrow pick\ a\ random\ vertex\ from\ GRAY\ vertices$
    $v^*.color \leftarrow BLACK$
    $S \leftarrow S \cup \{v^*\}$
    for all $v \in N(v^*) \setminus S$ **do**
        $v.color \leftarrow GRAY$
    end for
until (All vertices are colored either BLACK or GRAY)
output: S

Algorithm 13 Procedure neighbor_greedy(S)

input: $an\ incumbent\ solution\ S$
$S' \leftarrow S$
$v \leftarrow$ **argmin**$\{|N(v)|, v \in S'\}$
$v.color \leftarrow GRAY$
$S' \leftarrow S' \setminus \{v\}$
for all $vertex\ v^p \in N(v)$ **do**
    if $(v^p.color == GRAY) \wedge (N(v^p) \cap (S' \setminus \{v\}) = \emptyset)$ **then**
        $v^p.color \leftarrow WHITE$
    end if
end for
while There exists WHITE vertex **do**
    $v^* \leftarrow$ **argmax**$\{\mathrm{d}_{S'}(v) \mid (v \in V \setminus S') \wedge (color(v) = GRAY)\}$
    $v^*.color \leftarrow BLACK$
    $S' \leftarrow S' \cup \{v^*\}$
    for all $v \in N(v^*) \setminus S'$ **do**
        $v.color \leftarrow GRAY$
    end for
end while
output: S'

5.5.4 Change tempearture

The temperature plays a crucial role in controlling the evolution of the solution S. Initially, the algorithm starts with the initial temperature T_0 set to a high value ($T_0 = 1000$), then, after k consecutive iterations (here k is set to 3) at the same temperature, the temperature will be decreased by a factor of 1-γ (γ = 0.9). If the temperature takes a value lower than 1, we use a temperature reheating and it returns to the initial value T_0.

5.6 Complexity

Here, we describe the time complexity of GSA algorithm.
The *generate_initial_solution* procedure, which includes the application of a greedy heuristic and the generation of random solutions, is performed in $O(sol\_size \times (n + m))$ time. The *generate_greedy* and *generate_random* procedures used in neighbors' generations require $O(n+m)$ time for each neighbor. The acceptance criterion and temperature change steps each require $O(1)$ time. Consequently, the overall time complexity of the GSA algorithm for *max_iter* iterations can be expressed as:

$$O(sol\_size \times (n + m)) + (\textit{max\_iter} \times (n + m))$$

5.7 Experimental evaluation

The proposed algorithm GSA was implemented using C++ language. The experimental results were obtained on a PC with an Intel Core i5-1135G7 2.40GHz processor and 8.0 GB of memory.

Its performance was compared against two recent algorithms from the literature, namely MOGA [13] and MCDS [14]. The results of the two approaches are reproduced from [13].

GSA was evaluated on the same benchmark set introduced in [13] where each instance consists of a simple undirected edge-weighted graph modeling a data transfer system where every vertex transfers data at instant t with probability P_t and this data can be dropped with probability P_d. The distance traveled by the data in the transfer is represented indirectly by the weight, which is expressed by the energy consumed during the travel. Thus, the energy consumed in case of successful transfer is equal to the distance traveled by the data, and in the case of failed transfer is equal to half of distance traveled. The energy consumed in the transfer of all data in the network is represented by *energy consumption*.

5.8 Results

The performance of GSA against MOGA and MCDS with respect to energy consumption for 100 instances of data transfer is shown in Figure 5.1.
It can be seen that GSA consumes the least amount of energy in all cases compared with MCDS. Compared to MOGA, the energy requirements of GSA are lesser in all cases except the scenario ($n = 100$) where both algorithms give similar results.

Figure 5.2 represents the number of dominator vertices produced by GSA, MOGA, and MCDS for different networks. Here too it can be seen that GSA performs better than MCDS in all cases by producing CDS of minimal size. Compared to MOGA, GSA gives the same results in 4 cases (n=30, n=40, n=60, and n=70), in all other cases GSA performs the best.

From previous results, it can be deduced that the proposed algorithm GSA generates more useful solutions for the MWMCDS problem than MCDS and MOGA.

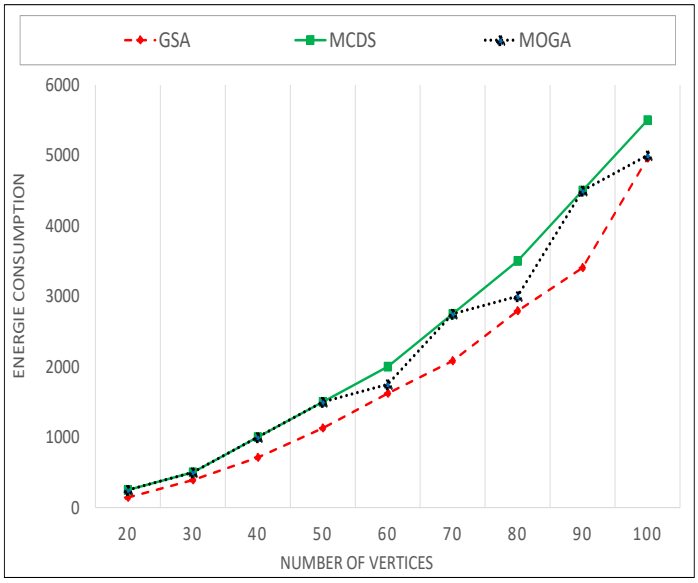


FIGURE 5.1: Energy consumption in GSA, MOGA, and MCDS

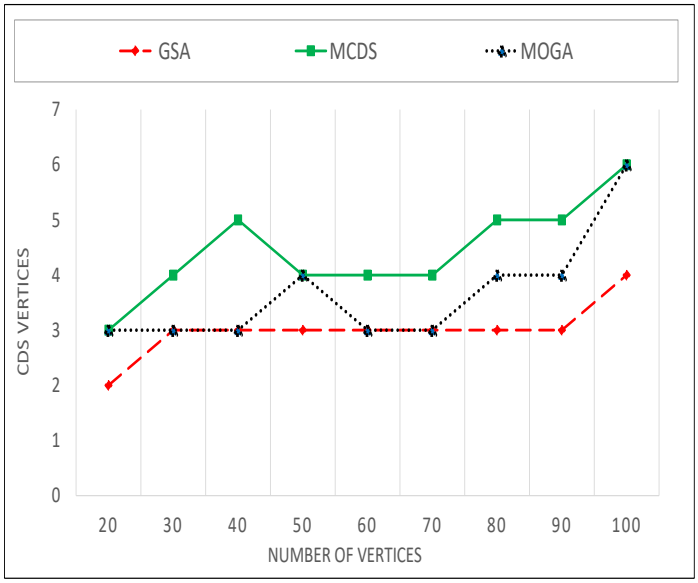


FIGURE 5.2: Size of CDS in GSA, MOGA, and MCDS

5.9 Conclusion

For the purpose of solving the minimum weight minimum connected dominant set problem (MWMCDSP) in wireless sensor networks, we have introduced a novel method in this chapter termed Greedy Simulated Annealing (GSA). GSA integrates a simulated annealing algorithm, which is seeded by a greedy constructive heuristic and employed in two hybridization models. The performance of the proposed algorithm was assessed and compared against existing algorithms such as MOGA and MCDS in terms of energy consumption and the size of the connected dominating set. Results obtained demonstrate the superiority of GSA.

Chapter 6

Improved Non-dominated Sorting Gentic Algorithm (I-NSGA-II) for MWMCDSP

6.1 Introduction

In this chapter, we introduce the Improved Non-dominated Sorting Genetic Algorithm II (I-NSGA-II) designed to solve the MWMCDSP. I-NSGA-II enhances the well-known algorithm NSGA-II in the generation of the initial population phase as well as in producing neighbors.

The chapter is organized as follows: we first discuss the greedy heuristics, and then we present the detailed design of the I-NSGA-II algorithm. Following this, we analyze the computational complexity of the proposed approach. Finally, we demonstrate the effectiveness of I-NSGA-II through extensive experiments and provide a comparative analysis with the standard NSGA-II.

6.2 Greedy heuristics

We have applied three greedy heuristics named GR1, GR2, and GR3 to determine MWMCDS, the first heuristic (GR1) is similar to the heuristic used by Dahmri and Bouamama in [12], the second heuristic (GR2) and the third heuristic (GR3) are new heuristics proposed for the MWMCDS problem. The overall process of these algorithms follows the general procedure outlined in Section 5.2.1, the selection methods are described below.

Let V^{cand} denote the set V if all vertices are WHITE, or the set of GRAY vertices if there is at least one GRAY vertex in V, $d_s(v)$ represents the number of WHITE neighbors of vertex v (current degree with respect to S), $t_w(v)$ denotes the total weight value of edges that connect vertex v with their neighbors $N(v)$, n represents the number of vertices in the graph G, and W_e denotes the total weight of all edges in G.

The first heuristic (GR1) selects the vertex having the greatest values of d_s.

$$v \leftarrow \textbf{argmax}\{d_s(v)\} \mid v \in V^{cand}\} \tag{6.1}$$

The second heuristic (GR2) selects the vertex with the greatest ratio between their d_s and t_w.

$$v \leftarrow \textbf{argmax}\{\frac{d_s(v)}{t_w(v)} \mid v \in V^{cand}\} \tag{6.2}$$

The third heuristic (GR3) chooses the vertex having the minimum value of the difference between the normalized value of its t_w and d_s.

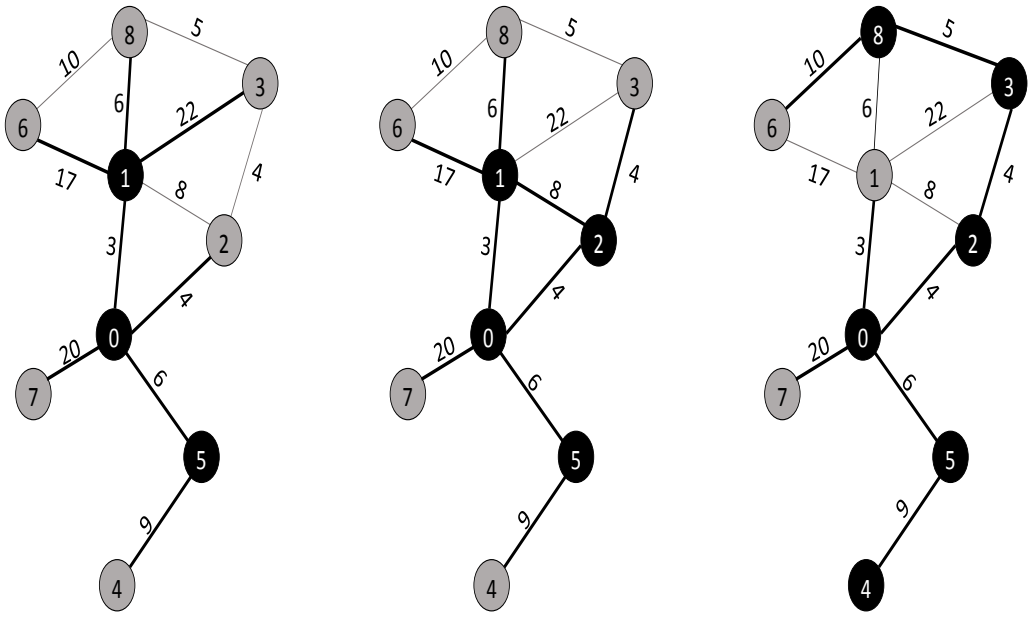
$$v \leftarrow \textbf{argmin}\{\frac{t_w(v)}{W_e} - \frac{d_s(v)}{n} \mid v \in V^{cand}\} \tag{6.3}$$

Fig. 6.1 gives an illustrative example of an MWMCDS problem instance and the solutions obtained using the three greedy heuristics. The latter is a simple undirected edge-weighted graph which contains 9 vertices and 12 edges. The constructed solution shown in Fig. 6.1.a, Fig. 6.1.b and Fig. 6.1.c corresponds to the one obtained using the first greedy heuristic, the second greedy heuristic and the third greedy heuristic respectively.

In Fig. 6.1.a, the first vertex chosen is 1 because it has the greatest degree of WHITE neighbors ($d_s(1) = 5$), then the vertex 0 then 5. Thus, $F_c = 3$ and $F_w = 87$. In Fig. 6.1.b the vertices 2, 0, 5, and 1 are selected in this order by applying Eq. 6.2. The values of objective functions of the obtained MWMCDS solution are $F_c = 4$ and $F_w = 77$. The MWMCDS solution found by the third heuristic (Eq. 6.3) is shown in Fig. 6.1.c, where the vertices 2, 0, 5, 4, 3, and 8 are selected in this order. This solution has a cardinality of $F_c = 6$ and a total weight of $F_w = 61$.

6.3 Design of I-NSGA-II algorithm

We propose to solve the MWMCDS problem with an improved Pareto genetic algorithm based on NSGA-II (see 3.5.1) refereed as I-NSGA-II. The original NSGA-II is enhanced by using greedy heuristics to generate the initial population of solutions. In addition, neighbors are obtained by applying either a genetic algorithm or a local search method. The genetic algorithm generates neighbors randomly which extends the solution space, and the local search method concentrates on obtaining better neighbors based on greedy heuristic which enhances the quality of front solutions. The nondominated sorting phase and crowding distance functions are employed to obtain a better spread of non-dominated solutions.



(a) GR1 (b) GR2 (c) GR3

FIGURE 6.1: An illustrative example of an MWMCDS problem instance and the solutions obtained using GR1, GR2, and GR3.

Algorithm 14 shows the main structure of I-NSGA-II. In line 2, the parent population named P_t, which contains sol_size individuals (each individual represents a feasible solution S) is constructed greedily or randomly. Lines 5-14 produce the offspring population called as Q_t by generating a neighbor for each solution in P_t. Neighbors are obtained by either applying local_search(S_i) method or by using genetic_algorithm(S_i) (crossover and mutations operations). The combined population R_t is constructed in line 15, then it is sorted into fronts on line 16 by applying the function fast_nondominated_sorting (R_t). A selection phase is needed to select the next generation P_{t+1} from the set of fronts F using the function selection(F) (line 17). These steps are repeated until a maximum number of iterations max_iter is reached.

Algorithm 14 I-NSGA-II for the MWMCDS problem

input: A problem instance (G, V, E, w), and parameter *sol_size* $\in \mathbf{Z}^+$
$P_t \leftarrow$ generate_initial_solution(sol_size)
repeat
    $Q_t \leftarrow \emptyset$
    for $i \leftarrow\ 0\ to\ sol\_size$ **do**
        $S_i \leftarrow$ the i^{th} solution from P_t
        $p \leftarrow$ random number uniformly distributed over[0,1]
        if $p > 0.5$ **then**
            $S'_i \leftarrow$ local_search(S_i)
        else
            $S'_i \leftarrow$ genetic_algorithm(S_i)
        end if
        $Q_t \leftarrow Q_t \cup \{S'_i\}$
    end for
    $R_t \leftarrow P_t \cup Q_t$
    $F \leftarrow$ fast_nondominated_sorting(R_t) //see algorithm 15
    $\mathrm{P_{t+1}} \leftarrow$ selection(F) //see algorithm 16
    $P_t \leftarrow \mathrm{P_{t+1}}$
until termination condition is satisfied
output: Approximate Pareto front P_t

6.3.1 Initial population

In order to obtain the initial solutions set P_t, we generate one of them using a greedy heuristic, and the remaining are generated randomly.

6.3.2 Neighbors production

In our algorithm, the set of neighbors Q_t is generated using the local search method or genetic algorithm with equal probability.

6.3.2.1 Local search method

The local search method is used to generate solutions neighbors in the search space. For a solution S we generate the neighbor S' by applying the following changes. Initially, we search for the worst vertex in S (the vertex that gives the

worst results using the greedy heuristic). Then, we remove it from the solution and we check if the produced solution stays connected. In that case, we stop, otherwise, we add to the solution a vertex greedily and we repeat this step until obtaining a connected set.

6.3.2.2 Genetic algorithm

The solution S' is produced from S using self-crossover and mutation operators. In the self-crossover operator, we exchange the segments of S at a randomly chosen crossover point and store the resulting new solution in S'. In bit flip mutation, an individual's gene is changed with a low random probability.

6.3.3 New population selection

The initial population P_t and the offspring population (Q_t) are combined to form the population R_t. From R_t only half of the individuals will be selected for the next generation $(\mathrm{P_{t+1}})$. Nondominated sorting is applied to sort and partition the population into fronts ($F_1, F2$,..., etc) according to the dominating status of each individual. The progress of the fast nondominated sorting strategy is explained in Algorithm 15.

After constructing the fronts, a selection phase is needed to choose among the fronts the solutions that will form $\mathrm{P_{t+1}}$. If the selection is made between two solutions from different fronts, we prefer the solution with the lower (better) rank. Otherwise, if both solutions belong to the same front, then we use the crowding distance operator to decide. More details are described in Algorithm 16.

The crowding distance calculation mechanism works as follows: for each objective function, the boundary solution is assigned an infinite distance value.

Algorithm 15 fast_nondominated_sorting() procedure

input: R_t
for all $p \in R_t$ **do**
  $S_p \leftarrow \emptyset$ //S_p is the set of solutions dominated by p
  $n_p \leftarrow 0$ //n_p is the domination counter of p
  for all $q \in R_t$ **do**
    if $(p < q)$ **then** //if p dominates q
      $S_p \leftarrow S_p \cup \{q\}$
    else if $(q < p)$ **then**
      $n_p \leftarrow n_p + 1$
    end if
  end for
  if $(n_p == 0)$ **then**
    $\mathrm{P}_{\mathrm{rank}} \leftarrow 1$
    $F_1 \leftarrow F_1 \cup \{p\}$ //F_1 represents the first front
  end if
end for
$i \leftarrow 1$
while $(F_i \neq \emptyset)$ **do**
  $Q \leftarrow \emptyset$ //Q will contain the members of the next front
  for all $p \in F_i$ **do**
    for all $q \in S_p$ **do**
      $n_q \leftarrow n_q - 1$
      if $(n_q == 0)$ **then**
        $\mathrm{q}_{\mathrm{rank}} \leftarrow i + 1$
        $Q \leftarrow Q \cup \{q\}$
      end if
    end for
  end for
  $i \leftarrow i + 1$
  $F_i \leftarrow Q$
end while
output: $\{F_1, F_2, ..., F_i\}$

All other intermediate solutions are assigned a distance value calculated as given in eq.6.4.

$$CD_i = \sum_{m=1}^{M} \frac{f_m(x+1) - f_m(x-1)}{f_m(\mathrm{x}_{\mathrm{max}}) - f_m(\mathrm{x}_{\mathrm{min}})} \tag{6.4}$$

Algorithm 16 selection() procedure

input: $\{F_1, F_2, ..., F_i\}$
$i \leftarrow 1$
$cn \leftarrow 0$
while $(F_i \neq \emptyset)$ **do**
    $cn \leftarrow cn + |F_i|$
    if $(cn == N)$ **then**
        P_{t+1} ← individuals which are in $[F_1, ..., F_i]$
        return P_{t+1}
    else if $(cn > N)$ **then**
        P_{t+1} ← individuals which are in $[F_1, ..., F_{i-1}]$
        I_{cr} ← crowding_distance(F_i) //I_{cr} contains the individuals selected using crowding distance calculation
        $P_{t+1} \leftarrow P_{t+1} \cup \{I_{cr}\}$
        return P_{t+1}
    end if
    $i \leftarrow i++$
end while
output:P_{t+1}

where CD_i represents the crowding distance value of the individual i, M represents the number of objectives, $f_m(x+1)$ represents the m^{th} objective function value of the individual $x+1$, and $f_m(x_{max})$ and $f_m(x_{min})$ separately represents the m^{th} objective function value's maximum and minimum value. The selected individuals are those with the maximum value of CD.

6.4 Complexity

In the following, we analyze the time complexity of the proposed algorithm I-NSGA-II. *generate_initial_solution* function is applied to form the initial population P_t, this process takes $O(n+m)$ time. In each iteration of the algorithm, and for every solution in the population, a neighbor is generated either using the *local_search* method, which has a time complexity of $O(n+m)$, or the genetic algorithm, which operates in $O(n)$ time. Thus, the overall time to generate

neighbors in one iteration is $O(sol\_size \times (n+m))$. After neighbors are generated, the algorithm runs two operations: *fast_nondominated_sorting* and *crowding_distance*, which have a combined complexity of $O(sol\_size^2)$, as governed by the nondominated sorting process.
This entire process is repeated for *max_iter* iterations, leading to an overall time complexity of:

$$O(max\_iter \times sol\_size \times (n+m+sol\_size))$$

6.5 Experiments

The proposed algorithm I-NSGA-II was implemented using C++ language, on a PC with an Intel Core i5-1135G7 2.40GHz processor and 8.0 GB of memory. We also implemented three greedy heuristics to evaluate the performance of the proposed algorithm I-NSGA-II.

Benchmark instances used in this work were originally proposed in [108] where each instance consists of an undirected connected vertex-weighted graph with n vertices and m edges. Since our problem requires an edge-weighted graph, we obtain the edge weights by averaging the weights of their endpoints.

The instances are divided into two groups: small and medium instances which include $\{10, 25, 50, 100, 200, 250\}$ vertices, and large instances which contain $\{500, 750, 1000\}$ vertices. The number of edges m is varied for each vertex count n to observe the impact of the degree of connectivity between vertices on the results. For a specific number of nodes and edges, 10 instances exist for most graphs. The obtained results represent the average of running the algorithms on these 10 instances.

The used parameters for the I-NSGA-II algorithm are the following: the population sol_size is set to 100. The maximum number of iterations max_iter is set to 100. The crossover probability is set to 0.8, and the mutation probability to 0.2.

6.6 Results

The experimental results are organized into two parts: the first one corresponds to the results generated by the greedy heuristics and the second represents the results of the proposed algorithm I-NSGA-II versus NSGA-II.

6.6.1 Part I: Results of greedy heuristics

The results obtained by applying the three greedy heuristics described in Section 6.2 are given in Table 6.1 for small and medium instances and those for large instances are presented in Table 6.2. These tables are organized as follows. The first two columns define the instance size, in terms of the number of vertices (n) and the number of edges (m). The results of each greedy heuristic are provided in two columns. The first one with heading **Time** shows the computation time (in seconds), and the second one with heading **HVI** provides the hypervolume indicator.

6.6.1.1 Results for small and medium size instances

From Table 6.1, the comparison between the results of greedy heuristics with regard to execution time, shows that the three algorithms give similar results.
If we consider the hypervolume indicator, it is clearly seen that GR2 obtains the

worst results. GR1 is better than GR2 up to 82.35% and GR3 is better than GR2 at most 88.23%, while GR3 is better than GR1 up to 64.70%.

TABLE 6.1: Results of greedy heuristics for small and medium instances.

n	m	GR1		GR2		GR3	
		Time (s)	HVI	Time (s)	HVI	Time (s)	HVI
10	20	0.000	5540.5	0.000	**5609.2**	0.000	55732.4
	40	0.000	17402.1	0.000	16833.0	0.000	**179022.9**
25	100	0.000	115431.4	0.000	110823.2	0.000	**120076.3**
	250	0.000	**362112.2**	0.000	355097.4	0.000	361091.5
50	100	0.000	104417.4	0.000	**113204.0**	0.000	111520.3
	500	0.001	**1.425E+06**	**0.000**	1.380E+06	0.001	1.392E+06
100	200	0.001	443624.7	**0.000**	408011.6	0.001	**450122.1**
	600	0.002	3.097E+06	**0.001**	2.811E+06	0.002	**3.146E+06**
	1000	0.004	**5.802E+06**	0.004	5.497E+06	**0.003**	5.664E+06
200	400	0.008	**1.733E+06**	0.007	1.400E+06	**0.006**	1.594E+06
	1200	**0.046**	1.160E+07	0.050	1.185E+07	0.051	**1.203E+07**
	2000	0.114	2.255E+07	**0.088**	2.118E+07	0.098	**2.294E+07**
250	500	**0.047**	**2.820E+06**	0.049	2.771E+06	0.053	2.801E+06
	1000	0.089	9.511E+06	**0.085**	9.208E+06	0.086	**9.744E+06**
	1500	0.110	1.621E+07	**0.107**	1.548E+07	0.119	**1.733E+07**
	2000	**0.139**	**2.721E+07**	0.146	2.531E+07	0.145	2.600E+07
	2500	0.179	3.386E+07	0.185	3.012E+07	**0.166**	**3.561E+07**
Average		0.043	8.050E+06	**0.042**	7.496E+06	0.043	**8.556E+06**

6.6.1.2 Results for large size instances

It can be observed from Table 6.2 that the time results of GR1 are generally the worst. GR2 and GR3 are faster than GR1 in 11 instances. GR3 executes 10 times faster than GR2. Concerning the hypervolume indicator, GR2 is the worst, GR1 and GR3 are better than GR2 for all datasets. GR3 is better than GR1 in 9 out of 15 cases.

From the previous results, we can conclude that GR3 is the best heuristic with regard to both execution time and hypervolume indicator in small and medium, and large size problem instances. Thus, this heuristic will be used in our proposed algorithm I-NSGA-II.

TABLE 6.2: Results of greedy heuristics for large instances.

n	m	GR1		GR2		GR3	
		Time (s)	HVI	Time (s)	HVI	Time (s)	HVI
500	1000	0.390	**1.023E+07**	0.383	9.541E+06	**0.368**	9.836E+06
	2000	0.504	4.167E+07	**0.452**	3.804E+07	0.506	**4.255E+07**
	3000	1.031	7.250E+07	**0.845**	7.018E+07	1.119	**7.291E+07**
	4000	1.235	8.444E+07	1.172	8.125E+07	**1.149**	**9.423E+07**
	5000	**1.007**	**1.178E+08**	1.273	8.940E+07	1.289	1.150E+08
750	1500	1.365	**2.105E+07**	1.524	1.763E+07	**1.188**	1.845E+07
	3000	1.498	7.447E+07	1.449	7.021E+07	**1.131**	**8.359E+07**
	4500	1.586	1.465E+08	1.547	1.381E+08	**1.477**	**1.609E+08**
	6000	1.710	**2.322E+08**	**1.604**	1.755E+08	1.674	2.246E+08
	7500	1.853	2.841E+08	1.701	2.530E+08	**1.690**	**3.009E+08**
1000	2000	**1.924**	**4.211E+07**	2.038	3.891E+07	1.964	4.203E+07
	4000	2.371	1.462E+08	2.415	1.224E+08	**2.290**	**1.503E+08**
	6000	2.675	2.405E+08	2.552	2.184E+08	**2.540**	**2.655E+08**
	8000	2.640	**4.015E+08**	2.623	3.414E+08	**2.512**	3.827E+08
	10000	2.802	4.277E+08	**2.765**	3.886E+08	2.786	**4.321E+08**
Average		1.639	1.561E+08	1.622	1.366E+08	**1.578**	**1.608E+08**

6.6.2 Part II: Results of I-NSGA-II versus NSGA-II

Table 6.3 reported the results of I-NSGA-II and NSGA-II for small and medium instances and the results for large instances are presented in Table 6.4. In C-metric, 1 refers to I-NSGA-II, and 2 refer to NSGA-II, that is C(1,2) means C(I-NSGA-II,NSGA-II), and C(2,1) means C(I-NSGA-II,NSGA-II).

Figures 6.2 and 6.3 visualize the approximate Pareto front for the two objective functions F_1 and F_2 for small and medium instances and large instances respectively.

TABLE 6.3: Results of I-NSGA-II and NSGA-II for small and medium instances.

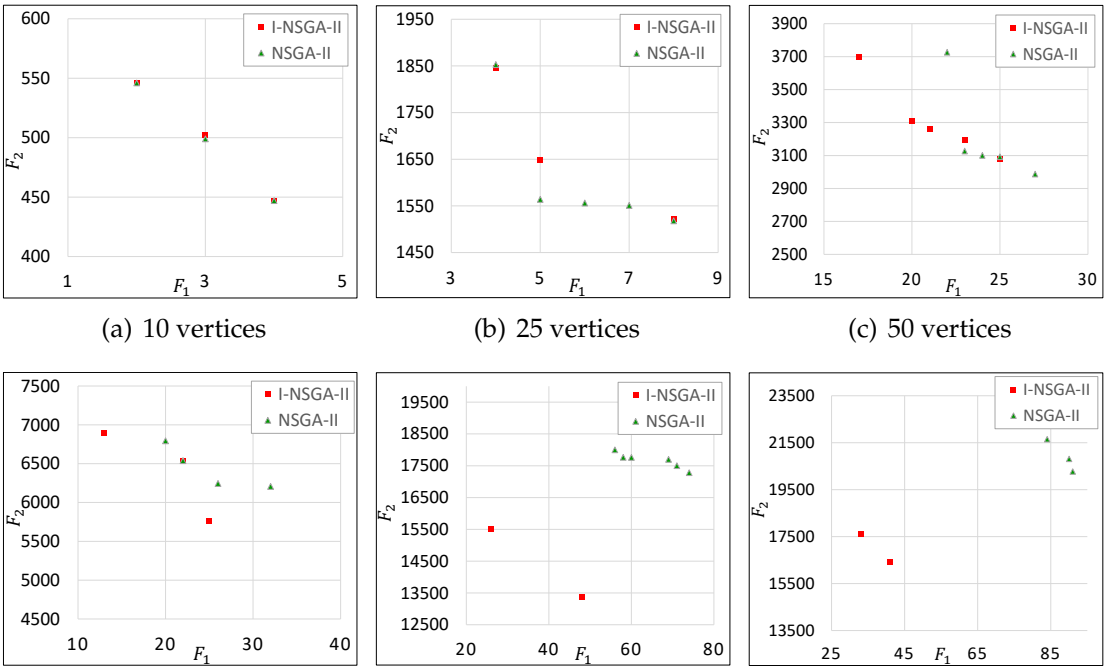
n	m	I-NSGA-II			NSGA-II		
		Time	HVI	C(1,2)	Time	HVI	C(2,1)
10	20	**19.992**	**6067.7**	0.616	21.119	6007.4	**0.950**
	40	19.643	**19762**	0.866	**19.012**	19583.9	**0.950**
25	100	**20.964**	**126476**	0.412	26.001	126318.3	**0.419**
	250	**18.214**	**381058.5**	0.683	21.021	378286.3	**0.800**
50	100	**25.281**	**125806.4**	**0.548**	31.910	105129.2	0.116
	500	**23.290**	**1.487E+06**	**0.475**	28.266	1.453E+06	0.300
100	200	**31.076**	**476289**	**0.372**	47.518	338618.1	0.350
	600	**31.520**	**3.156E+06**	**0.800**	38.186	2.748E+06	0
	1000	26.896	**5.843E+06**	**0.625**	**24.684**	5.471E+06	0.062
200	400	125.586	**1.846E+06**	**1**	**114.864**	984906	0
	1200	**50.662**	**1.248E+07**	**1**	81.229	9.767E+06	0
	2000	**36.313**	**2.312E+07**	**1**	53.2277	2.061E+07	0
250	500	**134.257**	**2.855E+06**	**1**	223.215	1.326E+06	0
	1000	**103.687**	**1.077E+07**	**1**	182.261	7.052E+06	0
	1500	**96.514**	**1.904E+07**	**1**	144.363	1.460E+07	0
	2000	**51.075**	**2.824E+07**	**1**	105.997	2.352E+07	0
	2500	**47.800**	**3.792E+07**	**1**	97.759	3.359E+07	0
Average		**50.751**	**8.699E+06**	**0.788**	74.154	7.182E+06	0.232

6.6.2.1 Results for small and medium size instances

From Table 6.3 is clearly seen that I-NSGA-II runs faster in 14 out of 17 cases compared with NSGA-II. The hypervolume indicator results of I-NSGA-II are

better than NSGA-II results in all cases. Regarding the C-metric values obtained, we can see that most solutions found by I-NSGA-II are better than those in NSGA-II, especially when the number of vertices increases.

With respect to F_1 and F_2 values, we can observe from Figure 6.2 that I-NSGA-II gives better results than NSGA-II in all the problem instances, except scenarios (n=10 and n=25).



(a) 10 vertices (b) 25 vertices (c) 50 vertices

(d) 100 vertices (e) 200 vertices (f) 250 vertices

FIGURE 6.2: Approximate Pareto fronts produced by I-NSGA-II and NSGA-II for small and medium instances.

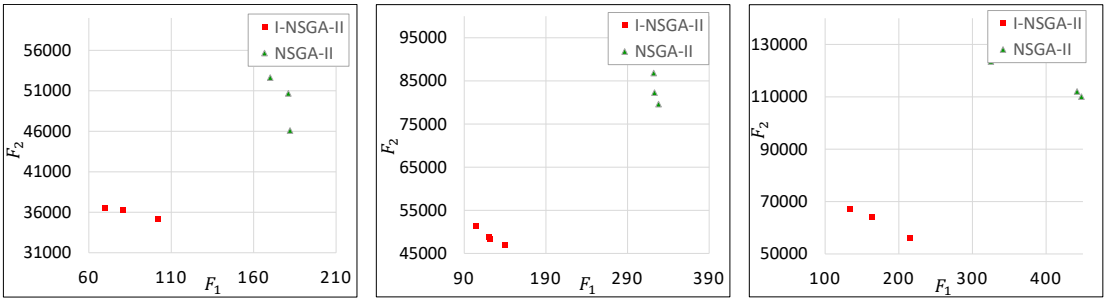
6.6.2.2 Results for large size instances

As shown in Table 6.4, I-NSGA-II performs better than NSGA-II in all instances with respect to the hypervolume indicator. In terms of the run time, I-NSGA-II obtains better results up to 93.33%. When considering the C-metric, it is evident that all solutions produced by NSGA-II are dominated by at least one solution

obtained by NSLS. Figure 6.3 shows that solutions produced by I-NSGA-II distribute with good diversity and dominate those of NSGA-II.

TABLE 6.4: Results of I-NSGA-II and NSGA-II for large instances.

n	m	I-NSGA-II			NSGA-II		
		Time	HVI	C(1,2)	Time	HVI	C(2,1)
500	1000	**565.842**	**1.160E+07**	**1**	857.378	3.056E+06	0
	2000	**646.827**	**4.371E+07**	**1**	968.586	2.634E+07	0
	3000	**301.842**	**7.548E+07**	**1**	543.483	4.970E+07	0
	4000	**232.733**	**1.157E+08**	**1**	326.853	8.834E+07	0
	5000	**180.837**	**1.439E+08**	**1**	238.011	1.170E+08	0
750	1500	**1541.980**	**2.517E+07**	**1**	1869.190	5.480E+06	0
	3000	**1587.260**	**9.485E+07**	**1**	2691.110	5.016E+07	0
	4500	**1084.457**	**1.738E+08**	**1**	1250.645	1.111E+08	0
	6000	**711.109**	**2.556E+08**	**1**	744.935	1.883E+08	0
	7500	**354.333**	**3.264E+08**	**1**	547.698	2.635E+08	0
1000	2000	3060.541	**4.578E+07**	**1**	**3003.330**	8.953E+06	0
	4000	**2754.252**	**1.722E+08**	**1**	3164.494	8.141E+07	0
	6000	2082.053	**3.087E+08**	**1**	**2052.498**	1.918E+08	0
	8000	**1103.110**	**4.395E+08**	**1**	1147.286	3.155E+08	0
	10000	**749.328**	**5.855E+08**	**1**	812.379	4.474E+08	0
Average		**1130.420**	**1.878E+08**	**1**	1347.858	1.298E+08	0



(a) 500 vertices (b) 750 vertices (c) 1000 vertices

FIGURE 6.3: Approximate Pareto fronts produced by I-NSGA-II and NSGA-II for large instances.

6.7 Conclusion

To solve the bi-objective problem MWMCDS, we developed an improved version of NSGA-II referred to as I-NSGA-II. The main feature of I-NSGA-II lies in the fact that greedy heuristic is used both for generating initial solutions and for exploring local search neighborhoods. Furthermore, we have implemented three greedy heuristics for the problem and the best performing one (on average) among them is chosen to be used in the development of our approach.

The performance of I-NSGA-II is evaluated on a set of test problem instances with different sizes. Computational experiments show a significant improvement in our approach over NSGA-II with respect to the hypervolume indicator, run-time, and quality of solutions.

Chapter 7

Multiobjective Greedy Simulated Annealing (MGSA) Algorithm for MWMCDSP

7.1 Introduction

This chapter presents the Multiobjective Greedy Simulated Annealing (MGSA) algorithm which is developed as a novel approach to solving the MWMCDSP. We first describe the greedy paradigm, then we provide the general framework of MGSA followed by details of each step. After that, we discuss the computational complexity of the algorithm. Finally, we present a thorough experimental evaluation to compare its performance against existing state-of-the-art techniques.

7.2 Greedy heuristic

To design a greedy heuristic for the minimum weight minimum connected dominating set problem, we need to consider both objectives: minimizing cardinality ($F_1(D)$) and minimizing total weight ($F_2(D)$). The heuristic follow the process outlined in Section 5.2.1.

The selection of vertices is based on a combined score that considers both the weight and the cardinality as given in Eq. 7.1. A higher value is considered better.

$$S(v) = \alpha \cdot GD'(v) - (1 - \alpha) \cdot GW'(v) \tag{7.1}$$

Where α is a parameter that allows adjusting the importance given to each objective. A higher α value emphasizes weight reduction, while a lower value prioritizes minimizing cardinality. GD' and GW' are the normalized values of GD and GW respectively.

$GD(v)$ counts for a vertex v the number of WHITE neighbors, GD is then calculated as given in Eq. 7.2.

$$GD(v) = \{d_s(v) \mid v \in V^{cand}\} \tag{7.2}$$

$GW(v)$ (Eq. 7.3) represents the weight to be added to F_2 upon the inclusion of vertex v in the solution ($GW(v)$ may have a positive or negative value). Hence, the best vertex is the one with the lowest value of GW.

$$GW(v) := wt_{1(v)} - \sum_{(v \in V^{\text{cand}} \wedge u \in V^{\text{cand}} | (v,u) \in E)} wt_{2(v,u)} \tag{7.3}$$

$$wt_{1(v)} = \sum_{((u,v) \in E, u \in D \wedge v \in V^{\text{cand}})} w_{(u,v)}$$

$$wt_{2\_1(u,u')} = \min\{w_{(u,u')} \mid (u,u') \in E, u \in V^{\text{cand}} \wedge u' \in D\}$$

$$\text{if } wt_{2\_1(u,u')} > w_{(v,u)} : \; wt_{2(v,u)} = wt_{2\_1(u,u')} - w_{(v,u)}$$

$$\text{else: } wt_{2(v,u)} = 0$$

7.3 MGSA framework

The Multiobjective Greedy Simulated Annealing (MGSA) algorithm is developed to tackle the MWMCDS problem. The concept of Pareto optimality is applied to evaluate multiobjective solutions and store the nondominated ones in the approximate Pareto front. A novel efficient greedy heuristic is proposed to seed the algorithm with a good initial solution as well as in generating neighbors. Moreover, MGSA uses useful methods for changing temperature and for accepting solutions. After a maximum number of iterations, the obtained approximate Pareto set will be improved by eliminating the redundant vertices. The flowchart of the MGSA algorithm is illustrated in Figure 7.1, and the pseudocode is given in Algorithm 17.

7.3.1 Initial solution

The initial solution is generated using the greedy heuristic defined in Section 7.2 which represents a combined score that considers both the weight and the cardinality. By starting with a strong initial solution, the algorithm can more

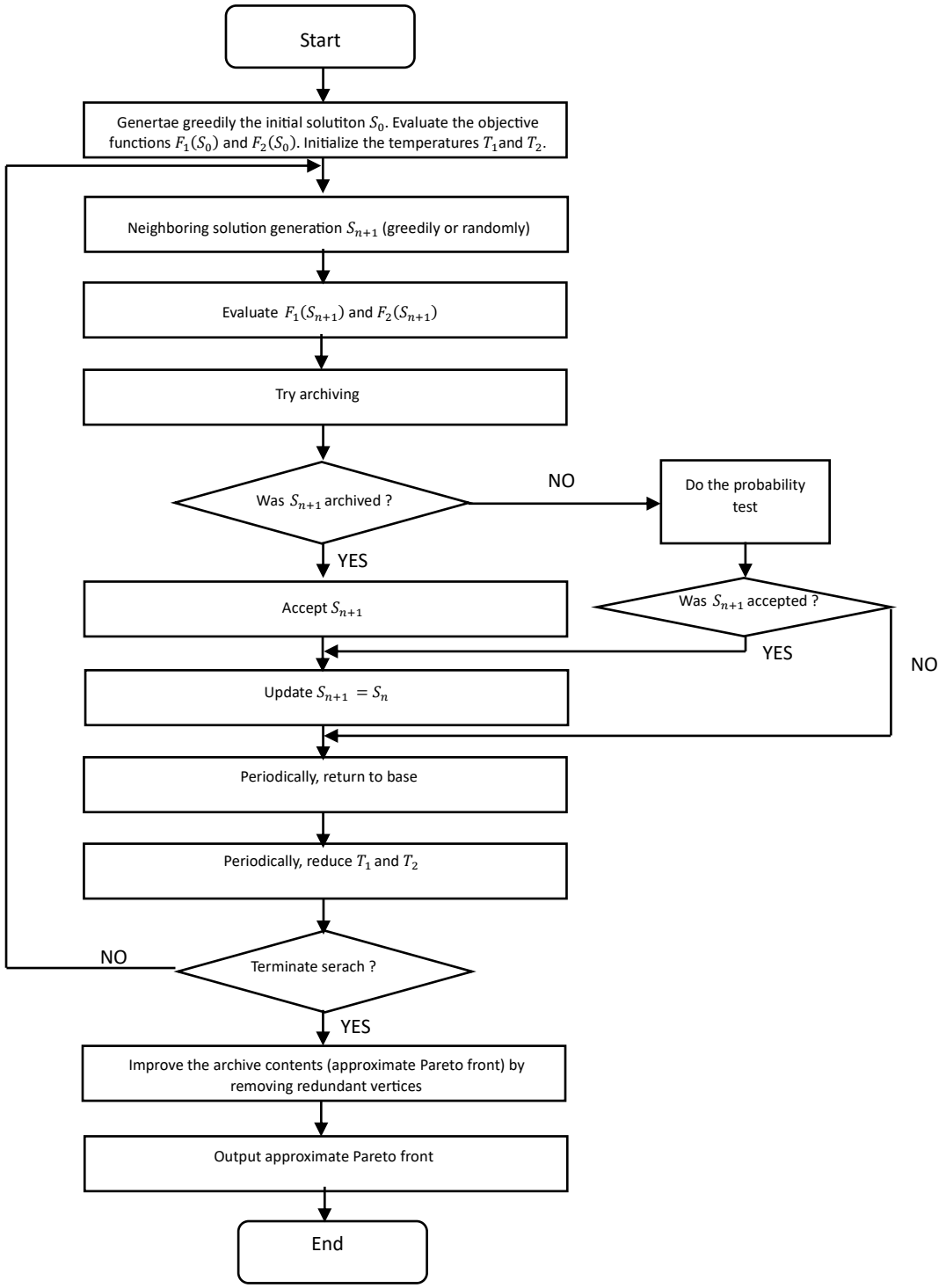


FIGURE 7.1: MGSA algorithm for MWMCDS problem.

Algorithm 17 MGSA for the MWMCDS problem

input: A problem instance (G, V, E, w), and parameters: maximum number of iterations $maxIter$, initial temperatures T_{01} and T_{02}, predetermined numbers of iterations used in the annealing schedule N_1 and N_2. The number of acceptances N_A, number of iterations to be executed before the first return to base N_{B0}, base return parameter r_B.
$T_1 \leftarrow T_{01}$
$T_2 \leftarrow T_{02}$
generate greedily the initial solution S_0
$archive \leftarrow S_0$
$S \leftarrow S_0$
for $i = 0$ to $maxIter$ **do**
    generate the neighbor $S_{new} = N(s)$ greedily or randomly with equal probability
    if (S_{new} is not dominated by any solution from $archive$) **then**
        $archive \leftarrow archive \cup \{S_{new}\}$
        $S \leftarrow S_{new}$
    else if (S_{new} verify the acceptance probability p) **then**
        $S \leftarrow S_{new}$
    end if
    periodically, return to base based on N_1, N_{Bi}, and r_B values
    periodically, reduce T_1 and T_2 based on N_1, N_2, and N_A values
end for
remove redundant vertices from $archive$
output: $archive$ that represents an approximate Pareto front

efficiently navigate the solution space and improve its overall performance in finding the best possible outcomes.

7.3.2 Neighbors production

A neighboring candidate solution is determined by using either the greedy approach or a random method, each with an equal probability. The greedy approach involves selecting the most promising solution available at each step, aiming to make the most immediate progress. In contrast, the random method selects solutions without any specific preference, introducing an element of

chance to the process. This combination of strategies helps balance the exploration of the solution space, potentially leading to more optimal solutions.

7.3.3 Archiving and acceptance

In the archiving procedure, if the new solution is dominated by any members of the archive, it is not archived, elsewhere, it is archived and the archive is updated by removing dominated solutions. All archived solutions are accepted. If a solution is not archived, then it is accepted with a probability given by

$$p = \prod_{i=1}^{2} \exp\left(-\frac{[f_i(s_{n+1}) - f_i(S_n)]}{T_i}\right)$$

Thus, the overall acceptance probability is the product of individual acceptance probabilities for each objective, and therefore, each objective is assigned an associated temperature T_i, which obviates the need to scale the objectives carefully with respect to each other, as long as suitable temperatures can be determined automatically, as described in the next section.

7.3.4 Annealing schedule

In our algorithm we adopted the annealing schedule proposed in [51]. Initially, all temperatures are initialized to large values, hence, all feasible solutions are accepted. A statistical record is maintained for each observed objective function value. After a pre-determined number of iterations N_{T1}, the temperatures T_1 and T_2 are set equal to the standard deviation σ_i, of the accepted values of F_1 and F_2, respectively, i.e.,

$$T_i = \sigma_i$$

Thereafter, after reaching either a specified number of iterations N_{T2}, or a certain number of acceptances N_A, the temperatures are lowered according to the formula:

$$T_i' = \alpha_i T_i$$

Where T_i' denotes the updated temperature, and α_i is computed using the formulation proposed in [109]:

$$\alpha_i = \max\left(0.5, \exp\left[-\frac{0.7T_i}{\sigma_i}\right]\right)$$

In this expression, σ_i represents the standard deviation of f_i values for the accepted solutions at temperature T_i. Subsequently, both counters for N_{T2} and N_A are reset to zero.

7.3.5 Return to base

In order to completely expose the trade-off between objectives, the periodic random selection of a solution from the archive, from which to recommence the search, is done as follows.

Following the initiation of the search process, the activation of a return-to-base occurs once the fundamental aspects of the trade-off between objectives have been established. It is prudent for this activation to coincide with the initial reduction in temperatures, specifically after N_{T1} iterations. Thereafter, the rate of return is naturally heightened to enhance the exploration within the trade-off. The number of iterations N_{Bi} to be executed before the i th return-to-base after the start of the search is given by

$$N_{Bi} = r_B N_{Bi-1}, \quad i = 2, 3, 4, \ldots$$

Where r_B is a parameter ranging from 0 to 1 which determines the frequency of return. Naturally, N_{Bi} cannot decrease indefinitely, and thus a lower bound for N_{Bi} is established to ensure $N_{Bi} \geq 10$.

7.4 Complexity

Several important steps that the Multiobjective Greedy Simulated Annealing (MGSA) algorithm involves determine its overall time complexity. The initial solution is created using the greedy heuristic which is performed in $O(n + m)$ time. Generating the neighbors is done greedily or randomly with equal probability requires $O(n + m)$. Archiving and acceptance checks require comparing the new solution with existing ones in the archive, this is done in $O(n \times |\text{archive}|)$ time. The annealing schedule involves constant-time operations $O(1)$, and the return-to-base mechanism involves resetting counters, which is $O(|\text{archive}|)$. Consequently, for max_iter iterations, the overall complexity becomes

$$O((n + m) + \text{max\_iter} \times (n + m + n \times |\text{archive}| + |\text{archive}|))$$

This simplifies to

$$O(\text{max\_iter} \times (n + m))$$

as |archive| is small compared to max_iter.

7.5 Experimental evaluation

The proposed algorithm MGSA was implemented using C++ language. The experimental results were obtained on a PC with an Intel Core i5-1135G7 2.40GHz processor and 8.0 GB of memory.

The used parameters for the MGSA algorithm are the following: the maximum number of iterations is defined as 1000, the initial temperatures are set to $T_1 = T_2 = 1000$, the predetermined numbers of iterations used in the annealing schedule are $N_{T_1} = 200$ and $N_{T_2} = 100$. The number of acceptances is $N_A = 10$, the number of iterations to be executed before the first return to base is $N_{B_0} = 50$, with a base return parameter of $r_B = 0.9$.

To benchmark our algorithm we utilized two distinct datasets. The first one comprises instances originally proposed by [13], featuring undirected edge-weighted graphs. While the second set, as suggested by [104], presents undirected vertex-weighted graphs. Given that the MWMCDS problem necessitates an edge-weighted graph, we derived edge weights by averaging the weights of their endpoints. Details on these datasets can be found in Sections 5.7, and 6.5.

To measure the performance in the second dataset, we make use of the hypervolume indicator, C-metric, the execution time and the approximate Pareto fronts.

7.6 Results

The performance of MGSA was compared against current state-of-the-art approaches, namely MOGA [13], MCDS [14], GSA [12], and I-NSGA-II [15].

7.6.1 Comparison in the first dataset

The performance of MGSA against MCDS, MOGA, and GSA with respect to energy consumption for 100 instances of data transfer is shown in Figure 7.2. It is evident from the figure that MGSA consumes the least amount of energy in all cases.

Figure 7.3 represents the number of dominator vertices produced by MGSA, MOGA, MCDS, and GSA for different problem instances. It can be seen that MGSA outperforms MCDS in 7 out of 9 cases while yielding identical results in the remaining two cases. When compared to MOGA, MGSA demonstrates equivalent performance in all instances except for n=80, n=90, and n=100, where MGSA exhibits superior performance. Against GSA, MGSA achieves identical results in 8 instances but performs worse in the cases of n=20 and n=50.

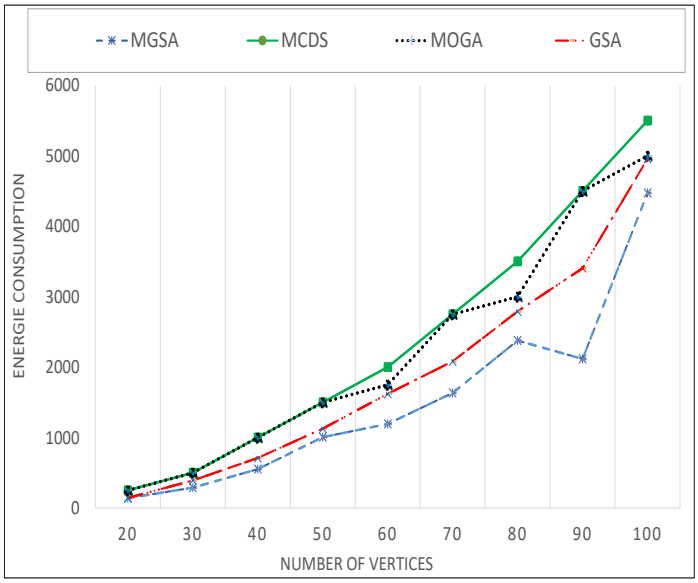


FIGURE 7.2: Energy consumption in MGSA, MOGA, MCDS, and GSA.

7.6.2 Comparison in the second dataset

The comparative analysis in terms of time, HVI, and C-metric between the algorithms MGSA and I-NSGA-II across small and medium instances are presented in Table 7.1, and for large instances in Table 7.2. In the context of the C-metric, "1" refers to MGSA, and "2" refers to I-NSGA-II. Therefore, C(1,2) represents C(MGSA, I-NSGA-II), and C(2,1) represents C(I-NSGA-II, MGSA).

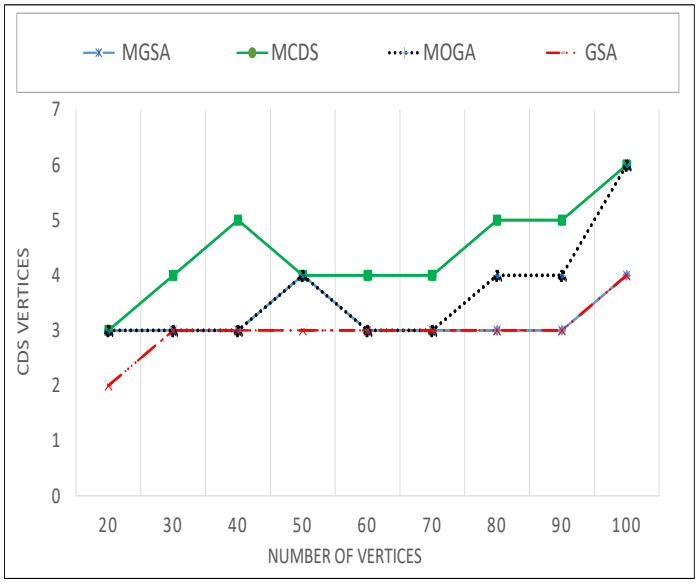


FIGURE 7.3: Size of CDS in MGSA, MOGA, MCDS, and GSA.

Table 7.1 reveals that MGSA outperforms I-NSGA-II in terms of execution time for all instances. Considering HVI, MGSA is better than I-NSGA-II in 10 out of 17 instances. With respect to C-metric, MGSA outperforms I-NSGA-II in 13 cases with total domination in 8 of them.

Examining Table 7.2, the evaluation of MGSA and I-NSGA-II in large instances highlights that MGSA maintains its superiority in time efficiency in all cases and 12 out of 15 cases regarding HVI. Regarding the C-metric values, we can see that MGSA frequently achieves higher values, indicating it often produces better solutions than I-NSGA-II.

Figures 7.4 and 7.5 depict the approximate Pareto fronts derived from MGSA and I-NSGA-II methods for small and medium instances, as well as large instances, respectively. It is clearly seen that MGSA consistently outperforms I-NSGA-II in all cases, except for instances (10, 20) and (25, 100) where the results are incomparable.

TABLE 7.1: Results of MGSA and I-NSGA-II for small and medium instances.

n	m	MGSA			I-NSGA-II		
		Time (s)	HVI	C(1,2)	Time (s)	HVI	C(2,1)
10	20	**0.113**	5768	0.450	19.992	**6067**	**0.754**
	40	**0.078**	18498	0.525	19.643	**19762**	**0.730**
25	100	**0.358**	126434	0.418	20.964	**126476**	**0.600**
	250	**0.366**	**386878**	**0.810**	18.214	381058	0.340
50	100	**0.649**	125287	**0.548**	25.281	**125806**	0.536
	500	**0.974**	**1.493E+06**	**1**	23.290	1.487E+06	0.090
100	200	**1.962**	**494505**	**1**	31.076	476289	0.067
	600	**2.390**	**3.157E+06**	**0.732**	31.520	3.156E+06	0.406
	1000	**7.184**	5.840E+06	**0.633**	26.896	**5.843E+06**	0.574
200	400	**8.773**	**1.954E+06**	**1**	125.586	1.846E+06	0
	1200	**16.069**	**1.263E+07**	**1**	50.662	1.248E+07	0
	2000	**9.847**	2.286E+07	0.428	36.313	**2.312E+07**	**0.720**
250	500	**9.095**	**2.994E+06**	**1**	134.257	2.855E+06	0
	1000	**11.697**	**1.114E+07**	**1**	103.687	1.077E+07	0
	1500	**11.627**	**1.937E+07**	**0.954**	95.514	1.904E+07	0.115
	2000	**13.923**	**2.854E+07**	**1**	51.075	2.824E+07	0
	2500	**16.968**	3.761E+07	**1**	47.800	**3.792E+07**	0
Average		**6.592**	**8.750E+06**	**0.794**	50.751	8.699E+06	0.290

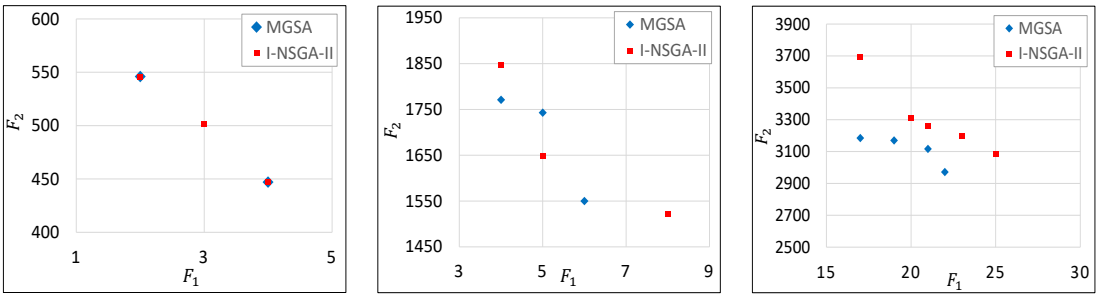
The previous results showcase the ability of MGSA to deliver solutions quickly while taking into account their quality.

TABLE 7.2: Results of MGSA and I-NSGA-II for large instances.

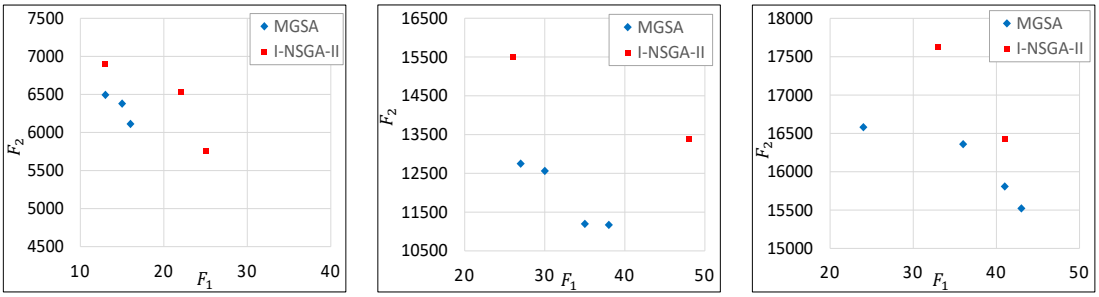
n	m	MGSA			I-NSGA-II		
		Time (s)	HVI	C(1,2)	Time (s)	HVI	C(2,1)
500	1000	**28.838**	**1.165E+07**	**1**	565.842	1.160E+07	0
	2000	**36.759**	**4.530E+07**	**1**	646.827	4.371E+07	0
	3000	**96.697**	**7.762E+07**	**1**	301.842	7.548E+07	0
	4000	**46.037**	1.154E+08	**0.682**	232.733	**1.157E+08**	0.526
	5000	**44.169**	**1.447E+08**	**0.814**	180.837	1.439E+08	0.241
750	1500	**75.306**	**2.727E+07**	**1**	1541.980	2.517E+07	0
	3000	**88.287**	**9.526E+07**	**0.832**	1587.260	9.485E+07	0.368
	4500	**100.016**	**1.774E+08**	**1**	1084.457	1.738E+08	0
	6000	**114.328**	2.501E+08	0.420	711.109	**2.556E+08**	**0.607**
	7500	**108.594**	**3.273E+08**	**0.790**	354.333	3.264E+08	0.342
1000	2000	**102.763**	**4.622E+07**	**0.736**	3060.541	4.578E+07	0.412
	4000	**163.637**	**1.749E+08**	**1**	2754.252	1.722E+08	0
	6000	**152.561**	**3.116E+08**	**0.820**	2082.053	3.087E+08	0.317
	8000	**145.614**	**4.419E+08**	**0.906**	1103.110	4.395E+08	0.208
	10000	**200.758**	5.815E+08	0.394	749.328	**5.855E+08**	**0.640**
Average		**100.290**	**1.885E+08**	**0.826**	1130.420	1.878E+08	0.244

7.7 Conclusion

In this chapter, we introduce a novel method called Multi-Objective Greedy Simulated Annealing algorithm (MGSA) for tackling the minimum weight minimum connected dominating set (MWMCDS) problem. MGSA integrates a simulated annealing algorithm, which is seeded by a combined greedy heuristic that considers both the weight and the cardinality reduction. The performance of the proposed algorithm was compared with existing state-of-the-art techniques, including MOGA, MCDS, GSA, and I-NSGA-II. The comparison was based on energy consumption and the size of CDS in the data transfer system for the first dataset, and on hypervolume indicator, C-metric, runtime, and

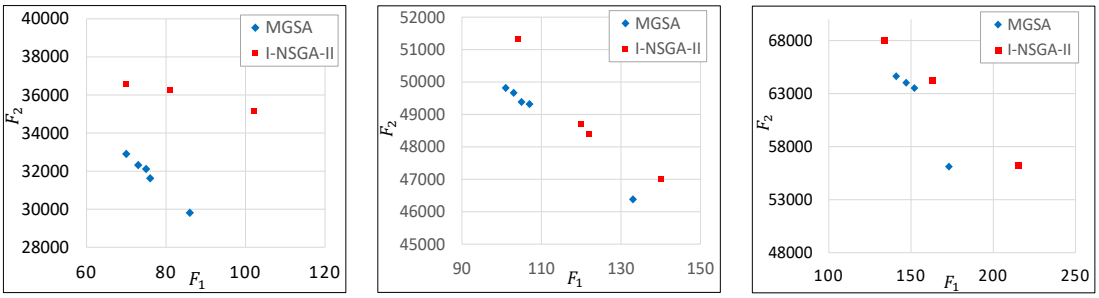


(a) (10,20) (b) (25,100) (c) (50,100)



(d) (100,600) (e) (200,1200) (f) (250,1500)

FIGURE 7.4: Approximate Pareto fronts produced by MGSA and I-NSGA-II for small and medium instances.



(a) (500,3000) (b) (750,4500) (c) (1000,6000)

FIGURE 7.5: Approximate Pareto fronts produced by MGSA and I-NSGA-II for large instances.

approximate Pareto fronts for the second dataset. The obtained results demonstrate the effectiveness of the MGSA algorithm.

Chapter 8

Conclusions and Future Works

8.1 Conclusions

This thesis has focused on the development of bi-objective modeling and optimization techniques using greedy stochastic algorithms to address the Minimum Weight Minimum Connected Dominating Set (MWMCDS) problem; a significant NP-hard problem in graph theory that aims to minimize the cardinality and the total edge-weight of the generated connected dominating set. The primary challenge tackled by this work has been the simultaneous minimization of often conflicting objectives while maintaining the quality of solutions. Another challenge is the lack of works that has studied this problem (one work to the best of our knowledge) which requires greater efforts to find suitable algorithms and techniques to solve it.

The core contributions of this work include the development of three algorithms called GSA, I-NSGA-II, and MGSA, each introducing an efficient greedy heuristic to solve the MWMCDSP. These algorithms were tested on one or both of these datasets [13, 104].

Greedy Simulated Annealing (GSA): The GSA algorithm integrates a greedy

heuristic with the simulated annealing framework, based on an aggregated objective function to guide the search process. Moreover, GSA uses a useful calculation method for changing temperature. The performance of the proposed algorithm was assessed and compared against a recent multiobjective genetic algorithm named MOGA [13], and an MCDS produced in [14]. The experimental results compared to those obtained show the superiority of GSA.

Improved Non-dominated Sorting Genetic Algorithm (I-NSGA-II): In this work, we introduce an improved version of NSGA-II. First, we developed three greedy heuristics for the problem and the best performing one among them was chosen to be used in our approach. The I-NSGA-II algorithm incorporates the selected greedy heuristic to initial population generation and neighbor production. Fast nondominated sorting and crowding distance procedures are used to sort and select vertices for the next generation. This algorithm has demonstrated superior performance compared to the traditional NSGA-II [78] with regard to solution quality and computation time.

Multiobjective Greedy Simulated Annealing (MGSA): MGSA is a multiobjective simulated annealing algorithm based on the Pareto optimization technique, seeded by a combined greedy heuristic that considers both the weight and the cardinality reduction. Neighbors are generated either greedily or randomly with equal probability. Moreover, MGSA uses useful methods for changing temperature and for accepting solutions. The final approximate Pareto front was refined by eliminating redundant vertices. The performance of MGSA was benchmarked against recent algorithms from the literature, namely MOGA [13], MCDS [14], GSA [12], and I-NSGA-II [15]. The results obtained demonstrate the superiority of MGSA.

8.2 Future Works

Although this thesis has made significant strides in addressing the challenges associated with the MWMCDS problem, where the proposed algorithms have demonstrated their effectiveness in solving it, there are many important points that we plan to carry out to strengthen our approaches. Our future perspectives are summarized as follows:

- Develop more advanced greedy heuristics tailored specifically to the MWMCDS problem.
- Use multiple greedy heuristics simultaneously.
- Integrate effective local search strategies such as tabu search, variable neighborhood search (VNS), and ant colony optimization (ACO).
- Apply the proposed algorithms to other bi-objective problems as well especially those that are similar to MWMCDS.
- Extend the application to solve optimization problems with more than two objectives.
- Apply the MWMCDS problem to other real-world networks, such as wireless sensor networks.

Bibliography

[1] Massimiliano Caramia and Paolo Dell'Olmo. "Multi-objective optimization". In: *Multi-objective management in freight logistics*. Springer, 2020, pp. 21–51 (cit. on p. 2).

[2] Fred W Glover and Gary A Kochenberger. *Handbook of metaheuristics*. Vol. 57. Springer Science & Business Media, 2006 (cit. on p. 2).

[3] Ibrahim H Osman and Gilbert Laporte. *Metaheuristics: A bibliography*. 1996 (cit. on pp. 2, 16).

[4] Jeremy Blum et al. "Connected dominating set in sensor networks and MANETs". In: *Handbook of combinatorial optimization*. Springer, 2004, pp. 329–369 (cit. on p. 2).

[5] A Chinnasamy et al. "Minimum connected dominating set based RSU allocation for smartCloud vehicles in VANET". In: *Cluster Computing* 22.5 (2019), pp. 12795–12804 (cit. on p. 2).

[6] Rajiv Misra and Chittaranjan Mandal. "Minimum connected dominating set using a collaborative cover heuristic for ad hoc sensor networks". In: *IEEE Transactions on parallel and distributed systems* 21.3 (2009), pp. 292–302 (cit. on p. 2).

[7] Kaushal K Shukla and S Sah. "Construction and maintenance of virtual backbone in wireless networks". In: *Wireless networks* 19.5 (2013), pp. 969–984 (cit. on p. 2).

[8] Igor Jurisica, Dennis A Wigle, and Bill Wong. "Cancer informatics in the post genomic era: toward information-based medicine". In: (2007) (cit. on p. 2).

[9] M Nehéz and J Mäsiar. "On Propagation in Social Networks—A Graph Mining Approach". In: *Proc. of the 6th Int. Workshop on Knowledge Management*. 2011 (cit. on p. 2).

[10] Haiying Wang et al. "Minimum dominating sets in cell cycle specific protein interaction networks". In: *2014 IEEE International Conference on Bioinformatics and Biomedicine (BIBM)*. IEEE. 2014, pp. 25–30 (cit. on p. 2).

[11] Stefan Wuchty. "Controllability in protein interaction networks". In: *Proceedings of the National Academy of Sciences* 111.19 (2014), pp. 7156–7160 (cit. on p. 2).

[12] Hayet Dahmri and Salim Bouamama. "A Simulated Annealing-Based Multiobjective Optimization Algorithm for Minimum Weight Minimum Connected Dominating Set Problem". In: *arXiv preprint arXiv:2312.11527* (2023) (cit. on pp. 3, 4, 60, 73, 96, 103).

[13] Dinesh Rengaswamy, Subham Datta, and Subramanian Ramalingam. "Multiobjective genetic algorithm for minimum weight minimum connected dominating set". In: *International Conference on Intelligent Systems Design and Applications*. Springer. 2017, pp. 558–567 (cit. on pp. 3, 4, 55, 60, 69, 96, 102, 103).

[14] Maryam Nazarieh et al. "Identification of key player genes in gene regulatory networks". In: *BMC systems biology* 10.1 (2016), pp. 1–12 (cit. on pp. 3, 4, 60, 69, 96, 103).

[15] Hayet Dahmri and Salim Bouamama. "Improved NSGA-II for Minimum Weight Minimum Connected Dominating Set Problem". In: *International Symposium on Modelling and Implementation of Complex Systems*. Springer. 2020, pp. 248–261 (cit. on pp. 3, 4, 60, 96, 103).

[16] Hayet Dahmri, Salim Bouamama, and Samir Balbal. "A Greedy Simulated Annealing-Based Multiobjective Algorithm for Minimum Weight Minimum Connected Dominating Set Problem". In: *Engineering, Technology & Applied Science Research* 14.5 (2024), pp. 16686–16691 (cit. on pp. 4, 61).

[17] Thomas Back. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford university press, 1996 (cit. on p. 7).

[18] Carlos A Coello Coello, Gary B Lamont, David A Van Veldhuizen, et al. *Evolutionary algorithms for solving multi-objective problems*. Vol. 5. Springer, 2007 (cit. on pp. 7, 33, 37, 38, 42).

[19] Xin-She Yang. *Engineering optimization: an introduction with metaheuristic applications*. John Wiley & Sons, 2010 (cit. on p. 8).

[20] Qian Jin and Mauro Overend. "Facade renovation for a public building based on a whole-life value approach". In: *Proceedings of building simulation and optimisation conference, Loughborough, UK*. Citeseer. 2012, 378e85 (cit. on p. 8).

[21] Fanny Pernodet Chantrelle et al. "Development of a multicriteria tool for optimizing the renovation of buildings". In: *Applied Energy* 88.4 (2011), pp. 1386–1394 (cit. on p. 8).

[22] Sanghamitra Bandyopadhyay and Sriparna Saha. "Some single-and multiobjective optimization techniques". In: *Unsupervised Classification*. Springer, 2013, pp. 17–58 (cit. on pp. 9, 10).

[23] Sanghamitra Bandyopadhyay and Sankar Kumar Pal. *Classification and learning using genetic algorithms: applications in bioinformatics and web intelligence*. Springer Science & Business Media, 2007 (cit. on p. 9).

[24] Y Bard. "Nonlinear parameter estimation. Academic Press, New York." In: *Nonlinear parameter estimation. Academic Press, New York.* (1974) (cit. on p. 10).

[25] Ll G Chambers. "Practical methods of optimization (2nd edn), by R. Fletcher. Pp. 436.£ 34.95. 2000. ISBN 0 471 49463 1 (Wiley)." In: *The Mathematical Gazette* 85.504 (2001), pp. 562–563 (cit. on p. 10).

[26] Ahmed Tchvagha Zeine. "Contribution à l'optimisation multi-objectifs sous contraintes: applications à la mécanique des structures". PhD thesis. Normandie, 2018 (cit. on pp. 10, 37).

[27] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. "Optimization by simulated annealing". In: *science* 220.4598 (1983), pp. 671–680 (cit. on pp. 11, 64).

[28] Engku Muhammad Nazri and Stanley Murairwa. "Classification of heuristic techniques for performance comparisons". In: *2016 12th International Conference on Mathematics, Statistics, and Their Applications (ICMSA)*. IEEE. 2016, pp. 19–24 (cit. on p. 11).

[29] Enrique Alba. *Parallel metaheuristics: a new class of algorithms*. Vol. 47. John Wiley & Sons, 2005 (cit. on pp. 12, 13, 17, 18).

[30] El-Ghazali Talbi. *Metaheuristics: from design to implementation*. Vol. 74. John Wiley & Sons, 2009 (cit. on pp. 14, 17, 18).

[31] Seval Capraz, Halil Azyikmis, and Adnan Ozsoy. "An optimized GPU-accelerated route planning of multi-UAV systems using simulated annealing". In: *International Journal of Machine Learning and Computing* 10.3 (2020) (cit. on p. 16).

[32] Fred Glover. "Future paths for integer programming and links to artificial intelligence". In: *Computers & operations research* 13.5 (1986), pp. 533–549 (cit. on p. 15).

[33] Michel Gendreau, Jean-Yves Potvin, et al. *Handbook of metaheuristics*. Vol. 2. Springer, 2010 (cit. on p. 16).

[34] Gustavo R Zavala et al. "A survey of multi-objective metaheuristics applied to structural optimization". In: *Structural and Multidisciplinary Optimization* 49.4 (2014), pp. 537–558 (cit. on p. 16).

[35] Christian Blum and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308 (cit. on pp. 17, 18).

[36] Alexander G Nikolaev and Sheldon H Jacobson. "Simulated annealing". In: *Handbook of metaheuristics*. Springer, 2010, pp. 1–39 (cit. on p. 20).

[37] Lester Ingber. "Simulated annealing: Practice versus theory". In: *Mathematical and computer modelling* 18.11 (1993), pp. 29–57 (cit. on p. 20).

[38] Richard W Eglese. "Simulated annealing: a tool for operational research". In: *European journal of operational research* 46.3 (1990), pp. 271–281 (cit. on p. 21).

[39] Mark Fleischer. "Simulated annealing: past, present, and future". In: *Winter Simulation Conference Proceedings, 1995.* IEEE. 1995, pp. 155–161 (cit. on p. 21).

[40] Darrall Henderson, Sheldon H Jacobson, and Alan W Johnson. "The theory and practice of simulated annealing". In: *Handbook of metaheuristics*. Springer, 2003, pp. 287–319 (cit. on p. 21).

[41] Christos Koulamas, SR Antony, and R Jaen. "A survey of simulated annealing applications to operations research problems". In: *Omega* 22.1 (1994), pp. 41–56 (cit. on p. 21).

[42] Fabio Romeo and Alberto Sangiovanni-Vincentelli. "A theoretical framework for simulated annealing". In: *Algorithmica* 6.1 (1991), pp. 302–345 (cit. on p. 21).

[43] Balram Suman and Prabhat Kumar. "A survey of simulated annealing as a tool for single and multiobjective optimization". In: *Journal of the operational research society* 57.10 (2006), pp. 1143–1160 (cit. on pp. 21, 23).

[44] Emile Aarts and Jan Korst. *Simulated annealing and Boltzmann machines: a stochastic approach to combinatorial optimization and neural computing*. John Wiley & Sons, Inc., 1989 (cit. on p. 21).

[45] Petrus Joseph Maria VanLaarhoven. *Theoretical and Computational Aspects of Simulated Annealing:(Theoretische en Computationele Aspekten Van Gesimuleerde Tempering)*. 1988 (cit. on p. 21).

[46] Nicholas Metropolis et al. "Equation of state calculations by fast computing machines". In: *The journal of chemical physics* 21.6 (1953), pp. 1087–1092 (cit. on p. 22).

[47] Vincent Granville, Mirko Krivánek, and J-P Rasson. "Simulated annealing: A proof of convergence". In: *IEEE transactions on pattern analysis and machine intelligence* 16.6 (1994), pp. 652–656 (cit. on p. 23).

[48] P Serafini. *Simulated annealing for multiple objective optimization problems, in volume 1 of the Proceedings of the Tenth International Conference on Multiple Criteria Decision Making*. 1992 (cit. on p. 24).

[49] LE Ulungu, J Teghem, and C Ost. "Interactive simulated annealing in a multiobjective framework: application to an industrial problem". In: *J. Oper. Res. Soc* 49 (1998), pp. 1044–1050 (cit. on p. 24).

[50] Ekunda Lukata Ulungu et al. "MOSA method: a tool for solving multiobjective combinatorial optimization problems". In: *Journal of multicriteria decision analysis* 8.4 (1999), p. 221 (cit. on p. 24).

[51] A Suppapitnarm et al. "A simulated annealing algorithm for multiobjective optimization". In: *Engineering optimization* 33.1 (2000), pp. 59–85 (cit. on pp. 24, 93).

[52] P Czyzak, M Hapke, and A Jaszkiewicz. *Application of the Pareto-simulated annealing to the multiple criteria shortest path problem*. Tech. rep. Technical Report, Politechnika Poznanska Instytut Informatyki, Poland, 1994 (cit. on p. 25).

[53] Piotr Czyżżak and Adrezej Jaszkiewicz. "Pareto simulated annealing—a metaheuristic technique for multiple-objective combinatorial optimization". In: *Journal of multi-criteria decision analysis* 7.1 (1998), pp. 34–47 (cit. on p. 25).

[54] Balram Suman. "Multiobjective simulated annealing-a metaheuristic technique for multiobjective optimization of a constrained problem". In: *Foundations of Computing and Decision Sciences* 27.3 (2002), pp. 171–191 (cit. on p. 25).

[55] Ozan Tekinalp and Gizem Karsli. "A new multiobjective simulated annealing algorithm". In: *Journal of Global Optimization* 39.1 (2007), pp. 49–77 (cit. on p. 25).

[56] Mario Villalobos-Arias, Carlos A Coello Coello, and Onésimo Hernández-Lerma. "Asymptotic convergence of a simulated annealing algorithm for multiobjective optimization problems". In: *Mathematical Methods of Operations Research* 64.2 (2006), pp. 353–362 (cit. on p. 25).

[57] James E Baker et al. "Reducing bias and inefficiency in the selection algorithm". In: *Proceedings of the second international conference on genetic algorithms*. Vol. 206. 1987, pp. 14–21 (cit. on p. 27).

[58] Peter JB Hancock. "An empirical comparison of selection methods in evolutionary algorithms". In: *AISB workshop on evolutionary computing*. Springer. 1994, pp. 80–94 (cit. on p. 28).

[59] J David Schaffer. "Multiple objective optimization with vector evaluated genetic algorithms". In: *Proceedings of the first international conference on genetic algorithms and their applications, 1985*. Lawrence Erlbaum Associates. Inc., Publishers. 1985 (cit. on pp. 30, 38).

[60] DE Goldberg. "Genetic algorithms in search, optimization, and machine learning, addison-wesley, reading, ma, 1989". In: *NN Schraudolph and J* 3.1 (1989) (cit. on p. 30).

[61] Carlos M Fonseca, Peter J Fleming, et al. "Genetic Algorithms for Multiobjective Optimization: FormulationDiscussion and Generalization." In: *Icga*. Vol. 93. July. Citeseer. 1993, pp. 416–423 (cit. on p. 30).

[62] Jeffrey Horn, Nicholas Nafpliotis, and David E Goldberg. "A niched Pareto genetic algorithm for multiobjective optimization". In: *Proceedings of the first IEEE conference on evolutionary computation. IEEE world congress on computational intelligence*. Ieee. 1994, pp. 82–87 (cit. on pp. 30, 39).

[63] Nidamarthi Srinivas and Kalyanmoy Deb. "Muiltiobjective optimization using nondominated sorting in genetic algorithms". In: *Evolutionary computation* 2.3 (1994), pp. 221–248 (cit. on pp. 30, 39).

[64] Eckart Zitzler and Lothar Thiele. "Multiobjective evolutionary algorithms: a comparative case study and the strength Pareto approach". In: *IEEE transactions on Evolutionary Computation* 3.4 (1999), pp. 257–271 (cit. on p. 30).

[65] Carlos A Coello Coello. "A comprehensive survey of evolutionary-based multiobjective optimization techniques". In: *Knowledge and Information systems* 1.3 (1999), pp. 269–308 (cit. on p. 30).

[66] Abdullah Konak, David W Coit, and Alice E Smith. "Multi-objective optimization using genetic algorithms: A tutorial". In: *Reliability engineering & system safety* 91.9 (2006), pp. 992–1007 (cit. on p. 30).

[67] R Timothy Marler and Jasbir S Arora. "Survey of multi-objective optimization methods for engineering". In: *Structural and multidisciplinary optimization* 26.6 (2004), pp. 369–395 (cit. on p. 30).

[68] Carlos M Fonseca and Peter J Fleming. "An overview of evolutionary algorithms in multiobjective optimization". In: *Evolutionary computation* 3.1 (1995), pp. 1–16 (cit. on p. 30).

[69] Vilfredo Pareto. *Cours d'économie politique*. Vol. 1. Librairie Droz, 1964 (cit. on p. 33).

[70] Kalyanmoy Deb et al. "Scalable multi-objective optimization test problems". In: *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No. 02TH8600)*. Vol. 1. IEEE. 2002, pp. 825–830 (cit. on p. 33).

[71] Kalyanmoy Deb. "Introduction to evolutionary multiobjective optimization". In: *Multiobjective Optimization*. Springer, 2008, pp. 59–96 (cit. on pp. 33, 38).

[72] Aharon Ben-Tal. "Characterization of Pareto and lexicographic optimal solutions". In: *Multiple Criteria Decision Making Theory and Application: Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20–24, 1979*. Springer. 1980, pp. 1–11 (cit. on p. 34).

[73] C-L Hwang and Abu Syed Md Masud. *Multiple objective decision making—methods and applications: a state-of-the-art survey*. Vol. 164. Springer Science & Business Media, 2012 (cit. on p. 36).

[74] Johan Andersson. "A survey of multiobjective optimization in engineering design". In: *Department of Mechanical Engineering, Linktjping University. Sweden* (2000) (cit. on p. 38).

[75] Aimin Zhou et al. "Multiobjective evolutionary algorithms: A survey of the state of the art". In: *Swarm and evolutionary computation* 1.1 (2011), pp. 32–49 (cit. on p. 38).

[76] David A Van Veldhuizen and Gary B Lamont. *Multiobjective evolutionary algorithm research: A history and analysis*. Tech. rep. Citeseer, 1998 (cit. on pp. 38, 46).

[77] Carlos M Fonseca and Peter J Fleming. "Multiobjective optimization and multiple constraint handling with evolutionary algorithms. II. Application example". In: *IEEE Transactions on systems, man, and cybernetics-Part A: Systems and humans* 28.1 (1998), pp. 38–47 (cit. on p. 39).

[78] Kalyanmoy Deb et al. "A fast and elitist multiobjective genetic algorithm: NSGA-II". In: *IEEE transactions on evolutionary computation* 6.2 (2002), pp. 182–197 (cit. on pp. 39, 103).

[79] Kalyanmoy Deb and Himanshu 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* 18.4 (2013), pp. 577–601 (cit. on p. 39).

[80] Eckart Zitzler and Lothar Thiele. "Multiobjective optimization using evolutionary algorithms—a comparative case study". In: *International conference on parallel problem solving from nature*. Springer. 1998, pp. 292–301 (cit. on pp. 39, 45).

[81] E Zitzler, M Laumanns, and L Thiele. "SPEA2: Improving the Strength Pareto Evolutionary Algorithm.–Evolutionary Methods for Design". In: *Optimization and Control with Applications to Industrial Problems, Athens, Greece* (2002), pp. 95–100 (cit. on pp. 39, 41).

[82] Qingfu Zhang, Wudong Liu, and Hui Li. "The performance of a new version of MOEA/D on CEC09 unconstrained MOP test instances". In:

2009 IEEE congress on evolutionary computation. IEEE. 2009, pp. 203–208 (cit. on pp. 39, 42).

[83] Joshua D Knowles and David W Corne. "Approximating the nondominated front using the Pareto archived evolution strategy". In: *Evolutionary computation* 8.2 (2000), pp. 149–172 (cit. on p. 42).

[84] Alejandro Rosales Pérez. "Surrogate-assisted evolutionary multi-objective full model selection". PhD thesis. Instituto Nacional de Astrofísica, Óptica y Electrónica, 2016 (cit. on p. 42).

[85] Kalyanmoy Deb. "Multi-objective Optimization Using Evolutionary Algorithms". In: (2001) (cit. on p. 44).

[86] Charles Audet et al. "Performance indicators in multiobjective optimization". In: *European journal of operational research* 292.2 (2021), pp. 397–422 (cit. on p. 44).

[87] Siwei Jiang et al. "Consistencies and contradictions of performance metrics in multiobjective optimization". In: *IEEE transactions on cybernetics* 44.12 (2014), pp. 2391–2404 (cit. on p. 45).

[88] Tatsuya Okabe, Yaochu Jin, and Bernhard Sendhoff. "A critical survey of performance indices for multi-objective optimisation". In: *The 2003 Congress on Evolutionary Computation, 2003. CEC'03.* Vol. 2. IEEE. 2003, pp. 878–885 (cit. on p. 45).

[89] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. "Performance metrics in multi-objective optimization". In: *2015 Latin American computing conference (CLEI)*. IEEE. 2015, pp. 1–11 (cit. on p. 45).

[90] Jason Ramon Schott. "Fault tolerant design using single and multicriteria genetic algorithm optimization". PhD thesis. Massachusetts Institute of Technology, 1995 (cit. on p. 47).

[91] Jonathan L Gross and Jay Yellen. *Handbook of graph theory*. CRC press, 2003 (cit. on p. 50).

[92] Claude Berge. "Graphs: North Holland mathematical library". In: (1985) (cit. on p. 50).

[93] B Bollobás. "Modern Graph Theory Springer". In: *New York* (1998) (cit. on p. 50).

[94] Douglas Brent West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001 (cit. on p. 50).

[95] TW Haynes, ST Hedetniemi, and PJ Slater. "Fundamentals of domination in graphs, ser". In: *Monographs and Textbooks in Pure and Applied Mathematics. New York: Marcel Dekker Inc* 208 (1998) (cit. on p. 54).

[96] Ran Raz and Shmuel Safra. "A sub-constant error-probability low-degree test, and a sub-constant error-probability PCP characterization of NP". In: *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*. 1997, pp. 475–484 (cit. on p. 54).

[97] Michael R Garey and David S Johnson. *Computers and intractability*. Vol. 174. freeman San Francisco, 1979 (cit. on p. 54).

[98] Mike Morgan and Vic Grout. "Metaheuristics for wireless network optimisation". In: *The Third Advanced International Conference on Telecommunications (AICT'07)*. IEEE. 2007, pp. 15–15 (cit. on p. 59).

[99] Raka Jovanovic and Milan Tuba. "Ant colony optimization algorithm with pheromone correction strategy for the minimum connected dominating set problem". In: *Computer Science and Information Systems* 10.1 (2013), pp. 133–149 (cit. on p. 59).

[100] Ruizhi Li et al. "GRASP for connected dominating set problems". In: *Neural Computing and Applications* 28.1 (2017), pp. 1059–1067 (cit. on p. 59).

[101] Xinyun Wu, Zhipeng Lü, and Philippe Galinier. "Restricted swap-based neighborhood search for the minimum connected dominating set problem". In: *Networks* 69.2 (2017), pp. 222–236 (cit. on p. 59).

[102] Abdel-Rahman Hedar et al. "Two meta-heuristics designed to solve the minimum connected dominating set problem for wireless networks design and management". In: *Journal of Network and Systems Management* 27.3 (2019), pp. 647–687 (cit. on p. 59).

[103] Christoph Ambühl et al. "Constant-factor approximation for minimum-weight (connected) dominating sets in unit disk graphs". In: *International Workshop on Approximation Algorithms for Combinatorial Optimization*. Springer. 2006, pp. 3–14 (cit. on p. 59).

[104] Zuleyha Akusta Dagdeviren, Dogan Aydin, and Muhammed Cinsdikici. "Two population-based optimization algorithms for minimum weight connected dominating set problem". In: *Applied Soft Computing* 59 (2017), pp. 644–658 (cit. on pp. 59, 96, 102).

[105] Salim Bouamama, Christian Blum, and Jean-Guillaume Fages. "An algorithm based on ant colony optimization for the minimum connected dominating set problem". In: *Applied Soft Computing* 80 (2019), pp. 672–686 (cit. on p. 59).

[106] Ruizhi Li et al. "A restart local search algorithm with Tabu method for the minimum weighted connected dominating set problem". In: *Journal of the Operational Research Society* 73.9 (2022), pp. 2090–2103 (cit. on p. 59).

[107] Ruizhi Li et al. "A Novel Local Search Algorithm Based on Connectivity Method and Vertex Penalty Mechanism for Minimum Weighted Connected Dominating Set Problem". In: *Available at SSRN 4714937* () (cit. on p. 60).

[108] Zuleyha Akusta Dagdeviren, Dogan Aydin, and Muhammed Cinsdikici. "Two population-based optimization algorithms for minimum weight connected dominating set problem". In: *Applied Soft Computing* 59 (2017), pp. 644–658 (cit. on p. 80).

[109] F Romeo, Vincentelli Ak Sangiovanni, and Md Huang. "An efficient general cooling schedule for simulated annealing". In: PROCEEDING OF IEEE INTERNATIONAL CONFERENCE ON COMPUTER AIDED DESIGN. 1986 (cit. on p. 94).