Models and Algorithms for Depot-Free Multiple Traveling Salesperson Problems

By:

José Alejandro Cornejo-Acosta

A dissertation submitted in partial fulfillment
of the requirements for the degree of:

Ph.D. IN COMPUTER SCIENCE

at

Instituto Nacional de Astrofísica, Óptica y Electrónica

November, 2024
Tonantzintla, Puebla, México

Supervised by:

Dr. Jesús García-Díaz
Dr. Julio César Pérez-Sansalvador

Abstract

This thesis addresses different aspects of Depot-Free Multiple Traveling Salesperson Problems (DFmTSPs), which generalize the Multiple Traveling Salesperson Problem (mTSP). Unlike traditional mTSPs, DFmTSPs do not require the depot concept. Thus, they pose unique challenges that need to be addressed. This research primarily focuses on developing and analyzing mathematical models and algorithms for DFmTSPs.

This document introduces innovative and novel integer programs that depend on the dummy depot concept. These integer programs show theoretical and practical advantages since they are general enough to be useful for the main variants of DFmTSPs.

Additionally, a two-phase constructive heuristic and a memetic algorithm are proposed to tackle *large* graph instances where exact methods are impractical. For the two-phase constructive heuristic, the cluster-first route-second technique is used. In the clustering phase, a heuristic for the Capacitated Vertex k-Center Problem is proposed. This heuristic is based on a relationship with the Minimum Capacitated Dominating Set. Afterward, for the routing phase, the farthest-first heuristic and Lin-Kernighan heuristic were used, the latter is one of the best-known heuristics for the classical Traveling Salesperson Problem (TSP) in the state-of-the-art.

For the memetic algorithm, a deep study of the diversity of solutions is performed. Thus, diversity management techniques and novel genetic components are designed and analyzed. The performance of these algorithms is rigorously tested on various benchmark instances, demonstrating significant improvements in the quality of the returned solutions. The results underline the practical viability of the proposed models and algorithms, paving the way for future research in related optimization problems.

Resumen

Esta tesis aborda diferentes aspectos de los Problemas de Múltiples Agentes Viajeros sin almacenes (DFmTSPs, por sus siglas en inglés), los cuales generalizan el Problema de Múltiples Agentes Viajeros (mTSP). A diferencia del mTSP tradicional, los DFmTSPs no consideran el concepto de almacén. Por lo tanto, presentan retos únicos y específicos que deben ser considerados para su resolución. Esta investigación se centra en el desarrollo y análisis de modelos matemáticos y algoritmos para los DFmTSPs.

Esta investigación propone formulaciones de programación entera que son innovadoras, novedosas, y que explotan el concepto de almacenes ficticios. Estas formulaciones ofrecen ventajas teóricas y prácticas, ya que son lo suficientemente generales para abordar las principales variantes de los DFmTSPs.

Además, se proponen una heurística constructiva de dos fases y un algoritmo memético para abordar instancias más grandes del estado del arte, en donde los métodos exactos resultan ser poco prácticos. Para la heurística constructiva de dos fases, se utiliza la técnica de cluster-first route-second (agrupar primero, enrutar después). En la fase de agrupamiento, se propone una heurística para el Problema de Localización de k Instalaciones Capacitadas (Capacitated Vertex k-center Problem), que se basa en una relación con el Conjunto Dominante Mínimo con Capacidades (Minimum Capacitated Dominating Set). Posteriormente, para la fase de enrutamiento se emplea la heurísticas farthest-first y la heurística Lin-Kernighan, la cual es considerada una de las mejores en el estado del arte para el problema clásico del Agente Viajero (TSP, por sus siglas en inglés).

En cuanto al algoritmo memético, se realiza un estudio a profundidad de la diversidad de las soluciones. Para ello, se diseñan y analizan técnicas de gestión de diversidad explícita y operadores genéticos novedosos. El rendimiento de estos algoritmos es probado rigurosamente en diversas instancias del estado del arte, mostrando mejoras significativas en la calidad de las soluciones encontradas. Los resultados demuestran la viabilidad práctica de los modelos y de los algoritmos propuestos, además de sentar las bases para futuras investigaciones en problemas de optimización relacionados.

Acknowledgments

I acknowledge Gurobi for providing a free-of-charge license for the Gurobi software, which was used to implement the integer programs presented in this document. I also want to acknowledge Consejo Nacional de Humanidades, Ciencias y Tecnologías (CONAHCYT), Instituto Nacional de Astrofísica, Óptica y Electrónica (INAOE), Universidad Autónoma de Nuevo León (UANL), Dr. Roger Z. Rios-Mercado, Instituto Tecnológico y de Estudios Superiores de Monterrey (ITESM), Dra. Yasmín Á. Ríos Solis, Centro de Investigación en Matemáticas (CIMAT), Dr. Carlos Segura, and Instituto Tecnológico Superior de Purísima del Rincón, for providing valuable advice to complete this research. In particular, I want to acknowledge my advisors Jesús García-Díaz and Julio César Pérez-Sansalvador, for their patience and support over all of these years. Thank you for giving me the opportunity to collaborate with you. I learned and had a lot of fun with you in the last years. I will always be grateful to you and will always carry you in my heart. I also want to thank my father and mother for supporting my education process in all of my life. Thank you to my brothers for your advice on personal facts. Finally, a special mention to Blanca Zuñiga, thank you for your love and support, I hope our love prevails forever.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Understanding and solving real-world problems is one of the ultimate goals of computer science. Among many others, mathematical modeling, discrete mathematics, graph theory, mathematical programming, and algorithm design are the main scientific tools for achieving such goals. This document approaches a fundamental problem from the aforementioned research fields: a relatively new variant of the classical $\mathcal{NP}$-hard Multiple Traveling Salesperson Problem (mTSP), the Depot-Free mTSP (DFmTSP). Along with the popular Traveling Salesperson Problem (TSP), the mTSP and DFmTSP are at the core of real-world logistics problems.

From complexity theory, it is well-established that many real-world problems are difficult to solve. In detail, they belong to complexity categories like $\mathcal{NP}$-hard. Thus, there are no efficient algorithms for solving them unless $\mathsf{P} = \mathsf{NP}$ [2]. Although the P vs NP question remains open, namely, we do not know whether $\mathsf{P} = \mathsf{NP}$ or $\mathsf{P} \neq \mathsf{NP}$, the scientific community is vastly inclined to the $\mathsf{P} \neq \mathsf{NP}$ conjecture [3]. Thus, it is doubtful that $\mathcal{NP}$-hard real-world problems can be solved efficiently. Of course, this includes the DFmTSP.

In simple and general terms, the mTSP consists of finding an optimal collection of routes followed by a set of salespersons. This collection of routes is usually constrained by the presence of depots, i.e., special facilities from where salespersons depart and arrive. Since not all real-world logistics problems require the depot concept, it is important to model and solve the DFmTSP and its main variants, which include load-balance (a.k.a. bounding constraints) and different objective functions.

Given the difficulty of DFmTSP, different algorithmic approaches can be used depending on the desired goals. For instance, if optimal solutions are required and the problem instance is relatively small, generic exact algorithms or black-box optimization solvers can be used. If near-optimal solutions are good enough and the instance is relatively large, then heuristics and metaheuristics have proven to be better options.

Like many other combinatorial optimization problems, DFmTSP is relatively simple

to state but difficult to model and solve. It belongs to the $\mathcal{NP}$-hard class, and for the first 2.5 years of this thesis' development, no mathematical model was available in the literature. Originally, one of this thesis' objectives was to present the first mathematical model of the problem. However, researchers from Turkey got ahead of us in May 2021 [4]. Nevertheless, Chapter 4 introduces novel models that are more general and proved advantageous under certain circumstances. Besides, Chapter 5 introduces new heuristics and metaheuristics with clear advantages over the state-of-the-art.

1.1 Problem statement

The DFmTSP receives a complete weighted graph $G = (V, E, w)$ and three positive integers $m, L, U \in \mathbb{Z}^+$ as input. Its goal is to find a set of m disjoint *paths* (See Equations (1.1) and (1.2)) with the following constraints: the set of paths must cover all vertices in G (Equation (1.3)) and must be load-balanced (Equation (1.4)). Finally, there are two classical objective functions to consider. The first one is the *minsum* function, which aims to minimize the sum of the cost of the salespersons' paths (Equation (1.5)). The second one is the *minmax* function, which consists of minimizing the longest path among the salespersons (Equation (1.6)).

$$\text{find} \quad P = \{p_1, p_2, ..., p_m\} \tag{1.1}$$

$$\text{subject to (s.t.)} \quad V(p_i) \cap V(p_j) = \emptyset \qquad \forall i, j \in [1, m] \text{ and } i \neq j \tag{1.2}$$

$$\sum_{i=1}^{m} |V(p_i)| = |V(G)| \tag{1.3}$$

$$L \leqslant |V(p_i)| \leqslant U \qquad \forall i \in [1, m] \tag{1.4}$$

Common objective functions:

$$\min \quad \sum_{i=1}^{m} c(p_i) \qquad \textit{minsum function} \tag{1.5}$$

$$\min \quad \max\{c(p_1), c(p_2), \cdots, c(p_m)\} \qquad \textit{minmax function} \tag{1.6}$$

1.2 Hypothesis

- Using dummy vertices, it is possible to model Depot-Free Multiple Traveling Salesperson Problems with bounding constraints (DFmTSPs) by exploiting its relationship with other $\mathcal{NP}$-hard problems such as the Traveling Salesperson

Problem (TSP) and the Fixed-Destination Multiple-Depots Multiple Traveling Salesperson Problem (FD-MmTSP).

- Optimal solutions for relatively small instances of the problems can be obtained based on the aforementioned models.
- Feasible and near-optimal solutions for larger instances can be obtained using heuristic-search approaches such as a clustering-routing strategy and evolutionary computing algorithms.

1.3 General objective

To propose mathematical models and to design and characterize algorithms to get feasible (optimal or near-optimal) solutions for Depot-Free Multiple Traveling Salesperson Problems (DFmTSP) with bounding constraints aiming at load balance.

1.4 Particular objectives

1. To explore and implement different state-of-the-art problem-solving techniques for DFmTSP problems with bounding constraints.
2. To propose mathematical models and integer programs for DFmTSP problems with bounding constraints.
3. To analyze and compare the proposed mathematical models against the state-of-the-art models for DFmTSP problems.
4. To design and characterize a two-phase constructive heuristic that uses the clustering-routing strategy and evolutionary computing algorithms for DFmTSP problems with bounding constraints.
5. To compare the proposed heuristic-search algorithms against state-of-the-art algorithms for DFmTSP problems by using the classic objective functions from the literature.

1.5 Methodology

The methodology used in this research project consists of studying the problem from two different perspectives: mathematical modeling and heuristics. Within each of these

big boxes, there is a series of tasks that is necessary to achieve the stated objectives and validate the proposed hypothesis.

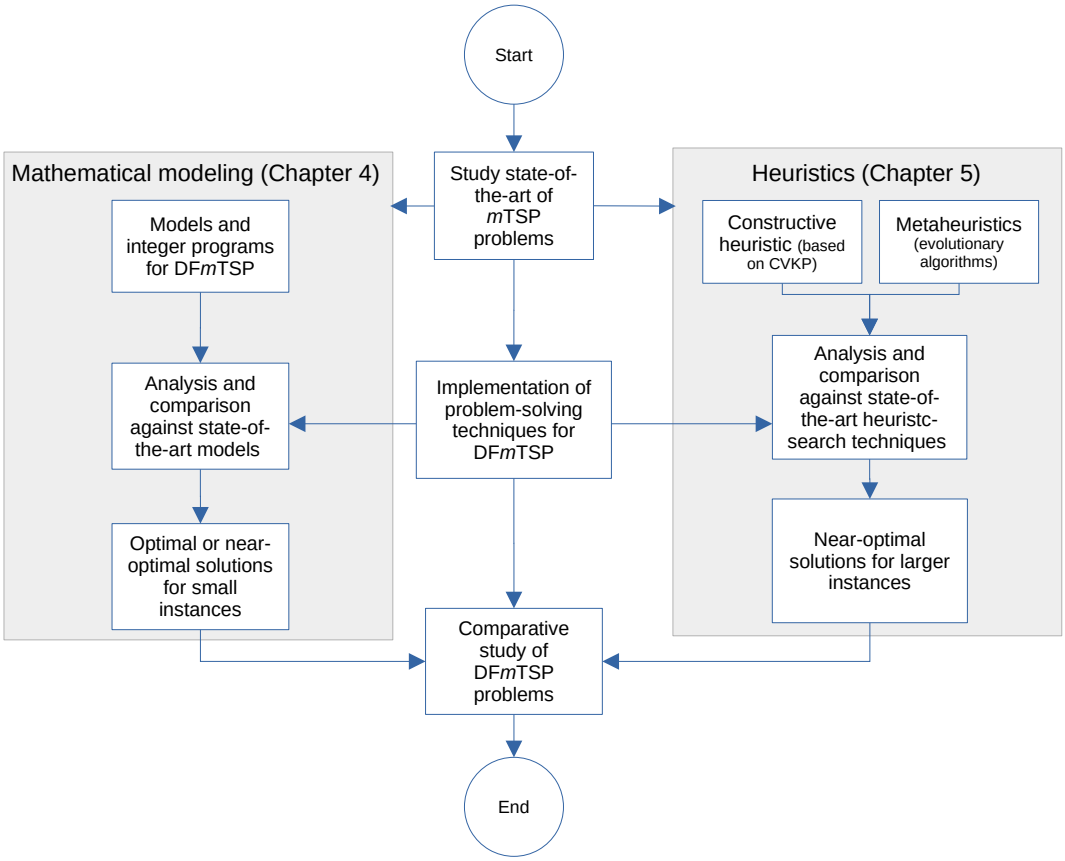


Figure 1.1: Followed methodology.

1.6 Contributions

Some of the contributions of this thesis have been already published in peer-reviewed journals:

- A series of **mathematical models** for the DFmTSP are proposed. They consider open and closed paths, load balance, and a hybrid case that considers the presence of a few depots. These results are related to **particular objectives 2 and 3**.
  - Cornejo-Acosta, J.A.; García-Díaz, J.; Pérez-Sansalvador, J.C.; Segura, C. Compact Integer Programs for Depot-Free Multiple Traveling Salesper-

son Problems. Mathematics 2023, 11, 3014. https://doi.org/10.3390/math11133014

- A **polynomial-time reduction** was found from the Capacitated Vertex k-center Problem (CVKCP) to the Minimum Capacitated Dominating Set Problem (CMDSP). This result relates to **objectives 4 and 5** (See the next bullet).
  - Cornejo Acosta, J.A.; García Díaz, J.; Menchaca-Méndez, R.; Menchaca-Méndez, R. Solving the Capacitated Vertex K-Center Problem through the Minimum Capacitated Dominating Set Problem. Mathematics 2020, 8, 1551. https://doi.org/10.3390/math8091551
- A **parallel constructive heuristic** for CVKCP was introduced. This result is related to **particular objectives 4 and 5** and follows from the results of the previous bullet.
  - Cornejo Acosta, J.A.; García Díaz, J; Pérez Sansalvador J.C.; Rios Mercado, R.Z.; Pomares Hernández, S.E. A Constructive Heuristic for the Uniform Capacitated Vertex k-center Problem. ACM Journal of Experimental Algorithmics 2023, 28. https://doi.org/10.1145/3604911
- An **update policy** called kFLS for cellular evolutionary algorithms is proposed. It is related to **particular objectives 4 and 5** and is used for an in-depth study of management diversity techniques of evolutionary algorithms for the DFmTSP.
  - J. A. Cornejo-Acosta and J. García-Díaz, "A First Approach to Asynchronous-Synchronous Tradeoff in 1D Cellular Genetic Algorithms," Research in Computing Science, vol. 150, no. 12, 2021
- A series of metaheuristic algorithms for DFmTSP. With a particular focus on a **memetic evolutionary algorithm with diversity management**. This paper is still in progress and soon to be submitted. These results are related to **particular objectives 4 and 5**.

Preliminary versions of some of the previous papers have been presented at the following conferences: 53 Congreso Nacional de la Sociedad Matemática Mexicana 2020, IX Congreso Nacional de la Sociedad Mexicana de Investigación de Operaciones 2021, Congreso Internacional CORE 2021, Escuela Latinoamericana de Verano en Investigación Operativa 2022, XI Congreso de la Sociedad Mexicana de Investigación de Operaciones 2023, and Congreso Internacional CORE 2024.

1.7 Thesis organization

The remaining part of this thesis is organized as follows: Chapter 2 presents the necessary background. It introduces fundamental concepts of the area, such as graph theory and optimization concepts and techniques. It also introduces combinatorial optimization problems, specifically location and routing problems. These concepts are widely used throughout the thesis. Chapter 3 is about the related work and the state-of-the-art of Multiple Traveling Salesperson Problems (mTSPs). It shows a chronological history of mTSPs and how they have been approached in the literature, including mathematical models, exact algorithms, and heuristic approaches, including clustering-routing. Chapter 4 introduces one of the main contributions of this thesis, novel mathematical models. These consider the main variants of the DFmTSP, including closed paths, open paths, bounding constraints, and a mix between DFmTSP and MmTSP. While the previous chapter focuses on modeling and solving relatively small instances of the problem, Chapter 5 focuses on novel heuristic and metaheuristic approaches. It is divided into two main sections. Section 5.1 introduces a clustering-routing heuristic for the DFmTSP that is based on the Capacitated Vertex k-center Problem (CVKCP). Section 5.2 introduces a metaheuristic algorithm with an explicit diversity management technique. Both sections are accompanied by a subsection of analysis and computational results that show their advantages. Finally, Chapter 6 states this thesis's conclusions, final remarks, and future research.

Chapter 2

Background

Although mTSPs have been studied from many different perspectives and research fields, this chapter focuses on the following:

- Graph theory.
  - Gives us the basic definitions, vocabulary, and notation used throughout this document.
- Mathematical programming.
  - Gives us basic definitions for mathematical modeling and algorithm design.
- Operations research.
  - Gives us a background for real-world and canonical location and routing problems.
- Heuristics and metaheuristics.
  - Gives us a background for the algorithms introduced in this document.

Since DFmTSP can be easily stated in terms of graphs, some graph theory concepts are listed. Then, some foundations of mathematical programming, heuristics, metaheuristics, and operations research are presented. It is worth mentioning that all these research areas are strongly intertwined. Therefore, from an appropriate point of view, any of them can be considered a subfield of the others. Nevertheless, this document tries to present each of them as independently as possible from the others.

2.1 Graph theory concepts

In 1736, there was a popular game among the citizens of Königsberg (today's Kaliningrad, Russia): *the seven Königsberg bridges*. The question was if all parts of the city

could be visited traversing all bridges once [5]. Although this problem can be solved by brute force, the answer being negative, it was Leonhard Euler who gave an explanation for the answer. With this explanation, Euler laid the foundations of graph theory.

Graphs are a popular mathematical tool for modeling and visualizing data. Given their relative simplicity and flexibility, they have been used in many applications, including chemistry, bioinformatics, social networks, linguistics, computer vision, image classification, and logistics problems [6]. Thus, knowing the main definitions of graph theory gives us a basic vocabulary and mathematical notation for modeling and tackling different kinds of problems.

Through the years, graph theory has grown, and today is a very extensive field; however, only the following concepts and terms are necessary for the context of this thesis. These concepts were obtained mainly from [7] and [8].

Definition 2.1.1. *Let* $[A]^k$ *be the set of all* k*-element subsets of a given set* A*.*

Definition 2.1.2. *A graph is an ordered pair* $G = (V, E)$*, where* V *is a set of vertices and* $E \subseteq [V]^2$*; namely, the elements of* E *are pairs of the elements in* V *and are called edges. The vertices of* V *are usually labeled as* $\{v_1, v_2, \ldots, v_n\}$*, where* $n = |V|$*.*

Definition 2.1.3. *The set of vertices and edges of a given graph* H *can be denoted by* $V(H)$ *and* $E(H)$*, respectively.*

Notice that a graph is usually referred to in the literature as an undirected simple graph to emphasize its main characteristics: the edges are undirected and unweighted.

Definition 2.1.4. *The order of a graph is its number of vertices. It can be denoted by* $|V(G)|$ *or* $|G|$*.*

Definition 2.1.5. *The size of a graph is its number of edges. It can be denoted by* $|E(G)|$ *or* $\|G\|$*.*

Definition 2.1.6. *A graph* $G = (V, E)$ *is a complete graph if* $E = [V]^2$*, i.e., for all pairs of vertices* $v_i, v_j \in V$*, exists an edge* $\{v_i, v_j\} \in E$*.*

Definition 2.1.7. *A directed graph* $G = (V, E)$ *is an ordered pair consisting of a set of vertices* V *and a set* E *of 2-tuples of* V*. So, an edge* (u, v) *goes from* u *to* v*.*

Definition 2.1.8. *A weighted graph* $G = (V, E, w)$ *is an ordered tuple consisting of a set of vertices* V*, a set of edges* E*, and a function* $w : E \to \mathbb{R}$*. That is, each edge* $e \in E$ *has an associated cost* $w(e) \in \mathbb{R}$*.*

Definition 2.1.9. *Given a graph $G = (V, E)$, a dominating set is a set $D \subseteq V$ such that for every vertex $v \in V \setminus D$, there is a vertex $u \in D$ such that $\{v, u\} \in E$.*

Definition 2.1.10. *A minimum dominating set is a set of minimum cardinality among all the dominating sets.*

Definition 2.1.11. *Given a graph $G = (V, E)$, and a capacity function $f_{cap} : V \to \mathbb{Z}^+$. A capacitated dominating set $D \subseteq V$ is a set such that every vertex $v \in V \setminus D$ is assigned to some vertex $u \in D \cap N(v)$, and the number of vertices assigned to each vertex $u \in D$ is not greater than its capacity $f_c(u)$, where $N(u)$ is the open neighborhood of $u \in V$.*

Definition 2.1.12. *A minimum capacitated dominating set is a set of minimum cardinality among all the capacitated dominating sets.*

Definition 2.1.13. *Given a weighted graph $G = (V, E, w)$, a bottleneck graph $G_r = (V, E_r)$ is such that E_r consists of all the edges in E with weight less than or equal to r, i.e., $E_r = \{e \in E : w(e) \leqslant r\}$.*

Definition 2.1.14. *A path in G is a sequence of vertices $p = (v_1, v_2, v_3, \ldots, v_k)$ is composed by edges $((v_1, v_2), (v_2, v_3), (v_{k-1}, v_k))$, such that $\forall v_i \in p$, $v_i \in V(G)$ and $\forall (v_i, v_j) \in p$, $(v_i, v_j) \in E(G)$.*

Definition 2.1.15. *The cost of a closed path p in a weighted graph $G = (V, E, w)$ is given by Equation (2.1).*

$$c(p) = w\big((p_{|p|}, p_1)\big) + \sum_{j=1}^{|p|-1} w\big((p_j, p_{j+1})\big) \tag{2.1}$$

From Equation (2.1), $|p|$ is the number of vertices in the path p. The sum represents cycling through all the vertices of the path. w is defined as $w : E \to \mathbb{R}$ and $w\big((p_j, p_{j+1})\big)$ indicates the cost between the jth vertex of the path and the $(j+1)$th vertex in the path. Finally, $w\big((p_{|p|}, p_1)\big)$ represents the cost from the last vertex of the path to the initial vertex of the path.

Definition 2.1.16. *The cost of an open path p in a weighted graph $G = (V, E, w)$ is similar to the previous definition but avoids the cost of the last vertex in the path to the first vertex of it. The cost of an open path is given by Equation (2.2).*

$$c(p) = \sum_{j=1}^{|p|-1} w\big((p_j, p_{j+1})\big) \tag{2.2}$$

Definition 2.1.17. *A path that visits all vertices of a given graph G once is known as a Hamiltonian path or, equivalently, a Hamilton path.*

Definition 2.1.18. *A closed path that visits all vertices of a given graph* G *once and only the first and last vertices are equal is known as a Hamiltonian cycle or, equivalently, a Hamilton cycle.*

Definition 2.1.19. *If a graph* G *contains a Hamilton cycle, it is known as a Hamiltonian graph.*

2.2 Mathematical programming

Mathematical programming (also known as *mathematical optimization* and *optimization theory*) deals with the modeling and solving of decision problems, which, within this branch of mathematics, are also known as *mathematical programs*. It is important to observe that the term *program* has a different meaning from the one that predominates nowadays. The terms are meant to contrast with *computer programming*, which solves such problems by implementing algorithms that may be designed specifically for a given problem [6]. In simpler terms, the word program is used as a synonym for agenda or schedule. To avoid confusion, equivalent terms for *mathematical program* are *mathematical model* and *mathematical formulation*.

Complementary to the previous paragraph, one of the pioneers of this research area defines mathematical programming as follows [9]:

> *Mathematical programming can be viewed as part of a great revolutionary development that has given mankind the ability to state general goals and lay out a path of detailed decisions to be taken in order to "best" achieve these goals when faced with practical situations of great complexity. The tools for accomplishing this are the models that formulate real-world problems in detailed mathematical terms, the algorithms that solve the models, and the software that executes the algorithms on computers based on the mathematical theory.*
>
> (George B. Dantzig, 1997)

Naturally, *optimization* is the most important concept in mathematical programming. Thus, the next section formally defines this term.

2.2.1 Optimization

In real-life scenarios, we often encounter situations where we need to make decisions to reach a goal while using the optimal amount of resources. This decision-making process could involve determining the best configuration for a task. The resources at our disposal could be time, money, effort, etc. These types of problems, where we

have a range of possible decisions and we need to select the "best" one, are known as *optimization problems*. They are not only practically relevant but also hold theoretical significance. To define an optimization problem, we first must introduce the concept of an *instance of a problem*.

Definition 2.2.1. *An instance of an optimization problem is a pair* (S, c)*, where* S *is the domain set of all possible solutions for such instance, and* c *is the cost function defined as:*

$$c : S \to \mathbb{R}$$

and we want to find an $s \in S$ *for which:*

$$c(s) \leqslant c(y), \forall y \in S \quad \textit{for a minimization problem}$$

$$c(s) \geqslant c(y), \forall y \in S \quad \textit{for a maximization problem}$$

Such object s *is called the global optimum for the given instance* (S, c)*.*

Definition 2.2.2. *An optimization problem consists of all of its instances.*

To define a few other important concepts, let us consider a simple instance of an optimization problem (S, c) defined by Eq. (2.3) with the cost function c defined by Figure 2.1.

$$S : [0, 1] \subseteq \mathbb{R} \tag{2.3}$$

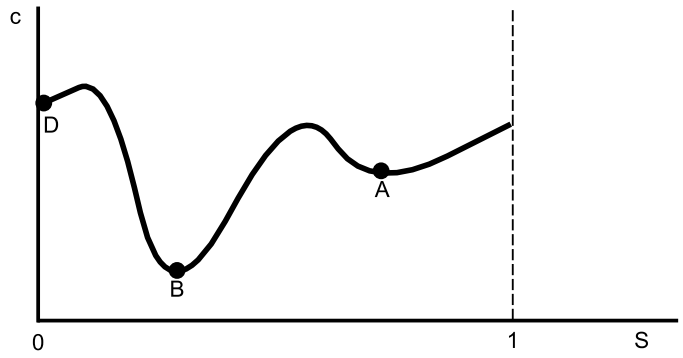


Figure 2.1: A 1-dimensional optimization problem.

Suppose the cost function is to be minimized. In that case, A, B, and D are all *local optimum* because each of them is the *minimum* in its respective neighborhood, but only B is the *global optimum* because it is the minimum over the complete domain S.

In general, the domain S can contain any kind of mathematical object. However, these are usually numbers or can be modeled by them. Thus, mathematical programming states problems using a series of algebraic expressions, including a set of objective functions, constraints, and feasible solutions. The general form of an optimization problem is [10]:

$$\begin{aligned} \max/\min \quad & f(\mathbf{x}) \\ \text{s.t.} \quad & h_i(\mathbf{x}) = 0 \quad i \in \{1, 2, \ldots, m\} \\ & g_j(\mathbf{x}) \geqslant 0 \quad j \in \{1, 2, \ldots, r\} \\ \text{where} \quad & \mathbf{x} \in S \end{aligned}$$

In this general formulation (or program), $\mathbf{x}$ is an n-dimensional vector of variables, $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$, and f, h_i, and g_j are real-valued functions of the variables $x_1, x_2, \ldots, x_n$. The set S is a subset of n-dimensional space. The function f is the problem's objective function, and the equations, inequalities, and set restrictions define the problem's constraints.

Depending on its specific attributes, a mathematical program can be classified as a Linear Program (LP), an Integer Linear Program (ILP), an Integer Nonlinear Program (INLP), a Quadratic Integer Program (QIP), a Quadratically-Contrained Integer Program (QCIP), and so on. From these categories, let us begin by showing the canonical structure of a LP:

$$\begin{aligned} \max/\min \quad & \mathbf{cx} \\ \text{s.t.} \quad & \mathbf{Ax} \leqslant \mathbf{b} \\ & \mathbf{x} \geqslant \mathbf{o} \end{aligned}$$

In this program, the input data include the row vector $\mathbf{c} = [c_1, \cdots, c_n]$, the $m \times n$ matrix $\mathbf{A} = (a_{ij})$, and the column vector $\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_n \end{bmatrix}$. The column vector $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ contains the variables to be optimized. In an LP problem, the n-vector $\mathbf{x}$ has real elements. These problems belong to the complexity class $\mathcal{P}$ [9]. Namely, they can be solved in polynomial time. It is very interesting how a *small* change to the previous program leads to a harder problem. Namely, if n-vector $\mathbf{x}$ can only take integer values $\mathbf{x} \in \mathbb{Z}^n$, this vector is said to be *integral*, and the problem becomes an Integer Linear

Program (ILP) if the set of constraints is linear too. Integer Linear Programming is usually known just as Integer Programming. A canonical Integer Program (IP) is of the following form and belongs to the $\mathcal{NP}$-hard class [11]:

$$\begin{aligned} \max/\min \quad & \mathbf{c}\mathbf{x} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} \leqslant \mathbf{b} \\ & \mathbf{x} \geqslant \mathbf{0} \qquad \text{integral} \end{aligned}$$

If some variables from vector $\mathbf{x}$ have to be integers while others can be real numbers, then the program is mixed. For instance, an ILP where some variables can be real numbers is known as a Mixed Integer Linear Program (MILP) and has the following canonical form.

$$\begin{aligned} \max/\min \quad & \mathbf{c}\mathbf{x} + \mathbf{h}\mathbf{y} \\ \text{s.t.} \quad & \mathbf{A}\mathbf{x} + \mathbf{G}\mathbf{x} \leqslant \mathbf{b} \\ & \mathbf{x} \geqslant \mathbf{0} \qquad \text{integral} \\ & \mathbf{y} \geqslant \mathbf{0} \end{aligned}$$

Although many other types of programs exist, most real-world problems can be modeled as ILPs or MILPs. Notice that IPs seek the "best" solution (the global optimum) from a finite set. These types of problems are known as combinatorial optimization problems.

2.2.2 Combinatorial optimization

As mentioned before, combinatorial optimization problems involve finding the "best" solution (the global optimum) from a finite set. Naturally, they can be modeled as IPs. In other words, an IP is both a general combinatorial optimization problem and a modeling tool for other well-defined problems. For instance, the OneMax problem consists of finding a n-bit string with the maximum number of ones. The optimal solution for this toy combinatorial optimization problem is setting all variables to one, and the search space size is 2^n. Its corresponding ILP is the following, where $\mathbf{c} = [1, \ldots, 1]$ is a row vector of n components, and $\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$ is the column vector representing the n-bit string.

$$\begin{aligned} \max \quad & \mathbf{cx} \\ \text{s.t.} \quad & \mathbf{x} \in \{0,1\}^n \end{aligned}$$

Notice that $\mathbf{cx}$ is equivalent to $\sum_{i=1}^{n} x_i$. In fact, because of their easier readability, sigma (Σ) and pi (Π) notation are more frequently used in mathematical programs.

Although OneMax can be solved in linear time, a generic combinatorial optimization problem and its corresponding IP might belong to the $\mathcal{NP}$-hard class [2]. For instance, the Minimum Dominating Set problem (MDSP), a classical $\mathcal{NP}$-hard graph problem, has the following ILP, where a_{ij} are constants that codify the input graph $G = (V, E)$. Namely, $a_{ij} = 1$ if $v_i \in N(v_j)$; otherwise, $a_{ij} = 0$, where $V = \{v_1, \ldots, v_n\}$.

$$\begin{aligned} \min \quad & \textstyle\sum_{i=1}^{n} x_i & \\ \text{s.t.} \quad & \textstyle\sum_{i=1}^{n} a_{i,j} x_i \geqslant 1 - x_j & \forall v_j \in V \\ & x_i \in \{0,1\} & \forall v_i \in V \end{aligned}$$

$\mathcal{NP}$-hard combinatorial optimization problems are unlikely to be solved efficiently unless $\mathcal{P} = \mathcal{NP}$ [2, 3]. Nevertheless, generic exact algorithms can be used to find optimal solutions at the expense of time. Namely, algorithms like branch-and-bound, cutting planes, branch-and-cut, branch-and-price, among others [12], can be used but with no guarantee of having a practical termination time. Therefore, integer programming is normally used to tackle relatively *small* instances of $\mathcal{NP}$-hard combinatorial optimization problems. Heuristic and metaheuristic computing can be used to balance the disadvantages of the exact approach; more details are given in Section 2.4.

Many classical combinatorial optimization problems from computer science belong to a subcategory known as $\mathcal{NP}$-optimization. Formally, an $\mathcal{NP}$-optimization problem Π consists of a set of valid instances $D(\Pi)$, where each instance $I \in D(\Pi)$ has a set of feasible solutions $S_\Pi(I)$. There is a polynomial time algorithm that, given a pair (I, s), decides whether $s \in S_\Pi(I)$. Besides, a polynomial time computable objective function assigns a value to each pair (I, s) [13]. Except from OneMax, all the problems considered in this thesis are $\mathcal{NP}$-hard $\mathcal{NP}$-optimization combinatorial optimization problems.

2.3 Operations research

Operations research (a.k.a. *operational research* and *management science*) is the scientific specialty about modeling decision problems and computing feasible, and if possible

optimal, solutions to them [6]. Its origins come from 1947, when the field of linear programming, together with its posterior extensions (mathematical programming), was established as the most widely used tool in the industry for planning and scheduling [9].

Operations research was born from the combination of two important events: World War II and the increasing capacities of computers. The military necessity for programming (scheduling) tasks with optimal resource usage opened the door to mathematicians and scientists. Soon after, the scope of this area extended beyond military tasks. Some of the main pioneers of this research area were George B. Dantzig and John von Newman [9].

Since its birth, operations research has considered mathematical programming its main scientific tool. Besides, it has always emphasized real-world problems, which can be categorized as scheduling, location, routing, network flow, logistics, transportation, and many others [9, 14]. For this thesis's purposes, location and routing problems are more relevant. Thus, the following sections focus on them.

2.3.1 Location

Since the first years of operations research and computer science, location problems have been identified as relevant real-world optimization problems. After a few decades, they accumulated and grew in complexity. Consequently, around 1960, a whole research field known as *location science* (a.k.a. *location theory*) emerged. Nevertheless, some important location problems had been modeled and studied centuries ago by characters like Evangelista Torricelli (1608–1647) and Bonaventura Francesco Cavalieri (1598-1647) [15]. More recently, Alfred Weber introduced important location problems in 1909 [15]. However, it was not until 1964 that location science gained more researchers' interest with a publication by Hakimi (1964), who wanted to locate switching centers in a communications network and police stations in a highway system [16].

The core of location science is facility location problems. These consist of determining the "best" location for one or several facilities or equipment in order to serve a set of demand points. The meaning of "best" depends on the constraints and the optimality criteria considered [17]. Naturally, mathematical programming is a suitable modeling tool for this kind of problem.

There are many prominent and popular location problems, such as Weber's problems, k-median, facility location, k-center, etc [17]. Nevertheless, the next section focuses on vertex k-center problems for this document's convenience.

2.3.1.1 Vertex k-center

One of the fundamental location problems that gave rise to many generalizations is the absolute 1-center problem. Introduced by Hakimi in 1964, this problem models a situation where the best location for a police station in a highway system has to be found [16]. This problem aims to minimize the distance from the farthest community to the police station. In graph theory terms, an absolute 1-center is a location along any edge that minimizes the distance from the farthest vertex to such location. The Vertex k-center Problem (VKCP) generalizes the absolute 1-center problem by adding the following constraint: $k \geqslant 1$ centers must be located at the vertices of the input graph.

The VKCP is $\mathcal{NP}$-hard and models real-world problems where k centers need to attend clients, and a cost metric (e.g. travel time, distance, etc.) from the clients to their nearest center must be minimized. These centers may be hospitals, schools, police stations, etc. The first proposed MILP for VKCP is the following [18], where the input graph $G = (V, E)$ is complete and weighted, $V = \{v_1, \ldots, v_n\}$, $n = |V|$, and d_{ij} is the distance between v_i and v_j.

$$\min \quad z \tag{2.4}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} d_{i,j} x_{i,j} \leqslant z \qquad \forall v_j \in V \tag{2.5}$$

$$\sum_{i=1}^{n} x_{i,j} = 1 \qquad \forall v_j \in V \tag{2.6}$$

$$x_{i,j} \leqslant y_i \qquad \forall v_i, v_j \in V \tag{2.7}$$

$$\sum_{i=1}^{n} y_i \leqslant k \tag{2.8}$$

$$x_{i,j}, y_i \in \{0, 1\} \qquad \forall v_i, v_j \in V \tag{2.9}$$

The Objective function (2.4) and Constraints (2.5) guarantee that the objective value is not greater than the maximum of the distances between demand points (all vertices in V) and the centers they are assigned to. Constraints (2.6) establish that each vertex is assigned to exactly one center. Constraints (2.7) avoid the assignment of vertices to other vertices where a center is not located. Constraint (2.8) restricts the number of centers to k. Nowadays, equivalent mathematical programs with theoretical and practical advantages are available [19, 20, 21].

Among the many variants of VKCP, the Capacitated Vertex k-center Problem

(CVKP) [22] stands out by adding a capacity restriction. Namely, each center can attend a maximum number of clients. This problem is better suited to real-world scenarios and can be used as a clustering technique aiming for load-balance in other problems, including routing ones. In fact, this approach was applied to DFmTSP and is explained later in this document. In particular, novel IPs and algorithms for CVKCP are introduced in Section 5.1.

2.3.2 Routing

The main component of logistics problems is routing. Modern logistics can be described as:

> *the process of strategically managing the movement or storage of materials, parts and finished, inventory from suppliers, through the firm and on to the consumers.*
>
> (Christopher M., 1985)

Companies ' business strategies often emphasize the efficient movement of goods or workers to increase and meet market demands. Distribution costs are estimated to account for approximately 10% of the firms' revenues [23]. In some cases, like in the soft drink industry, they represent approximately 70% of goods' value-added costs [24]. In the U.S.A., logistics costs have been estimated at between 15% and 23% of the gross national product. For individual firms, logistics costs represent approximately 20% of the firm's net sales [25].

The impact of good routing models and algorithms to solve them is evident. In fact, some of the two most important and popular combinatorial optimization problems from computer science belong to this category: the Shortest Path Problem (SPP) and the Traveling Salesperson Problem (TSP). Generally, the input data for both problems is modeled through a complete weighted graph. The SPP problem asks for the shortest path from a given origin and a given destiny. This problem belongs to the $\mathcal{P}$ class. If the origin equals the destiny and an extra constraint of visiting all vertices from the input graph is included, the problem becomes the TSP, which is part of the $\mathcal{NP}$-hard class.

It is not an exaggeration to state that the TSP problem is one of the most famous $\mathcal{NP}$-hard combinatorial optimization problems. In fact, it has been deeply studied by the pioneers of mathematical programming and operations research since 1954 [26]. Thus, it is strongly related to the progress of computer science. Besides, to some extent, every routing and logistics problem is rooted in the TSP. There are some other

important routing problems, like the vehicle routing problem (VRP) and its many generalizations [27]. However, since the DFmTSP is a generalization of TSP, the next section focuses on the latter.

2.3.2.1 Traveling salesperson problem

TSP is among the most popular $\mathcal{NP}$-hard combinatorial optimization problems. Its first explicit appearance in the scientific literature dates from 1954 [26], and its first mathematical formulations are from 1959 and 1960 [28, 29]. Interestingly, the authors of these early papers stated the problem using the depot concept, a special vertex where the salesperson starts and finishes her path. Nevertheless, it is easy to observe that the depot plays only a symbolic and didactic role, i.e., it is equivalent to stating that the path must be closed. An ILP for TSP is the following [29], where $G = (V, E)$ is an input complete weighted graph and $n = |V|$. Notice that the input graph is Hamiltonian because it is complete. Thus, the TSP can be stated as the problem of finding a minimum weight Hamilton cycle.

$$\min \quad \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,j} \tag{2.10}$$

$$\text{s.t.} \quad \sum_{v_j \in V} x_{i,j} = 1 \qquad \forall v_i \in V \tag{2.11}$$

$$\sum_{v_i \in V} x_{i,j} = 1 \qquad \forall v_j \in V \tag{2.12}$$

$$t_i - t_j + n x_{i,j} \leqslant n - 1 \qquad \forall v_i, v_j \in V \tag{2.13}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall v_i, v_j \in V \tag{2.14}$$

In this model, $c_{i,j}$ is the cost of going from vertex v_i to vertex v_j. If the solution includes an edge from v_i to v_j, then $x_{i,j} = 1$; otherwise, $x_{i,j} = 0$. The Objective function (2.10) minimizes the weight of the solution. Constraints (2.11) and (2.12) guarantee that each vertex is visited once. Constraints (2.13) avoid the creation of subtours; variables t_i are arbitrary real numbers. This last set of constraints is so relevant that has its own name: the Miller-Tucker-Zemlin (MTZ) subtour elimination constraints (SECs). Finally, (2.14) define the decision variables. Some direct generalizations of TSP are the mTSPs, of which DFmTSP is a member. In Chapter 4, new IPs for the latter are introduced.

Many real-world combinatorial optimization problems studied in operations research belong to the $\mathcal{NP}$-hard class. Although relatively small instances can be solved optimally using classical algorithms like branch-and-bound, cutting planes, branch-and-cut, etc., most instances are more difficult to tackle, and other techniques are

required. These include heuristics, metaheuristics, matheuristics, hyperheuristics, and simheuristics. The next section focuses on heuristics and metaheuristics.

2.4 Heuristics and metaheuristics

Ideally, the first step of algorithm design consists of identifying how difficult the problem is. Thus, if the problem belongs to the $\mathcal{P}$ class, an effort to design exact algorithms should be made. However, if the problem belongs to the $\mathcal{NP}$-hard class, a compromise between execution time and optimality should be considered. Namely, we might conform to "good enough" solutions that are found relatively fast. Although a formal theory of heuristics does not exist, most definitions of *heuristic* are similar.

> *A heuristic is a simple procedure that helps find adequate, though often imperfect, answers to difficult questions [30].*
>
> (Daniel Kahneman, 2017)

> *Heuristics are rules-of-thumb, i.e., simple polynomial-time algorithms. They work quickly and efficiently. However, the quality of the solution they deliver is another matter altogether [31].*
>
> (Dorit Hochbaum, 1997)

> *Heuristic, or heuretic, or* ars inveniendi *was the name of a certain branch of study, not very clearly circumscribed, belonging to logic, or to philosophy, or to psychology, often outlined, seldom presented in detail, and as good as forgotten today. The aim of heuristic is to study the methods and rules of discovery and invention [32].*
>
> (George Polya, 1945)

Ethimologically, *heuristic* originates from the Greek word *heurísko*, which means to discover or find out. Thus, heuristics are related to the discipline or art of discovering, which includes what-if scenarios and trial and error [33]. In computer science, the word heuristic can be used as an adjective and noun. However, in general, *heuristic* and *heuristic algorithm* can be used interchangeably.

Despite being many different kinds of heuristics, the community usually distinguishes between two main different types: *constructive* and *local search* methods [34].

Constructive methods are algorithms that generate solutions by incrementally adding elements to a partial solution until it is complete. This kind of heuristics tends to be very fast. However, the returned solutions may be of inferior quality compared with other algorithms. Within this category are greedy algorithms. The other kind of heuristics is *local search* algorithms. These differ in that they start from some initial solution that can be generated in any way and iteratively try to improve it by exploring the neighborhood of such a solution. Most heuristics are usually deterministic and thus vulnerable to getting stuck on locally optimal solutions. For instance, the Lin–Kernighan heuristic (LKH) is one of the best heuristics for the TSP. It is a local search algorithm that improves an input tour (Hamilton cycle) by exploring its neighborhood. Every time a shorter tour is found, the process is repeated until no better tour can be found. For this specific heuristic, a neighborhood is defined by considering the number of edges that are in one tour but not the other.

Opposed to constructive and local search heuristics, which usually get trapped into local optima, metaheuristics have a larger exploration level that helps find globally optimal solutions. That is what the prefix *meta* is meant to emphasize: a metaheuristic goes "beyond" a heuristic. Fred Glover coined the term in 1986 when he faced problems that could not be solved by mathematical programming methods and heuristics alone.

> *Metaheuristics are solution methods that orchestrate an interaction between local improvement procedures and higher level strategies to create a process capable of escaping from local optima and performing a robust search of a solution space [35].*
>
> (Michel Gendreau and Jean-Yves Potvin, 2019)

> *Metaheuristics are flexible frameworks that can be used to design heuristics for virtually any combinatorial optimization problem. [...] designing an efficient metaheuristics is an art requiring a lot of intuition on the part of the metaheuristic designer [36].*
>
> (Marc Sevaux et al., 2018)

> *A metaheuristic is a high-level problem-independent algorithmic framework that provides a set of guidelines or strategies to develop heuristic optimization algorithms [37].*
>
> (Kenneth Sörensen and Fred Glover, 2013)

In summary, a metaheuristic is a general suggestion that guides the search for the globally optimal solution. They are more complex than heuristics because the latter

tend to focus on finding locally optimal solutions [38]. Besides, heuristics are particular to a problem, while metaheuristics are general design frameworks.

Many metaheuristics have been proposed over the years. In fact, many have been accused of being unnecessarily complicated copies of already-known metaheuristics [37, 39]. Nevertheless, there are some that have prevailed over the decades and that rely on clear principles.

- Genetic algorithms (GAs). Formally introduced by John Holland in 1975, this metaheuristic takes inspiration from the natural selection process and is one of the most successful global search techniques [40].
- Tabu search (TS). Introduced by Fred Glover in 1986, this metaheuristic relies on a simple principle: *if you want different results, do something different*. This is achieved through the concept of *tabu lists* [41].
- Greedy randomized adaptive search procedures (GRASP). Introduced by Feo and Resende in 1989, this metaheuristic randomizes an input constructive deterministic heuristic [42].
- Variable neighborhood search (VNS). Introduced by Nenad Mladenović in 1995, this metaheuristic exploits the fact that every global optimum is a local optimum. Therefore, the optimum is eventually reached by applying local search using many neighborhood structures [43].

Metaheuristics can be considered an area of soft computing (SC) and computational intelligence (CI). Within these research areas, evolutionary computing (EC) stands out. For this document's convenience, the next section focuses on EC, with a particular emphasis on GAs.

2.4.1 Evolutionary computing

Evolutionary computing is a computer science research field inspired and motivated by natural evolution. The power of natural evolution is evident in the variety of species that live in our world, as each is tailored to survive in its own ecosystem. In general, the process of natural evolution is quite complex since it is influenced by many factors. For now, let us consider natural evolution in a simpler description: first, a population of individuals in an environment strives for survival and reproduction. Each of these individuals has a fitness that is determined by the environment and relates to how well they are adapted to survive in such an environment. In other words, the better the adaption, the better the fitness.

Table 2.1: Analogies used of evolutionary computing between natural evolution and problem-solving.

Evolution		Problem solving
Environment	$\longleftrightarrow$	Problem
Individual	$\longleftrightarrow$	Candidate solution
Fitness	$\longleftrightarrow$	Quality

In the context of computing and problem-solving processes, the individuals represent a collection of candidate solutions whose quality is determined by how well they solve the problem (how well they are adapted to the environment). The analogy of evolutionary computing is shown in Table 2.1.

The idea of applying natural evolution to problem-solving in computer science dates back to the 1940s when Turing proposed "genetical or evolutionary search". Later, in 1962, Bremermann worked with optimization through evolution and recombination. In fact, during the 1960s three similar ideas of evolutionary computing emerged in different locations. Fogel, Owens, and Walsh introduced **evolutionary programming** in the USA [44]. At the same time, Holland worked with similar ideas, but his method was called **genetic algorithms** [40]. Meanwhile, Rechenberg and Schwefel proposed **evolution strategies** in Germany [45]. For several years afterward, these methods were worked and developed separately, but it was not until the 1990s that they started being viewed as a unified field known as evolutionary computing. Later, in the 1990s, a new flavor of evolutionary computing known as **genetic programming** arose, proposed by Koza [46].

Since many combinatorial optimization problems belong to the $\mathcal{NP}$-hard class, under $\mathrm{P} \neq \mathrm{NP}$ no algorithm can solve them in polynomial time. Thus, this is the main limitation of IP and exact algorithms. So, algorithmic approaches such as heuristics and metaheuristics have been proposed to find good quality solutions in feasible running times. Within the metaheuristic proposals, in general, evolutionary computing stands out as a paradigm to approach this kind of computational problem. Finally, despite heuristics and metaheuristics (including evolutionary computing) being considered powerful tools to approach these hard problems, they usually do not find the optimum in optimization problems. Furthermore, most of them do not have theoretical guarantees, and their effectiveness has to be proven through experimentation.

Chapter 3

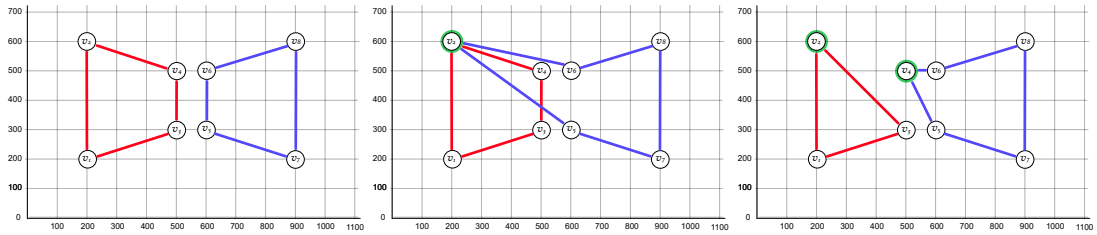
Related work and state-of-the-art

This chapter begins with a chronological account of the main results related to mTSPs. Afterward, independent sections explain some of the most relevant state-of-the-art models and algorithms in detail.

3.1 Multiple Traveling Salesperson Problems (mTSPs)

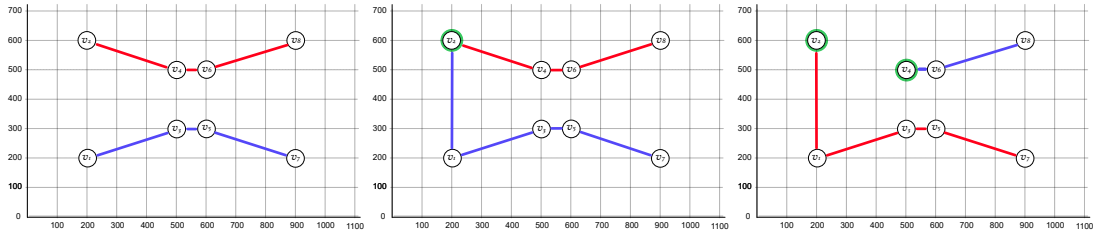
Every mTSP generalizes the canonical $\mathcal{NP}$-hard TSP. While the TSP seeks a minimum closed path that visits all vertices, the input for an mTSP is a weighted complete graph $G = (V, E)$ and a positive integer m, and its goal is to find a set of m paths such that all vertices are visited once by some salesperson [29, 47]. If the input graph is undirected (resp., directed), the problem is symmetric (resp., asymmetric). Naturally, the asymmetric version generalizes the symmetric one. An mTSP variant receives particular labels depending on its characteristics: depot-free (DF), single-depot (S), multiple-depots (M), closed-paths (CP), open-paths (OP), bounding constraints, etc. In most cases, the objective function to minimize is the sum of the paths' costs (minsum) or the largest path (minmax or makespan); other objective functions might be considered, such as the cost of the largest edge (bottleneck) [48].

The most widely studied mTSPs are Single-Depot mTSP (SmTSP) and Multiple-Depots mTSP (MmTSP). Nevertheless, the Depot-Free mTSP (DFmTSP) variant has received less attention. In SmTSP, all salespersons must start and finish their path at a specific vertex (the depot), which is part of the input. In the Fixed-Destination Multiple-Depots mTSP (FD-MmTSP), m depots are part of the input, and each salesperson must start and finish their path at their respective depot. In the Non-Fixed-Destination Multiple-Depots mTSP (NFD-MmTSP), each salesperson can finish their path at a different depot [48]. In DFmTSP, the depot concept is not involved. Therefore, it seeks a disjoint collection of closed paths that visit all vertices [4]. If the objective function is minsum and no other constraints are involved, this problem receives the



(a) Depot Free (DF) **(b)** Single Depot (S) **(c)** Multiple Depots (M)

Figure 3.1: Optimal solutions for mTSP with closed paths and bounding constraints ($m = 2$, $L = 3$, and $U = 5$).



(a) Depot Free (DF) **(b)** Single Depot (S) **(c)** Multiple Depots (M)

Figure 3.2: Optimal solutions for mTSP with open paths and bounding constraints ($m = 2$, $L = 3$, and $U = 5$).

name of Hamiltonian p-median problem [49]. Since this document considers many other constraints, we stick to the name DFmTSP. In all the mentioned mTSPs, every solution consists of exactly m paths, and the path followed by each salesperson is closed. However, if the salespersons do not need to return to their depot, the problem is an open-paths (OP) variant. Figures 3.1 and 3.2 show a set of optimal solutions for SmTSP, FD-MmTSP, and DFmTSP to clarify these variants' differences. Notice that, in all these examples, each path must have between three and five vertices; these are bounding constraints that aim at preserving load balance.

To date, some surveys on mTSP have been published [48, 50, 51]. However, they are not structured in chronological order. Thus, one of the papers resulting from this research presents a historical account of mTSP that sheds light on how DFmTSP has received very little attention [52]. To avoid repeating information, this document only summarizes the larger historical account published in the aforementioned paper.

3.1.1 Historical development

The TSP has been studied since 1954 [26]. Its first mathematical models are from 1959 and 1960 [28, 29]. For instance, see Expressions (2.10) to (2.14) at Chapter 2. Naturally, mTSP with one salesperson ($m = 1$) is equivalent to TSP; therefore, mTSP is $\mathcal{NP}$-hard too. In fact, the aforementioned ILP (Expressions (2.10) to (2.14)) can be easily adapted to the SmTSP with closed paths and minsum objective function by adding the extra Constraint (3.5), where $G = (V, E)$ is the input graph, m is the number of salespersons, $V = \{v_1, \ldots, v_n\}$, and v_1 is the depot.

$$\min \quad \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,j} \tag{3.1}$$

$$\text{s.t.} \quad \sum_{v_j \in V} x_{i,j} = 1 \qquad \forall v_i \in V \tag{3.2}$$

$$\sum_{v_i \in V} x_{i,j} = 1 \qquad \forall v_j \in V \tag{3.3}$$

$$t_i - t_j + n x_{i,j} \leqslant n - 1 \qquad \forall v_i, v_j \in V \tag{3.4}$$

$$\sum_{i=1}^{n} x_{i,1} = m \tag{3.5}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall v_i, v_j \in V \tag{3.6}$$

mTSPs gained more attention between 1973 and 1975, and more efficient mathematical models were introduced [47, 53]. Some of the most remarkable of them are based on SmTSP being reduced in polynomial time to TSP by adding some extra vertices to the original graph [53].

Between 1976 and 1995, exact algorithms were proposed; these were mainly based on Benders decomposition, cutting planes, and branch-and-bound [52]. Besides, some heuristics for SmTSP were introduced too. By 1995, SmTSP with a minsum objective function was the most studied mTSP; only the MmTSP with two salespersons ($m = 2$) was mentioned and reduced to TSP in 1980 [54]. It was not until 1995 that MmTSP was reduced to TSP [55]. Only a tabu search metaheuristic was developed in this period for SmTSP [56].

From 1996 to 2005, MmTSP gained more attention, and some more heuristics and metaheuristics considering minsum and minmax objective functions for SmTSP, MmTSP, and DFmTSP were introduced too. The main approaches were neural networks, genetic algorithms, particle swarm optimization, evolutionary strategies, and simulated annealing [52]. In this period, a few authors started paying attention to

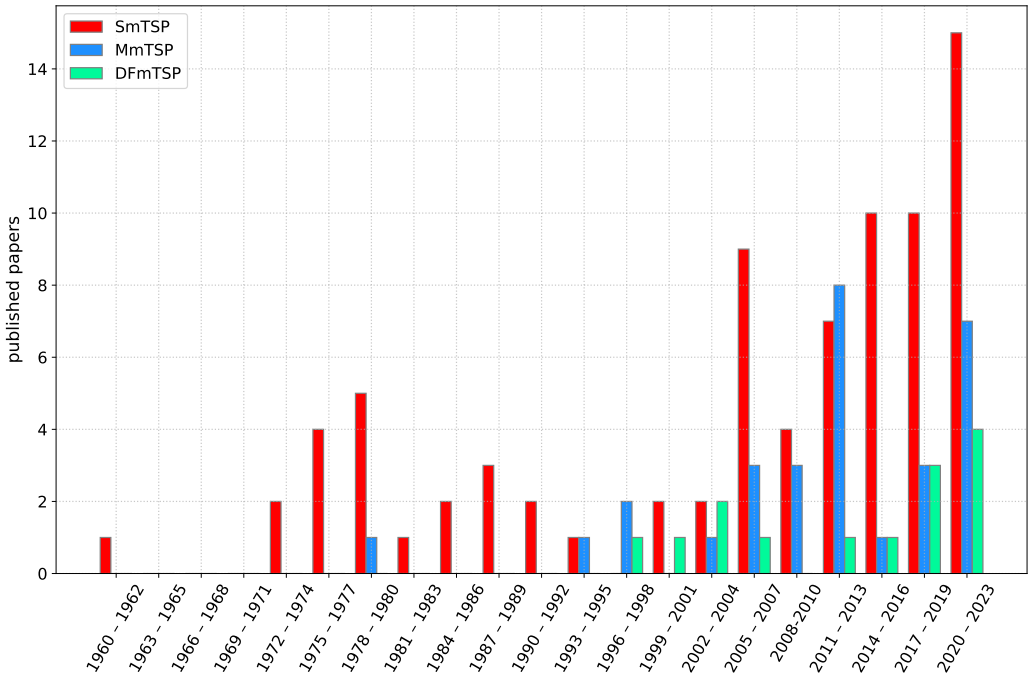


Figure 3.3: Number of published papers for mTSPs.

DFmTSP.

From 2006 to date, the interest in SmTSP, MmTSP, DFmTSP, and their variants has grown much more. Nevertheless, most of the research was still focused on SmTSP and the minsum objective function. Many heuristics, exact algorithms, and IPs were proposed during this period. In fact, it was metaheuristics that dominated the scene with neural networks, genetic algorithms, clustering strategies, ant colony optimization, firefly algorithm, ant colony system, market-based algorithms, imperialist competitive algorithm, tabu search, gravitational emulation local search algorithm, variable neighborhood search, bee colony optimization, invasive weed optimization, wolf pack search algorithm, discrete pigeon optimization, reinforcement learning, evolutionary strategies, hybrid search, memetic search, simulated annealing, and bees algorithm. As in years before, only a few authors worked on DFmTSP. Remarkably, it was not until 2017 and 2021 that the first reported IPs for DFmTSP were published [57, 4]. From them, we could reproduce and validate the IP of Karabulut et al. [4]. Figure 3.3 summarizes this section's historical account and lets us observe how DFmTSP has been scarcely studied through the years. It is worth mentioning that the Hamiltonian p-median problem was not considered in this chronological account.

3.1.2 Objective functions

There are two popular objective functions for mTSPs that have been used in the literature: minsum and minmax [58]. The first one aims to minimize the sum of the costs of the salespersons' paths (See Expression (3.7)).

$$\min \sum_{i=1}^{m} c(p_i) \tag{3.7}$$

The sum goes from 1 to m, p_i represents the path of the ith salesperson, and $c(p_i)$ is the cost of such path. This equation is the same for closed or open paths. The second objective function, minmax, aims at minimizing the longest path among salespersons. Expression (3.8) defines the *minmax* objective function, like minsum, which can be used for closed or open paths.

$$\min \max_{1 \leqslant i \leqslant m} c(p_i) \tag{3.8}$$

Other objective functions have been scarcely considered, such as bottleneck, which aims to minimize the weight of the highest-weight edge [59]. Nevertheless, minsum is definitely the most frequently used objective function.

3.2 Integer programs for mTSPs

This section presents the state-of-the-art IPs for the main variants of mTSP, i.e., SmTSP, MmTSP, and DFmTSP. These IPs are the most recent and complete available in the literature. They tackle the closed-paths (CP) version and integrate bounding constraints.

3.2.1 Single depot (SmTSP)

This section presents the most general ILP for SmTSP [48]. It uses the minsum Objective function (3.9) and considers closed paths with bounding constraints. In addition to m, two positive integers L and U are part of the input. So, each salesperson must visit between L and U vertices. The input graph $G = (V, E)$ has vertices $V = \{v_1, \ldots, v_n\}$, $v_d \in V$ is the depot, and $c_{i,j}$ is the cost of traveling from vertex v_i to v_j. Notice that $c_{i,j}$ can differ from $c_{j,i}$. Thus, this ILP models the asymmetric SmTSP, which is a generalization of the symmetric version.

$$\min \quad \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,j} \tag{3.9}$$

$$\text{s.t.} \quad \sum_{v_j \in V \setminus \{v_d\}} x_{d,j} = m \tag{3.10}$$

$$\sum_{v_j \in V \setminus \{v_d\}} x_{j,d} = m \tag{3.11}$$

$$\sum_{v_i \in V} x_{i,j} = 1 \qquad \forall v_j \in V \setminus \{v_d\} \tag{3.12}$$

$$\sum_{v_j \in V} x_{i,j} = 1 \qquad \forall v_i \in V \setminus \{v_d\} \tag{3.13}$$

$$t_i + (U-2)x_{d,i} - x_{i,d} \leqslant U - 1 \qquad \forall v_i \in V \setminus \{v_d\} \tag{3.14}$$

$$t_i + x_{d,i} + (2-L)x_{i,d} \geqslant 2 \qquad \forall v_i \in V \setminus \{v_d\} \tag{3.15}$$

$$x_{d,i} + x_{i,d} \leqslant 1 \qquad \forall v_i \in V \setminus \{v_d\} \tag{3.16}$$

$$t_i - t_j + U x_{i,j} + (U-2)x_{j,i} \leqslant U - 1 \qquad \forall v_i, v_j \in V \setminus \{v_d\} : v_i \neq v_j \tag{3.17}$$

$$x_{i,j} \in \{0,1\} \qquad \forall v_i, v_j \in V : v_i \neq v_j \tag{3.18}$$

where

$$x_{i,j} = \begin{cases} 1, & \text{if the } k\text{th salesperson visits vertex } v_j \text{ immediately after vertex } v_i \\ 0, & \text{otherwise} \end{cases} \tag{3.19}$$

$$t_i = \text{visiting rank of node } i \tag{3.20}$$

Constraints (3.10) and (3.11) guarantee that all salespersons depart and arrive at the depot. Constraints (3.12) and (3.13) guarantee that all vertices are visited once (except v_d). Constraints (3.14) to (3.17) guarantee that all the paths have between L and U vertices and include subtour elimination. It is worth mentioning that a path must have at least two vertices in this model and others to come. In other words, isolated vertices are forbidden as part of a solution.

3.2.2 Fixed-Destination Multiple depots (FD-MmTSP)

This section presents the more general IP for FD-MmTSP [48]. Namely, there are m depots as part of the input, and each salesperson must depart and arrive at its corresponding depot, i.e., it considers closed paths. For clarity, let us assume that the vertices in $G = (V, E)$ are labeled as $V = \{v_1, v_2, ..., v_n\}$, the set of depots is $D \subseteq V$ such

that $|D| = m$, and J is the set of vertices that must be visited by the salespersons, i.e., $J = V \setminus D$. $c_{i,j}$ is the cost of traveling from v_i to v_j. Notice that $c_{i,j}$ might differ from $c_{j,i}$; we are working with the asymmetric version of the problem.

$$\min \quad \sum_{v_k \in D} \sum_{v_j \in J} \left(c_{k,j} x_{k,j,k} + c_{j,k} x_{j,k,k} \right) + \sum_{k \in D} \sum_{i \in J} \sum_{j \in J} \left(c_{i,j} x_{i,j,k} \right) \tag{3.21}$$

$$\text{s.t.} \quad \sum_{v_j \in J} x_{k,j,k} = 1 \qquad \forall v_k \in D \tag{3.22}$$

$$\sum_{v_k \in D} x_{k,j,k} + \sum_{v_k \in D} \sum_{v_i \in J} x_{i,j,k} = 1 \qquad \forall v_j \in J \tag{3.23}$$

$$x_{k,j,k} + \sum_{v_i \in J} x_{i,j,k} - x_{j,k,k} - \sum_{v_i \in J} x_{j,i,k} = 0 \qquad \forall v_k \in D, j \in J \tag{3.24}$$

$$\sum_{v_j \in J} x_{k,j,k} - \sum_{v_j \in J} x_{j,k,k} = 0 \qquad \forall v_k \in D \tag{3.25}$$

$$t_i + (U-2) \sum_{v_k \in D} x_{k,i,k} - \sum_{v_k \in D} x_{i,k,k} \leqslant U - 1 \qquad \forall v_i \in J \tag{3.26}$$

$$t_i + \sum_{v_k \in D} x_{k,i,k} + (2-L) \sum_{v_k \in D} x_{i,k,k} \geqslant 2 \qquad \forall v_i \in J \tag{3.27}$$

$$\sum_{v_k \in D} x_{k,i,k} + \sum_{v_k \in D} x_{i,k,k} \leqslant 1 \qquad \forall v_i \in J \tag{3.28}$$

$$t_i - t_j + U \sum_{v_k \in D} x_{i,j,k} + (U-2) \sum_{v_k \in D} x_{j,i,k} \leqslant U - 1 \qquad \forall v_i, v_j \in J \tag{3.29}$$

$$x_{i,j,k} \in \{0,1\} \qquad \forall v_i, v_j \in V, v_k \in D \tag{3.30}$$

where

$$x_{i,j,k} = \begin{cases} 1, & \text{if the salesperson that departs from the } v_k \text{ dummy depot goes from } v_i \text{ to } v_j \\ 0, & \text{otherwise} \end{cases} \tag{3.31}$$

$$t_i = \text{time at which vertex } i \text{ is visited in the path} \tag{3.32}$$

$$2 \leqslant L \leqslant \lceil |V|/m \rceil \leqslant U \leqslant |V| \tag{3.33}$$

In this formulation, Expression (3.21) is the minsum objective function for the CP-FD-MmTSP. Constraints (3.22) guarantee that exactly one salesperson departs from each depot $v_k \in D$. Constraints (3.23) ensure that each vertex is visited exactly once by

a salesperson (a.k.a. flow constraints). Constraints (3.24) and (3.25) represent the route continuity for vertices and depots. Constraints (3.26)-(3.27), and (3.33) serve as upper and lower bound constraints on the number of vertices visited by each salesperson. That is, ensure that each salesperson visits between L and U vertices in his path. These constraints are called bounding constraints and are useful for load-balance among salespersons. Constraints (3.28) prohibit a salesperson from visiting only one vertex (a.k.a. return trips). Constraints (3.29) are the subtour elimination constraints (SECs) that avoid the formation of any subtours. Finally, Expressions (3.30)-(3.33) define the decision variables.

3.2.3 Depot free (DFmTSP)

As far as we know, there is only one IP for the DFmTSP that considers upper bound constraints, the minsum objective function, and closed paths [4]. As in previous IPs, each salesperson must visit at least two vertices in her tour.

$$\min \quad \sum_{v_i \in V} \sum_{v_j \in V: v_i \neq v_j} \left(c_{i,j} \sum_{k=1}^{m} x_{i,j,k} \right) \tag{3.34}$$

$$\text{s.t.} \quad \sum_{v_i \in V: v_i \neq v_j} \sum_{k=1}^{m} x_{i,j,k} = 1 \qquad \forall v_j \in V \tag{3.35}$$

$$\sum_{v_i \in V: v_i \neq v_p} x_{i,p,k} - \sum_{v_j \in V: v_j \neq v_p} x_{p,j,k} = 0 \qquad \forall v_p \in V, k \in [1, m] \tag{3.36}$$

$$\sum_{v_i \in V} \sum_{v_j \in V: v_i \neq v_j} x_{i,j,k} \geqslant 1 \qquad \forall k \in [1, m] \tag{3.37}$$

$$t_i - t_j + U \sum_{k=1}^{m} x_{i,j,k} \leqslant U - 1 + U z_j \qquad \forall v_i, v_j \in V : v_i \neq v_j \tag{3.38}$$

$$1 \leqslant t_i \leqslant U \qquad \forall v_i \in V \tag{3.39}$$

$$\sum_{v_i \in V} z_i = m \tag{3.40}$$

$$x_{i,j,k} \in \{0, 1\} \qquad \forall v_i, v_j \in V : v_i \neq v_j, k \in [1, m] \tag{3.41}$$

$$z_i \in \{0, 1\} \qquad \forall v_i \in V \tag{3.42}$$

where

$$x_{i,j,k} = \begin{cases} 1, & \text{if the } k\text{th salesperson visits vertex } v_j \text{ immediately after vertex } v_i \\ 0, & \text{otherwise} \end{cases} \tag{3.43}$$

$$z_i = \begin{cases} 1, & \text{if vertex } v_i \text{ is the starting (origin) vertex in a tour of any salesperson} \\ 0, & \text{otherwise} \end{cases} \tag{3.44}$$

$$t_i = \text{visiting rank of node } i \tag{3.45}$$

The Objective function is minsum (See Equation (3.34)), i.e., it is the total cost of the m tours. Constraints (3.35) ensure that each node is visited exactly once. Constraints (3.36) are the flow conservation constraints, which guarantee that if a salesperson visits a specific vertex, such salesperson must also depart from that vertex. Constraints (3.37) ensure that each salesperson is used in the problem. Constraints (3.38) and (3.39) represent the subtour elimination constraints (SECs) and the bounding constraints, where U is the maximum number of vertices that each salesperson can visit. Constraints (3.40) state that m starting vertices for m tours exist. Constraints (3.41) and (3.42) define the decision variables.

Although it was not mentioned by Karabulut et al., their IP is easily adapted to the minmax objective function by replacing Expression (3.34) with Expressions (3.46)-(3.47), where P_{max} is the maximum tour length among the salespersons.

$$\min \quad P_{max} \tag{3.46}$$

$$P_{max} \geqslant \sum_{i=1}^{n} \sum_{j=1:i\neq j}^{n} \left(c_{i,j} x_{i,j,k}\right) \qquad \forall k \in [1, m] \tag{3.47}$$

3.3 Location science in routing problems

Although location and routing problems have their own body of knowledge, they can be useful to one another. For instance, a common strategy for finding feasible solutions to routing problems is known as *clustering-routing* or *cluster-first route-second*. It is difficult to establish who first devised such an intuitive strategy. However, one of the earlier works where it can be found is [60]. In that paper, a heuristic named MTOUR for the Vehicle Dispatch Problem (VDP) was introduced; this problem seeks m routes for m vehicles under certain constraints.

The idea behind clustering-routing is simple: there are two phases. The first groups vertices, and the second determines feasible routes for each cluster. Although location problems do not explicitly group vertices, they can be used to that end.

3.3.1 Clustering-routing for mTSP

Some works where clustering-routing has been used for mTSPs are the following. For SmTSP, Bolaños et al. [61] proposed heuristic algorithms for the clustering phase, and then a GA with intra-route heuristics is used to improve the quality of the routes. Another work that uses the clustering-routing strategy is [62], where a variation of the k-means algorithm is used at the clustering phase, then a GA is used to build a route within each cluster. In fact, most works have used variations of the k-means algorithm for the clustering phase for the mTSP [63, 64, 65]. An interesting point of [65] is that the authors used a parallel approach to improve the execution times.

Regarding MmTSP the situation is similar, variations of the k-means clustering have been used in [66, 67]. It is worth noting that, for the second phase, most authors have used GAs, ant-based algorithms, and hybridizations between them.

3.3.2 Capacitated Vertex k-center as a clustering technique

Along with k-means, k-center problems (See Section 2.3.1.1) are natural clustering methods. In particular, the capacitated version of VKCP (CVKCP) imposes load-balance by considering an upper bound on the number of *clients* each center can attend. The CVKP has not been used as a clustering strategy for mTSP in the literature. Thus, part of this project explored its advantages as a clustering technique. The basic MILP for CVKCP is the following, where $G = (V, E)$ is the input graph, $V = \{v_1, \ldots, v_n\}$, k is the number of centers, and $f_c : V \to \mathbb{N}$ is a capacity function that specifies how many vertices can be assigned to each center.

$$\min \quad z \tag{3.48}$$

$$\text{s.t.} \quad \sum_{i=1}^{n} d_{i,j} x_{i,j} \leqslant z \qquad \forall v_j \in V \tag{3.49}$$

$$\sum_{i=1}^{n} x_{i,j} = 1 - y_i \qquad \forall v_j \in V \tag{3.50}$$

$$x_{i,j} \leqslant y_i \qquad \forall v_i, v_j \in V \tag{3.51}$$

$$\sum_{i=1}^{n} y_i \leqslant k \tag{3.52}$$

$$\sum_{i=1}^{n} x_{i,j} \leqslant f_c(v_j) \qquad \forall v_j \in V \tag{3.53}$$

$$x_{i,j}, y_i \in \{0, 1\} \qquad \forall v_i, v_j \in V \tag{3.54}$$

In comparison to the MILP for the uncapacitated version (Expressions (2.4) to (2.9) at Section 2.3.1.1), this MILP only adds Constraints (3.53), which imposes a load balance given by the input function f_c. Besides, Constraints (3.50) avoids a center to be assigned to itself. Namely, centers do not need to be *attended* by anyone. Now, notice that a solution to CVKCP is a set of vertices, not a cluster of vertices. However, such cluster can be obtained through decision variables $x_{i,j}$, which indicates which vertices are assigned to which centers; since there are k of them, k clusters can be inferred. In Section 5.1, novel IPs for CVKCP are introduced. Afterward, they are used as the basis for novel clustering-routing heuristics for DFmTSP.

3.4 Heuristics and metaheuristics for mTSP

As previously mentioned, the mTSP is $\mathcal{NP}$-hard. For this reason, in the last years, many researchers have focused on using heuristic and metaheuristic algorithms for this problem in order to try to solve relatively big instances in practical amounts of time [68].

In general, the literature on the mTSPs includes many metaheuristics for the mTSPs that consider depots as part of the problem. Nevertheless, there are just a few proposals for the mTSP that consider the specific restrictions that we do in this thesis, the DFmTSP with bounding constraints. In the literature, evolutionary computing stands as a good approach that has been capable of finding good-quality solutions for this studied problem. Such is the case of [69], where a Partheno Genetic Algorithm

(PGA) for the DFmTSP is proposed, but it only considers lower bound constraints. That is, each path is constrained to visit a minimum number of vertices but not a maximum. Despite this, this PGA can be adapted to consider lower-bound constraints by making the appropriate modifications. Also, [1] proposes a metaheuristic combining an Ant Colony and a Partheno Genetic Algorithm. This algorithm is called AC-PGA and considers both lower-bound and upper-bound constraints. In [1], experimentation is performed to compare the performance between the PGA of [69] and the AC-PGA, concluding that AC-PGA outperformed it, finding solutions of better quality. Algorithm 1 shows the pseudocode of the AC-PGA.

Input: A population P, a natural selection ratio γ, a distance matrix D, a pheromone volatilization rate ρ, the importance of pheromone and visibility α and β, ACO steps iteration n_0, and maximum iteration N_0.
Output: The best found solution

```
1 Initialize |P| pairs of depot sequence and city number sequence, then associate
    them with individuals in initial population
2 while ¬StopCondition() do
3     // ACO steps
4     Run ACO-based algorithm
5     // PGA steps
6     Calculate each individual's fitness function value Retain the individual with
        larger fitness value on the ratio of γ
7     Use the retained individuals to produce offspring randomly and each kid's
        gene is composed of a depot sequence and a city number sequence
8 end
9 return best found solution
```

Algorithm 1: AC-PGA.

Input: A population P, a distance matrix D, a pheromone volatilization rate ρ, the importance of pheromone and visibility α and β, ACO steps iteration n_0, and maximum iteration N_0.

Output: The best found solution

Start from the first individual
do
  Initial pheromone matrix τ and tabu list A
  do
    Put an ant on each salesman's depot. From the first ant, each of them moves to the next cities in order
    do
      Update tabu list A
      if *current ant has finished its tour* **then**
        Turn to the next ant
      else
        Assume that the current ant is at the ith vertex and the probability of moving to the jth vertex is $p_{i,j}$. The calculation formula shows as follows:

$$p_{i,j} = \begin{cases} \frac{\tau(i,j)^{\alpha} c_{i,j}{}^{\beta}}{\sum_{k \notin A} \tau(i,k)^{\alpha} c_{i,k}{}^{\beta}} & \text{, if } j \notin A \\ 0 & \text{, otherwise} \end{cases}$$

      end
    while $|A| \neq n$
  while *The number of iterations reaches* n_0
  Get a feasible solution and replace individual's gene with it
while *Perform ACO steps on each individual*
return best found solution

Algorithm 2: ACO-based algorithm.

Chapter 4

Mathematical models of DFmTSP

This chapter presents new integer programs (IPs) for DFmTSP and its primary variants, featuring *dummy depots* as a key characteristic. The section is structured into two parts. Section 4.2 introduces IPs for the CP and OP variants, incorporating bounding constraints and both minsum and minmax objective functions. These IPs include $\mathcal{O}(n^2m)$ binary variables and are inspired by an existing IP for FD-MmTSP proposed in [48]. Subsequently, Section 4.3 proposes more compact IPs for the CP and OP variants with bounding constraints and a minsum objective function. These IPs reduce the number of binary variables to $\mathcal{O}(n^2)$.

4.1 Dummy depots

Definition 4.1.1. *A dummy depot is a vertex* $v_k \notin V(G)$*, such that* $\forall v_i \in V(G), c_{i,k} = c_{k,i} = 0$*, where* G *is the input graph and* $c_{a,b}$ *is the cost of the edge* (v_a, v_b)*. As the name suggests, it is a "fake depot".*

4.2 Model 1: based on FD-MmTSP

This section extends a state-of-the-art IP for the FD-MmTSP (See Section 3.2.2) [48]. The connection between FD-MmTSP and DFmTSP comes from the following intuitive Observation 1.

Observation 1. *A solution to CP-DFmTSP defines a partition of vertices. If one vertex from each partition's element is known in advance, the problem can be directly modeled as CP-FD-MmTSP.*

Expressions (4.1)-(4.13) define an IP for the minsum CP-DFmTSP with bounding constraints. This IP will serve as the foundation for the remaining integer programs in this chapter. Specifically, we will demonstrate how to adapt this model to the minmax objective function and the OP variant. To begin, let us describe this IP in

detail. Let V denote the set of vertices in the complete weighted graph $G = (V, E, w)$, and let D represent a set of m dummy depots. For generality, assume that the input graph is directed. For clarity, the vertices in V are labeled as $\{v_1, v_2, \ldots, v_n\}$, while the dummy depots in D are labeled as $\{v_{n+1}, v_{n+2}, \ldots, v_{n+m}\}$, where $V \cap D = \emptyset$. The cost of traveling from v_i to v_j is denoted as $c_{i,j}$. Note that $c_{i,j}$ may differ from $c_{j,i}$, meaning we are addressing the asymmetric version of the problem, which generalizes its symmetric counterpart.

$$\min \quad \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} \sum_{v_k \in D} \left(x_{i,j,k} + x_{i,k,k} x_{k,j,k} \right) \tag{4.1}$$

$$\text{s.t.} \quad \sum_{v_j \in V} x_{k,j,k} = 1 \qquad \forall v_k \in D \tag{4.2}$$

$$\sum_{v_k \in D} x_{k,j,k} + \sum_{v_k \in D} \sum_{v_i \in V} x_{i,j,k} = 1 \qquad \forall v_j \in V \tag{4.3}$$

$$x_{k,j,k} + \sum_{v_i \in V} x_{i,j,k} - x_{j,k,k} - \sum_{v_i \in V} x_{j,i,k} = 0 \qquad \forall v_k \in D, v_j \in V \tag{4.4}$$

$$\sum_{v_j \in V} x_{k,j,k} - \sum_{v_j \in V} x_{j,k,k} = 0 \qquad \forall v_k \in D \tag{4.5}$$

$$t_i + (U-2) \sum_{v_k \in D} x_{k,i,k} - \sum_{v_k \in D} x_{i,k,k} \leqslant U - 1 \qquad \forall v_i \in V \tag{4.6}$$

$$t_i + \sum_{v_k \in D} x_{k,i,k} + (2-L) \sum_{v_k \in D} x_{i,k,k} \geqslant 2 \qquad \forall v_i \in V \tag{4.7}$$

$$\sum_{v_k \in D} x_{k,i,k} + \sum_{v_k \in D} x_{i,k,k} \leqslant 1 \qquad \forall v_i \in V \tag{4.8}$$

$$t_i - t_j + U \sum_{v_k \in D} x_{i,j,k} + (U-2) \sum_{v_k \in D} x_{j,i,k} \leqslant U - 1 \qquad \forall v_i, v_j \in V \tag{4.9}$$

$$x_{i,j,k} \in \{0, 1\} \qquad \forall v_i, v_j \in V \cup D, \forall v_k \in D \tag{4.10}$$

where

$$x_{i,j,k} = \begin{cases} 1, & \text{if the salesperson at the } v_k \text{ dummy depot goes from } v_i \text{ to } v_j \\ 0, & \text{otherwise} \end{cases} \tag{4.11}$$

$$t_i = \text{time at which vertex } v_i \text{ is visited in the path} \tag{4.12}$$

$$2 \leqslant L \leqslant \lceil |V|/m \rceil \leqslant U \leqslant |V| \tag{4.13}$$

In this IP, Constraints (4.2) ensure that exactly one salesperson departs from each dummy depot $v_k \in D$. Constraints (4.3) ensure that each vertex $v_j \in V$ is visited

exactly once from some vertex $v_i \in V \cup D$. Constraints (4.4) and (4.5) enforce route continuity for both vertices and dummy depots. Constraints (4.6), (4.7), and (4.13) ensure that each salesperson visits between L (lower bound) and U (upper bound) vertices, collectively referred to as bounding constraints. Constraints (4.8) prevent a salesperson from visiting only one vertex (a.k.a., a return trip). Constraints (4.9) are subtour elimination constraints (SECs), ensuring that $t_j = t_i + 1$ if and only if $x_{i,j} = 1$. Constraints (4.10) define the decision variables. Note that we use the term *time* metaphorically, where variables t represent the order in which a salesperson visits vertices. Finally, the objective function (4.1) minimizes the total cost of all paths (minsum). The number of binary variables and constraints is $\mathcal{O}(n^2m)$ and $\mathcal{O}(n^2)$, respectively. Note that the objective function contains a quadratic term, making this program an integer quadratic program (IQP). However, as we will show later, it can be linearized by introducing additional variables and constraints.

Figure 4.1 illustrates the optimal solution of this IP applied to a small graph with $m = 2$, $L = 3$, and $U = 5$. In this figure, two salespersons depart from and return to the dummy depots v_9 and v_{10}, respectively. Since traversing a dummy depot incurs no cost, we must identify the edges (v_2, v_1) and (v_6, v_5) and include their costs in the objective function, as explained in Lemma 1. Regarding the matrices $\mathbf{x}_9$ and $\mathbf{x}_{10}$, each corresponds to a distinct dummy depot and encodes a unique closed path.

Lemma 1. *Expression (4.1) is the minsum objective function for CP-DFmTSP.*

Proof. Expression (4.1) comes from the following expanded form:

$$\underbrace{\sum_{v_k \in D} \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,j,k}}_{\text{first term}} + \underbrace{\sum_{v_k \in D} \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,k,k} x_{k,j,k}}_{\text{second term}} \tag{4.14}$$

The first term of Expression (4.14) adds up the cost of the traveled edges (solid red and blue edges in Figure 4.1). The second term adds up the cost $c_{i,j}$ of the edge of each pair of vertices $v_i, v_j \in V$ adjacent to a dummy depot $v_k \in D$ (dotted red and blue edges in Figure 4.1). Let v_k be any dummy depot in D, and let v_i and v_j be any pair of different vertices in V. If $x_{i,k,k} = x_{k,j,k} = 1$, then the salesperson associated with the v_k dummy depot goes from vertex v_i to v_k and then from vertex v_k to v_j. Thus, the path is closed if we consider the edge (v_i, v_j). This way, the cost of edge (v_i, v_j) is considered in the objective function because $c_{i,j} x_{i,k,k} x_{k,j,k} = c_{i,j}$. Finally, the Objective function (4.14) considers the sum of the paths of all the salespersons. Therefore, it is the minsum objective function for CP-DFmTSP. □

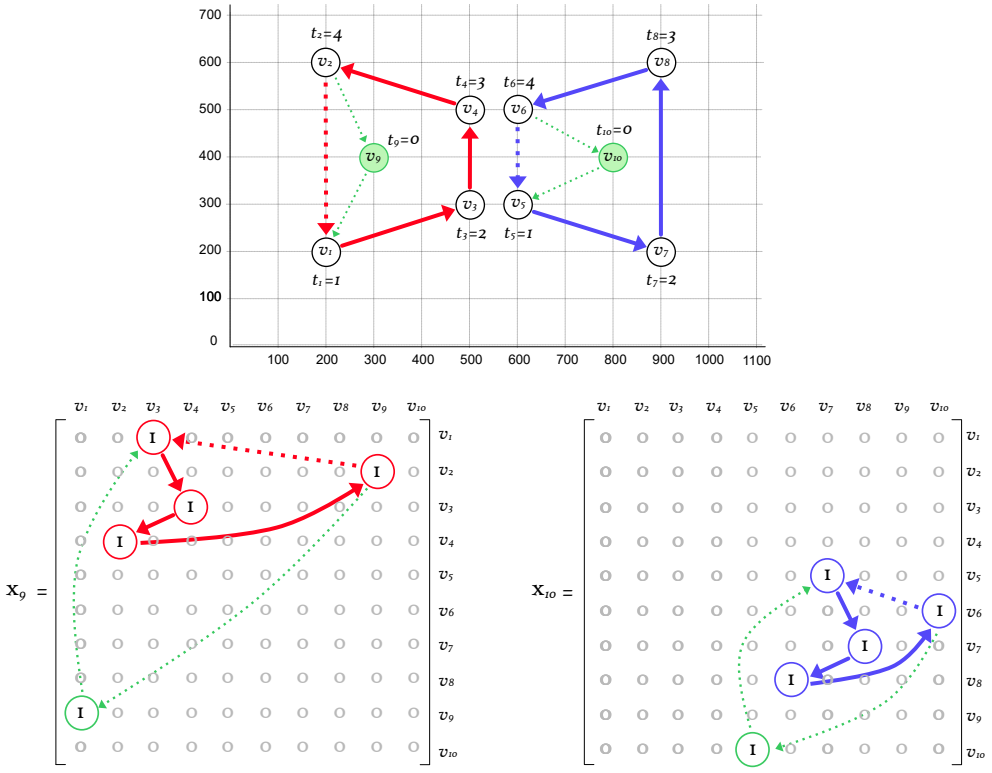


Figure 4.1: Exact solution of the IP for minsum CP-DFmTSP with bounding constraints. In this graph instance, $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, $m = 2$, $L = 3$, $U = 5$, $D = \{v_9, v_{10}\}$, and the cost of each edge equals the euclidian distance between its vertices (except edges with some dummy depot). There is a 10×10 matrix for each dummy depot, $\mathbf{x}_9 = [x_{i,j}]_{10\times10}$ and $\mathbf{x}_{10} = [x_{i,j}]_{10\times10}$.

Although the Objective function (4.1) has a quadratic term, it can be linearized by noting that, in the second term of Expression (4.14), $c_{i,j}$ is added up if and only if $x_{i,k,k}x_{k,j,k} = 1$. So, this product can be replaced by the binary variable $y_{i,j,k}$ (See Constraints (4.16)-(4.19).) By doing so, the minsum objective function is Eq. (4.15) and we get an integer linear program (ILP) with $\mathcal{O}(n^2m)$ binary variables and constraints.

$$\sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} \sum_{v_k \in D} (x_{i,j,k} + y_{i,j,k}) \tag{4.15}$$

where,

$$y_{i,j,k} \geqslant x_{i,k,k} + x_{k,j,k} - 1 \qquad \forall v_i, v_j \in V, \forall v_k \in D \tag{4.16}$$

$$y_{i,j,k} \leqslant x_{k,j,k} \qquad \forall v_i, v_j \in V, \forall v_k \in D \tag{4.17}$$

$$y_{i,j,k} \leqslant x_{i,k,k} \qquad \forall v_i, v_j \in V, \forall v_k \in D \tag{4.18}$$

$$y_{i,j,k} \in \{0,1\} \qquad \forall v_i, v_j \in V, \forall v_k \in D \tag{4.19}$$

The IP for minsum CP-DFmTSP can be adapted for the minmax objective function if we replace Expression (4.1) by Expressions (4.20)-(4.21), where P_{max} is the longest path among the salespersons.

$$\min \quad P_{max} \tag{4.20}$$

$$P_{max} \geqslant \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} \left(x_{i,j,k} + x_{i,k,k} x_{k,j,k} \right) \qquad \forall v_k \in D \tag{4.21}$$

Lemma 2. *Expressions (4.20) and (4.21) are the minmax objective function for CP-DFmTSP.*

Proof. Expression (4.21) comes from the following expanded form:

$$P_{max} \geqslant \underbrace{\sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,j,k}}_{\text{first term}} + \underbrace{\sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,k,k} x_{k,j,k}}_{\text{second term}} \qquad \forall v_k \in D \tag{4.22}$$

Let $v_k \in D$ be a specific dummy depot and let $v_i, v_j \in V$ be a pair of vertices in its respective path. Then, the first term of Expression (4.22) adds up the cost of the edges in the path of the dummy depot v_k, and the second term adds the cost $c_{i,j}$ of the edge (v_i, v_j) such that the salesperson goes from v_i to v_k and then from v_k to v_j. Thus, by considering the edge (v_i, v_j) the path is closed; the cost of this edge is considered in the objective function because $c_{i,j} x_{i,k,k} x_{k,j,k} = c_{i,j}$. Note that Expression (4.22) computes the cost of the closed path per each dummy depot $v_k \in D$. Besides, variable P_{max} must be greater or equal to each salesperson's path's cost and must be minimized. So, Expressions (4.20) and (4.21) are the minmax objective function for CP-DFmTSP. □

As with minsum, the minmax objective function defined by Expressions (4.20) and (4.21) can be linearized by adding variables $y_{i,j,k} = x_{i,k,k} x_{k,j,k}$ and Constraints (4.16)-(4.18).

In this thesis, a singleton path (v_i) of length 0 is invalid. However, a two-vertices path (v_i, v_j) is a valid closed path of length $c_{i,j} + c_{j,i}$ or an open path of length $c_{i,j}$.

Therefore, a solution to any variant of DFmTSP must consist of m paths, each with at least two vertices. Of course, the number of vertices in each path must also be between the bounding constraints L and U. Lemmas 3 and 4 show that the proposed IPs are consistent with these considerations.

Lemma 3. *Constraints (4.9) guarantee that* $t_b = t_a + 1$ *if and only if* $x_{a,b,k} = 1$ *for some dummy depot* v_k *[48].*

Proof. First, let us consider the case of a pair of vertices, v_a and v_b, such that $x_{a,b,k} = x_{b,a,k} = 0$ for every dummy depot v_k. In this scenario, constraints (4.9) are reduced to the single constraint $t_a - t_b \leqslant U - 1$, which is basically deactivated. Next, let us consider the case of a pair of vertices v_a and v_b such that $x_{a,b,k} = 1$ and $x_{b,a,k} = 0$ for some dummy depot v_k. In this scenario, constraints (4.9) become two constraints, $t_j \geqslant t_i + 1$ (with $i = a$ and $j = b$) and $t_i \leqslant t_j + 1$ (with $i = b$ and $j = a$). Thus, $t_a + 1 \leqslant t_b \leqslant t_a + 1$, i.e., $t_b = t_a + 1$. □

Lemma 4. *A solution to the proposed IPs consists of* m *paths, each having between* $\max\{2, L\}$ *and* U *vertices.*

Proof. By constraints (4.8), $\forall v_i \in V$, $\sum_{v_k \in D} x_{k,i,k} = \sum_{v_k \in D} x_{i,k,k} = 1$ is not allowed; this case alone avoids singleton paths to occur. Now, let us inspect the three remaining cases; the consequent of each implication follows from constraints (4.6) and (4.7) combined:

(i) If $\sum_{v_k \in D} x_{k,i,k} = \sum_{v_k \in D} x_{i,k,k} = 0$, then $2 \leqslant t_i \leqslant U - 1$.

- This case corresponds to the vertices non-adjacent to any dummy depot in any path. In Figure 4.1, these are v_3, v_4, v_7, and v_8.

(ii) If $\sum_{v_k \in D} x_{k,i,k} = 1$ and $\sum_{v_k \in D} x_{i,k,k} = 0$, then $t_i = 1$.

- This case corresponds to the first vertex visited by each salesperson after leaving its dummy depot. In Figure 4.1, these are v_1 and v_3.

(iii) If $\sum_{v_k \in D} x_{k,i,k} = 0$ and $\sum_{v_k \in D} x_{i,k,k} = 1$, then $L \leqslant t_i \leqslant U$.

- This case corresponds to the last vertex visited by each salesperson before returning to its dummy depot. In Figure 4.1, these are v_2 and v_6.

By Lemma 3, $t_j = t_i + 1$ if and only if $x_{i,j} = 1$. In other words, the variables t_i and t_j of adjacent vertices v_i and v_j in a path must differ by exactly one unit. Thanks to

this, constraints (4.6) and (4.7) guarantee that each path has between $\max\{2, L\}$ and U vertices. Finally, since there cannot be empty paths, a solution must have exactly m paths. □

So far, we have introduced and explained the main elements of IPs for CP-DFmTSP with bounding constraints and minsum and minmax objective functions. Next, we adapt this formulation to the OP variant. To our knowledge, this is the first reported mathematical formulation for this variant. Fortunately, all we have to modify is the objective function. The minsum objective function for the OP variant is:

$$\sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} \left(\sum_{v_k \in D} x_{i,j,k} \right) \tag{4.23}$$

Lemma 5. *Expression (4.23) is the minsum objective function for OP-DFmTSP.*

Proof. Expression (4.23) is the first term of expression (4.14). It considers only the cost of the traveled edges (solid red and blue edges in Figure 4.1). The vertices v_i and v_j that makes $x_{i,k,k} = 1$ and $x_{k,j,k} = 1$ are in the extremes of the open path; thus, $c_{i,j}$ needs not to be considered. Finally, the Objective function (4.23) considers the sum of the path's length for all salespersons. Therefore, it is the minsum objective function for OP-DFmTSP. □

Next, we show the minmax objective function for the OP variant.

$$\min \quad P_{max} \tag{4.24}$$

$$P_{max} \geqslant \sum_{v_i \in V} \sum_{v_j \in V} c_{i,j} x_{i,j,k} \qquad \forall v_k \in D \tag{4.25}$$

Lemma 6. *Expressions (4.24) and (4.25) are the minmax objective function for OP-DFmTSP.*

Proof. Expression (4.25) is the first term of Expression (4.22). It only considers the cost of the traveled edges. So, the paths are open. Besides, variable P_{max} is greater or equal to each salesperson's open path length, and P_{max} is minimized. Therefore, Expressions (4.24) and (4.25) are the minmax objective function for OP-DFmTSP. □

By Lemmas 2-4, the proposed IPs are valid formulations for DFmTSP and its main variants: OP, CP, minsum, minmax, and bounding constraints. To finish this section, notice that all these IPs can be adapted to FD-MmTSP by adding Constraints (4.26); these force each dummy depot to directly visit a vertex $v \in R$, where R is the input set of depots, $V \cap R = R$, $V \cap D = \emptyset$, and $|R| = m$. Of course, this adaptation is redundant

because, in the first place, the proposed IPs extend an IP for FD-MmTSP. However, this adaption becomes useful when $|R| < m$; in this scenario, we deal with a combination between FD-MmTSP and DFmTSP where less than m depots are known, but the solution still consists of m paths. For further illustration, Figure 4.2 shows the optimal solution for Mexicos city's town halls graph (cdmx16). We implemented and executed all the proposed formulations to compute the optimal solutions using off-the-shelf optimization software [70].

$$\sum_{v_k \in D} \sum_{v_i \in R} x_{k,i,k} = |R| \tag{4.26}$$

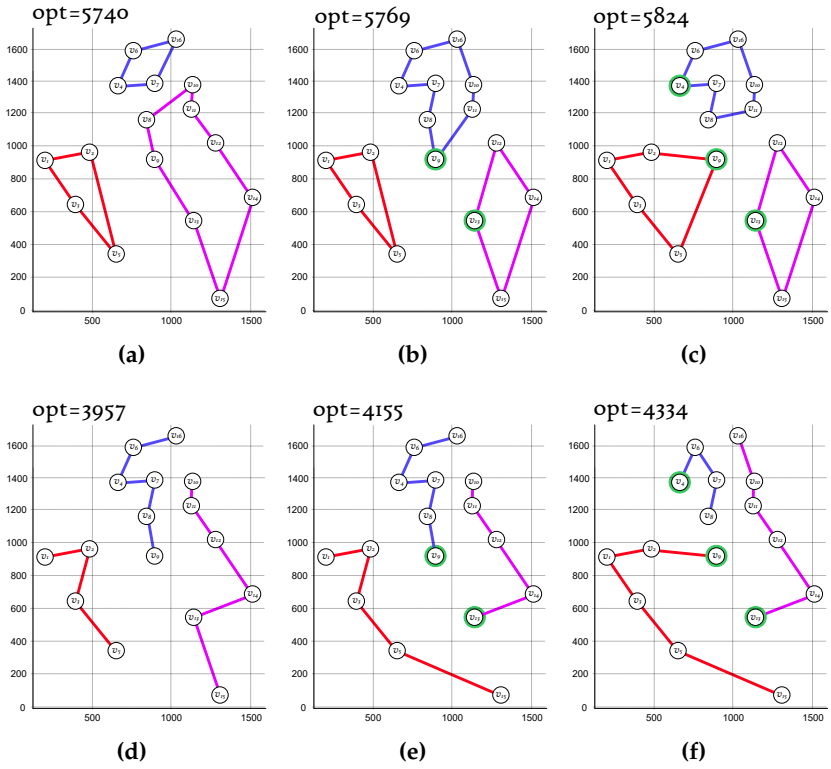


Figure 4.2: Optimal solutions for (**a**) CP-DFmTSP, (**b**) a combination between CP-DFmTSP and CP-FD-MmTSP ($R = \{v_9, v_{13}\}$), and (**c**) CP-FD-MmTSP ($R = \{v_9, v_{13}, v_4\}$). The objective function is minsum, the number of salespersons is three ($m = 3$), $L = 4$, $U = 10$, the cost of each edge equals the euclidian distance between its vertices, and the depots are marked in green. Subfigures (**d**), (**e**), and (**f**) correspond to the open-paths (OP) variants.

The following section introduces more compact formulations for the same problems and most of their variants.

4.3 Model 2: more compact integer programs

This section introduces more compact IPs for DFmTSP and some of its main variants. The main IP of this section has $\mathcal{O}(n^2)$ binary variables, $\mathcal{O}(n^2)$ constraints, and require m dummy depots. Expressions (4.27)-(4.37) define an IP for minsum CP-DFmTSP; we will use this IP as the basis for the rest of this section's IPs. For clarity, let us assume that the vertices in V are labeled as $\{v_1, v_2, ..., v_n\}$, the m dummy depots in D are labeled as $\{v_{n+1}, v_{n+2}, ..., v_{n+m}\}$, and $V' = V \cup D$, where $G = (V, E)$ is the input graph. Notice that $V \cap D = \emptyset$.

$$\min \quad \sum_{v_i \in V'} \sum_{v_j \in V'} c_{i,j} \Big(x_{i,j} + x_{n+m,j} x_{i,n+1} + \sum_{v_k \in D \setminus \{v_{n+m}\}} x_{k,j} x_{i,k+1} \Big) \tag{4.27}$$

$$\text{s.t.} \quad \sum_{v_j \in V' \setminus \{v_i\}} x_{i,j} = 1 \qquad \forall v_i \in V' \tag{4.28}$$

$$\sum_{v_i \in V' \setminus \{v_j\}} x_{i,j} = 1 \qquad \forall v_j \in V' \tag{4.29}$$

$$t_{n+1} = 0 \tag{4.30}$$

$$t_{k+1} - t_k \geqslant 3 \qquad \forall v_k \in D \setminus \{v_{n+m}\} \tag{4.31}$$

$$|V'| - t_{n+m} \geqslant 3 \tag{4.32}$$

$$1 \leqslant t_i \leqslant |V'| - 1 \qquad \forall v_i \in V' \setminus \{v_{n+1}\} \tag{4.33}$$

$$t_i - t_j + x_{i,j}|V'| \leqslant |V'| - 1 \qquad \forall v_i, v_j \in V' \setminus \{v_{n+1}\} \tag{4.34}$$

$$x_{i,j} \in \{0, 1\} \qquad \forall v_i, v_j \in V' \tag{4.35}$$

where

$$x_{i,j} = \begin{cases} 1, & \text{if a salesperson travels from } v_i \text{ to } v_j \\ 0, & \text{otherwise} \end{cases} \tag{4.36}$$

$$t_i = \text{time at which vertex } v_i \text{ is visited in the path} \tag{4.37}$$

The Objective function (4.27) has a quadratic term. Therefore, this formulation is an IQP. Later, we will show how to linearize the objective function; but before, let us explain this formulation. The solution to this formulation is a closed path that visits all vertices once, similar to TSP. From this single path, all m paths are inferred. Constraints (4.28) and (4.29) are flow constraints; they guarantee that there is a single

closed path that visits all vertices once. Constraints (4.30)-(4.32) are depot ordering constraints; they have two goals, to avoid singleton paths and to force to visit the dummy depots in order, i.e, $t_{n+1} < t_{n+2} < \cdots < t_{n+m}$. Constraints (4.33) and (4.34) are the classical Miller-Tucker-Zemlin SECs [**?**]. Expressions (4.35)-(4.37) define the decision variables. The Objective function (4.27) is minsum (See Lemma 7.) Notice that this IP has only one bounding constraint, i.e., each path must have more than two vertices. However, to extend this IP to the more general case, we can add Constraints (4.38)-(4.42) and remove Constraints (4.31) and (4.32). In this manner, each path must have between L and U vertices.

$$t_{k+1} - t_k \leqslant U + 1 \qquad \forall v_k \in D \setminus \{v_{n+m}\} \tag{4.38}$$

$$t_{k+1} - t_k \geqslant L + 1 \qquad \forall v_k \in D \setminus \{v_{n+m}\} \tag{4.39}$$

$$|V'| - t_{n+m} \leqslant U + 1 \tag{4.40}$$

$$|V'| - t_{n+m} \geqslant L + 1 \tag{4.41}$$

$$2 \leqslant L \leqslant \lceil |V|/m \rceil \leqslant U \leqslant |V| \tag{4.42}$$

Figure 4.3 shows the optimal solution of this IP over a small graph with $m = 2$, $L = 3$, and $U = 5$. In this figure, two paths are codified into one path that departs from dummy depot v_9, travels through every other vertex, including dummy depot v_{10}, and returns to v_9. Since traveling through a dummy depot does not incur any cost, we must identify the edges (v_1, v_2) and (v_8, v_7) and add their cost to the objective function. This is explained in Lemma 7. Notice that one matrix is enough to codify all m paths.

Lemma 7. *Expression (4.27) is the minsum objective function for CP-DFmTSP.*

Proof. Expression (4.27) comes from the following expanded form:

$$\underbrace{\sum_{v_i \in V'} \sum_{v_j \in V'} c_{i,j} x_{i,j}}_{\text{first term}} + \underbrace{\sum_{v_i \in V'} \sum_{v_j \in V'} c_{i,j} \left(x_{n+m,j} x_{i,n+1} + \sum_{v_k \in D \setminus \{v_{n+m}\}} x_{k,j} x_{i,k+1} \right)}_{\text{second term}} \tag{4.43}$$

The first term of Expression (4.43) adds up the cost of the traveled edges (solid red and blue edges in Figure 4.3). The second term adds up the cost $c_{i,j}$ of the edge of each pair of vertices $v_i, v_j \in V$ adjacent to a pair of consecutive dummy depots (dotted red and blue edges in Figure 4.3). Let v_k and v_{k+1} be a pair of consecutive dummy depots in $D \setminus \{v_{n+m}\}$ and let v_i and v_j be any pair of different vertices in

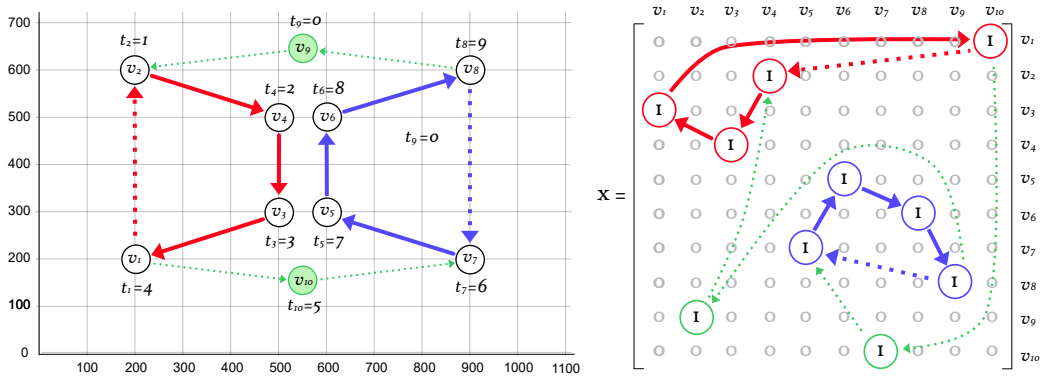


Figure 4.3: Exact solution of the IP for minsum CP-DFmTSP with bounding constraints. In this graph instance, $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, $m = 2$, $L = 3$, $U = 5$, $D = \{v_9, v_{10}\}$, and the cost of each edge equals the euclidian distance between its vertices (except edges with some dummy depot). There is only one matrix, $\mathbf{x} = [x_{i,j}]_{10\times 10}$.

V. If $x_{k,j} = 1$ and $x_{i,k+1} = 1$, then the salesperson associated with the v_k dummy depot goes from vertex v_i to v_j (notice that v_{n+1} is the consecutive dummy depot of v_{n+m}.) Thus, the path is closed if we consider the edge (v_i, v_j). Thanks to the flow Constraints (4.28)-(4.29), and the depot ordering Constraints (4.30)-(4.32), the inner sum $x_{n+m,j}x_{i,n+1} + \sum_{v_k \in D\setminus\{v_{n+m}\}} x_{k,j}x_{i,k+1}$ can only take values in $\{0, 1\}$. Therefore, Expression (4.27) is the minsum objective function for CP-DFmTSP. □

Although the Objective function (4.27) has a quadratic term, it can be linearized by noting that, in the second term of Expression (4.43), $c_{i,j}$ is added up if and only if there is a pair of consecutive dummy depots, v_k and v_{k+1}, such that $x_{k,j}x_{i,k+1} = 1$. So, this product can be replaced by the binary variable $y_{i,j}$ (See Constraints (4.45)-(4.49)). By doing so, the minsum objective function is Eq. (4.44) and we get an ILP with $\mathcal{O}(n^2)$ binary variables and $\mathcal{O}(n^2 m)$ constraints. Notice that v_{n+1} is the consecutive dummy depot of v_{n+m}.

$$\sum_{v_i \in V'} \sum_{v_j \in V'} c_{i,j} \left(x_{i,j} + y_{i,j}\right) \tag{4.44}$$

where,

$$y_{i,j} \geqslant x_{k,j} + x_{i,k+1} - 1 \qquad \forall v_i, v_j \in V, \forall v_k \in D \setminus \{v_{n+m}\} \tag{4.45}$$

$$y_{i,j} \geqslant x_{n+m,j} + x_{i,n+1} - 1 \qquad \forall v_i, v_j \in V \tag{4.46}$$

$$y_{i,j} \leqslant \sum_{v_k \in D} x_{i,k} \qquad \forall v_i, v_j \in V \tag{4.47}$$

$$y_{i,j} \leqslant \sum_{v_k \in D} x_{k,j} \qquad \forall v_i, v_j \in V \tag{4.48}$$

$$y_{i,j} \in \{0,1\} \qquad \forall v_i, v_j \in V \tag{4.49}$$

Adapting these IPs to the OP variant is straightforward. We only have to replace the Objective function (4.27) by (4.50), and omit variables $y_{i,j}$ with their respective constraints. This way, the dotted lines from Figure 4.3 are not considered.

$$\sum_{v_i \in V'} \sum_{v_j \in V'} c_{i,j} x_{i,j} \tag{4.50}$$

To finish this section, notice that this section's IPs can be adapted to FD-MmTSP with CP, OP, minsum, and bounding constraints. We must include Constraints (4.51), which force each dummy depot to go directly to one depot from the input set R of actual depots. Figure 4.4 serves as an example for $R = \{v_2, v_4\}$. As the previous section's IPs, this adaptation is advantageous too when $|R| < m$ (See Figure 4.2.)

$$\sum_{v_k \in D} x_{k,i} = 1 \qquad \forall v_i \in R \tag{4.51}$$

4.4 Analysis and empirical evaluation

Models 1 and 2 consist of a series of novel IPs for DFmTSP. These include IQPs and ILPs for the main variants of the problem: CP, OP, with bounding constraints, and for the minsum and minmax objective function. Tables 4.1 and 4.2 summarize the main features and scope of the proposed IPs.

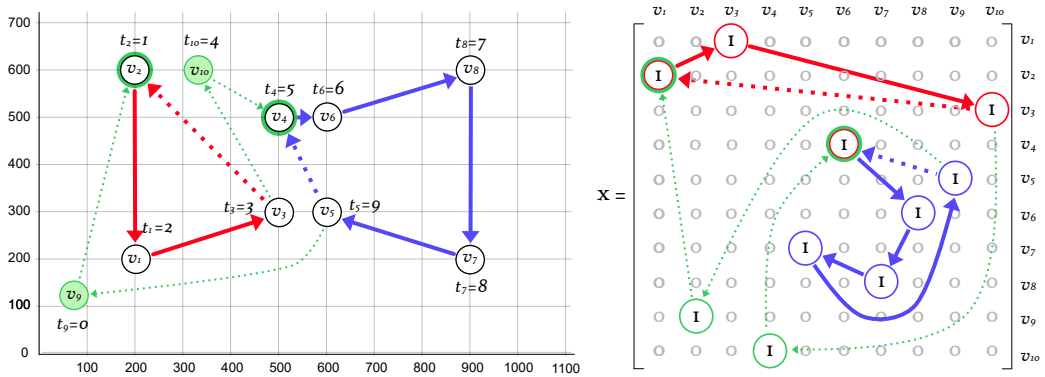


Figure 4.4: Exact solution of the IP for minsum CP-FD-MmTSP with bounding constraints (Section 4.3.) In this graph instance, $V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8\}$, $m = 2$, $L = 3$, $U = 5$, $R = \{v_2, v_4\}$, $D = \{v_9, v_{10}\}$, and the cost of each edge equals the euclidian distance between its vertices (except edges with some dummy depot). There is only one matrix, $\mathbf{x} = [x_{i,j}]_{10\times 10}$.

Table 4.1: IPs' variables and constraints.

IP	Model	objective function	binary variables	constraints
IQP1	Model 1	quadratic	$\mathcal{O}(n^2m)$	$\mathcal{O}(n^2)$
ILP1		linear	$\mathcal{O}(n^2m)$	$\mathcal{O}(n^2m)$
IQP2	Model 2	quadratic	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$
ILP2		linear	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2m)$

Table 4.2: IPs' features and scope.

IP	dummy depots	CP	OP	minsum	minmax	L	U	FD-M+DF
IQP1	m	✓		✓	✓	✓	✓	✓
ILP1	m	✓	✓	✓	✓	✓	✓	✓
IQP2	m	✓		✓		✓	✓	✓
ILP2	m	✓	✓	✓		✓	✓	✓

To test the empirical performance of the proposed IPs, we ran some experiments using off-the-shelf optimization software and some of the classical TSPLIB graph

instances [71]. From the literature, we could reproduce and validate the IP of Karabulut et al. [4]; so, we used this IP for comparison purposes. In all the experiments, we set the lower bounding constraint to two ($L = 2$) because the IP of Karabulut et al. is limited to such a value.

All the IPs were implemented using Gurobi 9.5.1 [70] through its Python interface. This off-the-shelf optimization software executes optimization techniques such as simplex, branch-and-bound, branch-and-cut, cutting planes, parallelism, and heuristics to find optimal solutions to mathematical formulations with linear or quadratic constraints. In all the experiments, we set Gurobi's Presolve parameter to 2. All the experiments were executed on a desktop Windows 11 Pro computer with an Intel Core i7-9700 processor. All the executions were set to a maximum of 12 GB of RAM and a 2 hours time limit.

Tables 4.3 and 4.4 show the results obtained by all the proposed IPs and the IP of Karabulut et al. [4] for the minsum CP-DFmTSP. We only experimented with this problem variant because it is the only one modeled by all the presented IPs, including the IP of Karabulut et al. In these tables, f stands for the reported value of the objective function, $t(s)$ for the time in seconds to find such objective function value, and g for the gap reported by the Gurobi software. A dash "-" character means no feasible solution was found within two hours. For convenience, we refer to the IPs from Section 4.2 with quadratic objective function as IQP1, and to the IPs from the same section with linear objective function as ILP1. Similarly, we refer to the IPs from Section 4.3 as IQP2 and ILP2.

Table 4.3 shows the results for the minsum CP-DFmTSP with tight bounding constraints, namely, the upper bound U is set to $\lceil n/m \rceil$. On the one hand, we can observe that only IQP1 and ILP2 could find feasible solutions for all the cases, suggesting they may be better suited for this specific variant of the problem. Besides, IQP1 found the best solutions in more cases than others. On the other hand, the IP of Karabulut et al. could not find feasible solutions in three cases within the two hours time limit. Figure 4.5 shows the convergence of the IPs reported by the Gurobi software for one of the used instances. From this figure, we can observe two facts. Firstly, from Figure 4.5a ($m = 3$), we can see that for some IPs it may not make sense to let them run for a long time since most of the IPs reach their best-found solution in much less time than the limit setup. Secondly, from Figure 4.5b ($m = 5$), we observe cases where the IPs may require more time to find feasible solutions. For instance, IQP1, IQP2, ILP1, and ILP2 needed 546s, 1565s, 2209s, and 3070s, respectively, to find the first feasible solution (the IP of Karabulut et al. did not find any feasible solution.)

These figures suggest that, as expected, the problems with a bigger value of m may be considerably challenging. Similar results were obtained for the other graph instances (See Figures 4.6 and 4.7.)

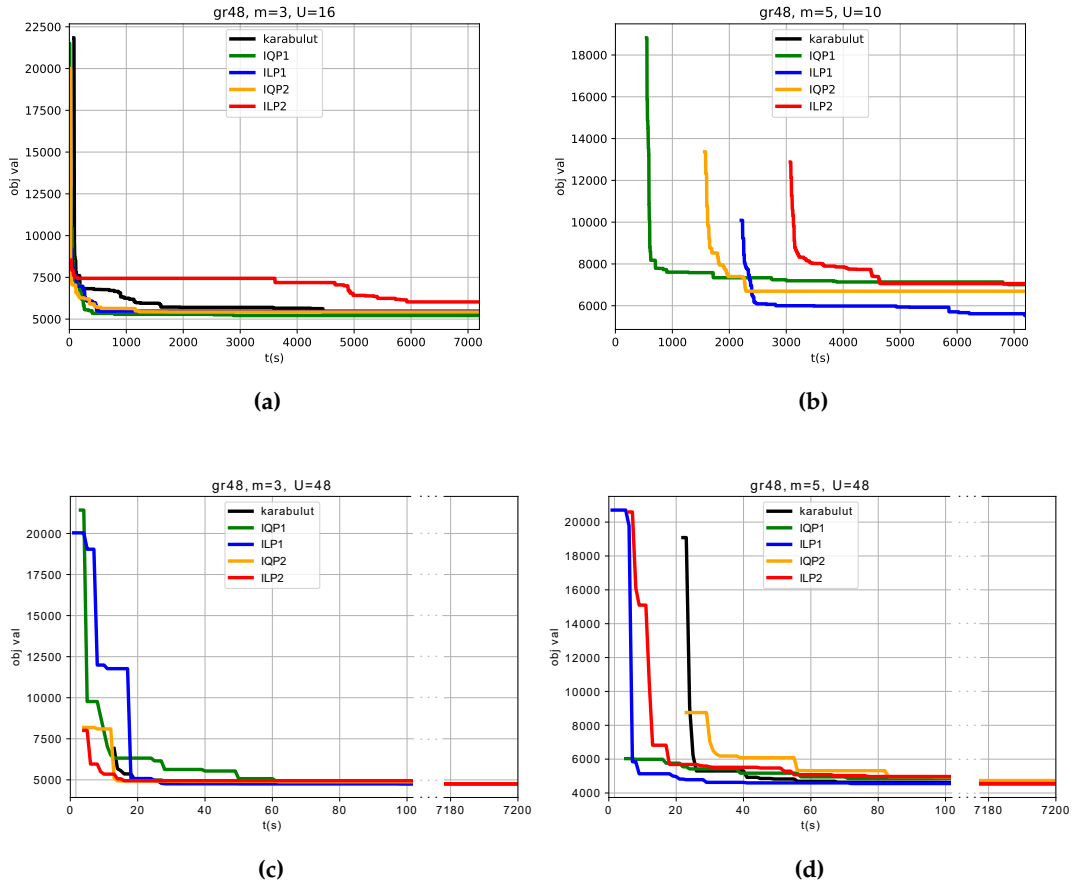


Figure 4.5: Convergence time reported by Gurobi for the instance gr48 for the minsum CP-DFmTSP with $L = 2$. Subfigures (**a**)-(**b**) correspond to tight bounding constraints, i.e., $U = \lceil n/m \rceil$. Subfigures (**c**)-(**d**) correspond to loose bounding constraints, i.e., $U = n$.

The experimentation reported in Table 4.4 is similar to that presented in Table 4.3. The only difference is that in Table 4.4, the upper bound constraint U is set to n. From this table, we observe that all of the IPs found feasible solutions for all of the cases, and the IP of Karabulut et al., IQP1, and IQP2 found the best solutions among all the IPs; even the optimality of some of them was proved because a value of $g = 0$ was reported. Furthermore, we can see that the reported gaps are considerably lower than those reported in Table 4.3. This suggests that CP-DFmTSP variant with bounding

constraints may be harder to solve when the number of vertices per salesperson is tight.

Figures 4.5c and 4.5d show the convergence time reported by Gurobi for the presented IPs for one of the used instances with loose bounding constraints. This figure shows that most IPs quickly found good-quality solutions, considering the relatively small reported gaps. This suggests that for this variant of the problem, setting long running times may not be necessary since the first few seconds (< 100s) are enough to get the most significant improvements in their found solutions. This supports the observation that tight bounding constraints make the instances harder to solve. Similar results were obtained for the other graph instances (See Figures 4.8 and 4.9.)

Table 4.3: IPs' comparison for minsum CP-DFmTSP with L = 2 and tight bounding constraints. The best-found solutions are bold.

instance	n	m	U	Karabulut et al. [4]			IQP1			ILP1			IQP1			ILP2		
				f	t(s)	g	f	t(s)	g	f	t(s)	g	f	t(s)	g	f	t(s)	g
dantzig42	42	3	14	748	320	18%	772	2914	24%	**739**	3934	21%	772	1125	24%	**739**	6823	21%
		5	9	**754**	5559	21%	787	1843	34%	786	2639	34%	879	4030	43%	1055	1763	52%
swiss42	42	3	14	1410	5682	17%	**1407**	1155	21%	1629	6377	33%	1430	1043	24%	1616	107	33%
		5	9	1397	886	19%	1603	4673	38%	**1394**	1741	26%	1470	1493	37%	1787	246	48%
att48	48	3	16	12557	7200	25%	**10941**	859	17%	12339	777	26%	12683	6329	31%	12432	668	28%
		5	10	-	-	-	**11149**	2272	27%	11505	1884	28%	11748	4602	33%	16160	7129	52%
gr48	48	3	16	5337	4452	13%	**5213**	2881	15%	5478	528	21%	5423	1724	23%	6030	5921	30%
		5	10	-	-	-	7009	6895	45%	5530	7184	30%	6696	2286	44%	7060	4639	47%
hk48	48	3	16	**11999**	1656	9%	12620	663	20%	12568	6956	19%	13576	643	26%	12456	4357	20%
		5	10	-	-	-	13546	5583	33%	-	-	-	-	-	-	15316	6864	42%

Table 4.4: IPs' comparison for minsum CP-DFmTSP with L = 2 and loose bounding constraints. The best-found solutions are bold.

instance	n	m	U	Karabulut et al. [4]			IQP1			ILP1			IQP2			ILP2		
				f	t(s)	g	f	t(s)	g	f	t(s)	g	f	t(s)	g	f	t(s)	g
dantzig42	42	3	42	**633**	74	0%	**633**	206	5.69%	**633**	91	7.11%	**633**	100	3.95%	**633**	40	3.79%
		5	42	**604**	84	0%	**604**	36	4.64%	**604**	38	14.07%	**604**	135	15.07%	605	83	13.22%
swiss42	42	3	42	**1208**	188	0%	**1208**	89	2.15%	**1208**	27	1.57%	**1208**	203	9.11%	**1208**	27	9.27%
		5	42	**1155**	50	1.47%	**1155**	413	5.11%	**1155**	812	5.63%	1167	2200	19.37%	**1155**	60	17.49%
att48	48	3	48	**9946**	442	4.25%	**9946**	11	5.81%	**9946**	85	4.83%	**9946**	51	7.57%	**9946**	26	7.51%
		5	48	**9448**	43	2.64%	**9448**	182	13.52%	**9448**	400	13.79%	**9448**	85	15.81%	**9448**	2000	14.73%
gr48	48	3	48	**4761**	201	1.70%	**4761**	96	3.91%	**4761**	27	6.87%	**4761**	1369	8.36%	**4761**	227	8.25%
		5	48	**4544**	113	0%	**4544**	207	8.78%	4558	71	7.79%	4735	171	18.59%	4558	2829	14.26%
hk48	48	3	48	**11101**	115	0%	**11101**	335	2.08%	**11101**	206	2.20%	11332	3037	10.98%	11134	1950	8.45%
		5	48	**10834**	164	0%	**10834**	919	6.98%	**10834**	786	7.30%	10888	1524	17.20%	10967	127	17.42%

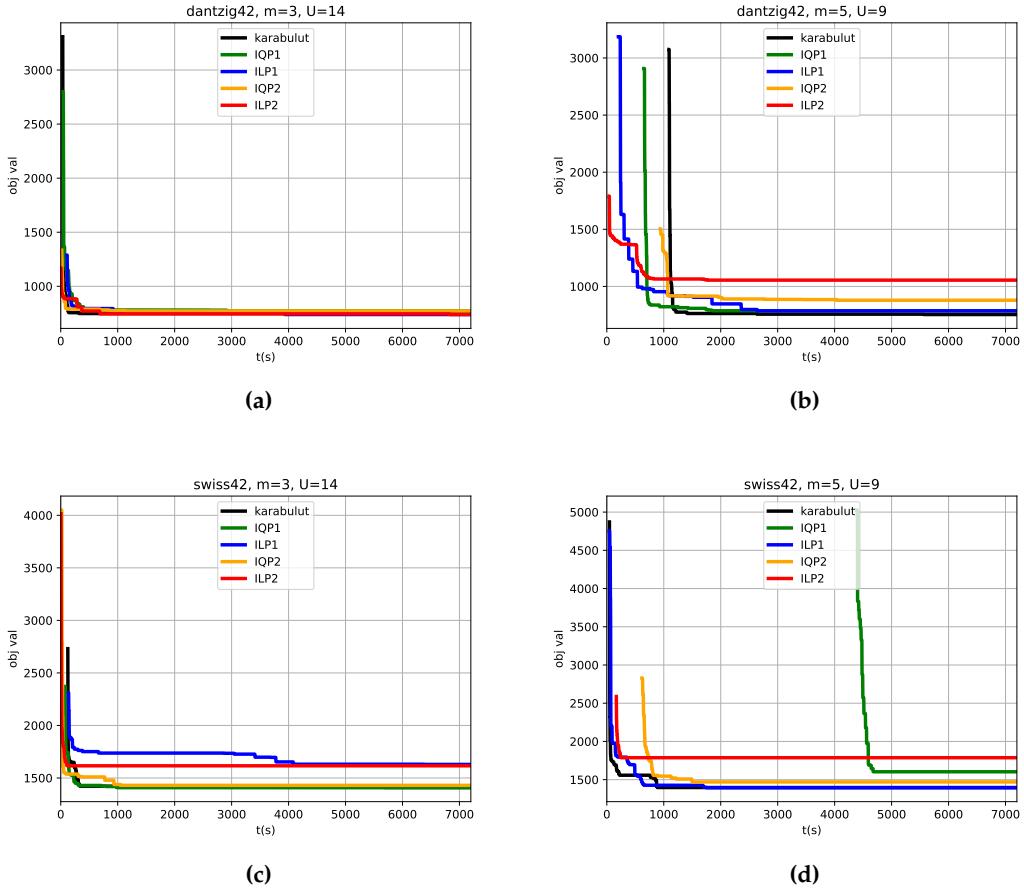


Figure 4.6: Convergence time reported by Gurobi for the minsum CP-DFmTSP with L = 2 and tight bounding constraints (Part 1).

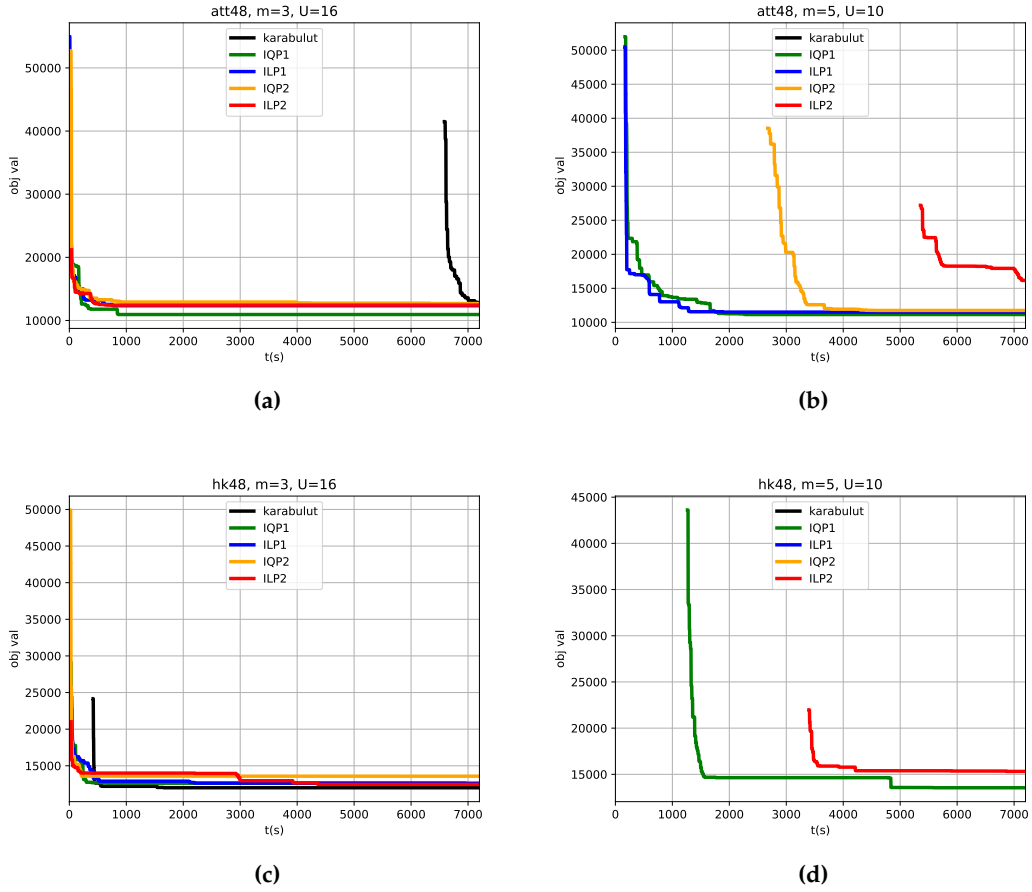


Figure 4.7: Convergence time reported by Gurobi for the minsum CP-DFmTSP with L = 2 and tight bounding constraints (Part 2).

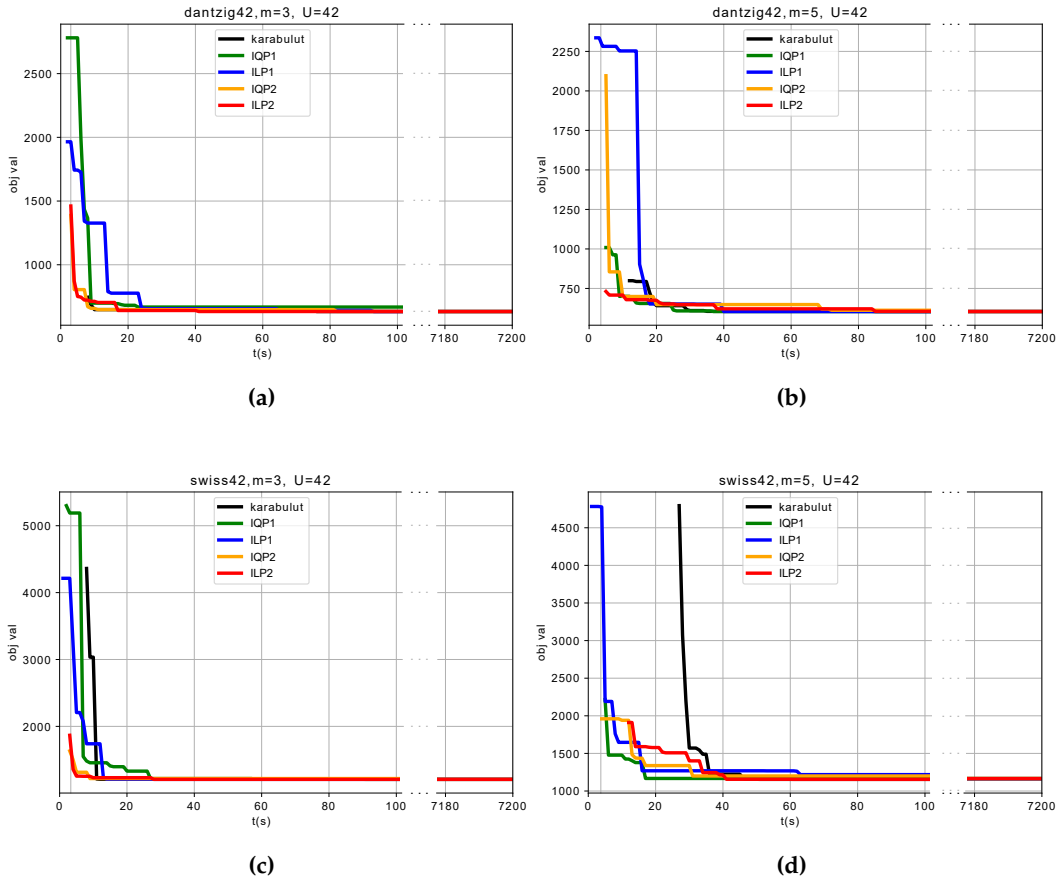


Figure 4.8: Convergence time reported by Gurobi for the minsum CP-DFmTSP with L = 2 and loose bounding constraints (Part 1).

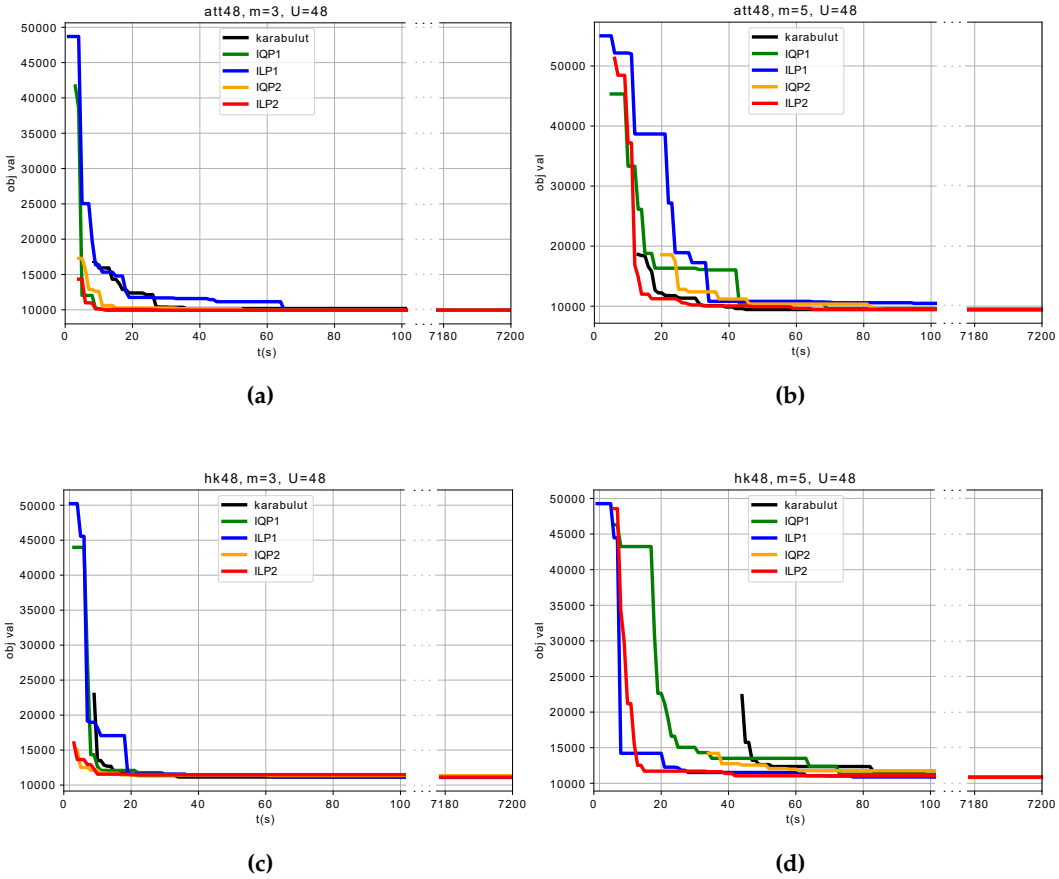


Figure 4.9: Convergence time reported by Gurobi for the minsum CP-DFmTSP with $L = 2$ and loose bounding constraints (Part 2).

Chapter 5

Heuristics and metaheuristics for DFmTSP

The first part of this chapter introduces a clustering-routing heuristic for DFmTSP with an upper bounding constraint. The second part presents a Memetic Algorithm (MA) with explicit diversity management for DFmTSP with lower and upper bounding constraints. Thanks to being a global-search technique, this metaheuristic outperformed the previous heuristics and the state-of-the-art metaheuristics for DFmTSP.

5.1 A Two-Phase Constructive Heuristic

This section is divided into four parts. Section 5.1.1 shows how CVKCP can be reduced to a series of MCDSPs. Section 5.1.2 introduces a heuristic for CVKCP modeled as a series of MCDSPs. Section 5.1.3 explains a heuristic for the routing phase of the clustering-routing constructive heuristic. Finally, Section 5.1.4 integrates the clustering and routing phases and tests the resulting heuristic over some de-facto benchmark datasets for routing problems.

5.1.1 Capacitated Vertex k-center as a series of simpler subproblems

In the paper [72], the relationship between the CVKCP and the MCDSP is explored. In a nutshell, if the size of the optimal solution (OPT) for CVKCP is known in advance, solving the CVKCP is equivalent to solving the MCDSP. Formally,

Theorem 5.1.1. *The minimum capacitated dominating set over the bottleneck graph* $G_{OPT} = (V, E_{OPT})$ *is the optimal solution to the CVKCP over the original input graph* $G = (V, E, w)$*, where* OPT *is the value of the optimal solution to the latter problem.*

The theorems' proofs in this section are not included to avoid repeated information. However, can be consulted at [72]. From an intuitive point of view, Theorem 5.1.1 relies on the fact that the bottleneck graph contains only the edges that suffice to compute an optimal solution. Theoretically, this reduction is not very useful because

the subproblems to solve are $\mathcal{NP}$-hard, too. However, from a practical point of view, it is usually easier to solve the MCDSP than the CVKCP. For completeness, let us show an ILP for CVKCP with known OPT as a MCDSP, where $G = (V, E, w)$ is the input weighted graph, $V = \{v_1, \ldots, v_n\}$, $f_c : V \to \mathbb{N}^+$ is the capacity function, and $a_{i,j} = 1$ if $w(\{v_i, v_j\}) \leqslant OPT$ and $v_i \neq v_j$; otherwise, $a_{i,j} = 0$. Namely, $a_{i,j}$ codifies the bottleneck graph.

$$\min \quad \sum_{i=1}^{n} y_i \tag{5.1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} a_{i,j} x_{i,j} = 1 - y_i \qquad \forall v_i \in V \tag{5.2}$$

$$\sum_{i=1}^{n} x_{i,j} \leqslant f_c(v_j) \qquad \forall v_j \in V \tag{5.3}$$

$$x_{i,j} \leqslant y_j \qquad \forall v_i, v_j \in V \tag{5.4}$$

$$x_{i,j}, y_i \in \{0, 1\} \qquad \forall v_i, v_j \in V \tag{5.5}$$

Variables y_i are used to decide which vertices are part of the MCDS, and variables $x_{i,j}$ decide if a vertex v_i is assigned to a *center* v_j. The Objective function (5.1) aims to minimize the MCDS's cardinality. Constraints (5.2) guarantee that each vertex outside of the MCDS is assigned to exactly one vertex into the MCDS. Constraints (5.3) are the capacity constraints. Finally, Constraints (5.4) guarantee that vertices are assigned only to vertices in the MCDS. Compared to the MILP for the CVKCP (See Expressions (3.48) to (3.54) at Section 3.3.2), this ILP is simpler.

Nevertheless, this reduction has a "problem": OPT must be known in advance. To solve this issue, a binary search is performed over the ordered set of possible sizes (See Algorithm 3). Algorithm 3 receives as input the complete weighted graph $G = (V, E, w)$, the capacity function f_c, and the ordered set of possible values of OPT. Then, it performs a binary search over such a list, solving the MCDSP over the bottleneck graph every time. The returned solution consists of a set of centers C and its corresponding partition P_C, where each vertex is assigned to exactly one center. At the end of the binary search, the last returned MCDS has $|C| \leqslant k$, and its corresponding partition consists of, at most, k clusters of vertices that respect the capacity function f_c.

The intuition behind the correctness of Algorithm 3 is that OPT will be eventually found by exploring all of its possible values; the binary search helps reduce the number of subproblems from $O(n^2)$ to $O(\log n)$, where $n = |V|$.

Input: A complete weighted graph $G = (V, E)$, a positive integer k, a capacity function $f_c : V \to \mathbb{Z}^+$, and an ordered list of the $|E|$ edge weights of G: $w(e_1), w(e_2), ..., w(e_{|E|})$ where $w(e_i) \leqslant w(e_{i+1})$

Output: A set of vertices $C \subseteq V$, $|C| \leqslant k$, and an assignment function $P_C : V \setminus C \to C$

1 $high \leftarrow |E|$
2 $low \leftarrow 1$
3 $(C, P_C) \leftarrow (\emptyset, \emptyset)$
4 **while** $low \leqslant high$ **do**
5 $mid \leftarrow \lfloor (high + low)/2 \rfloor$
6 $(C', P_{C'}) \leftarrow$ A minimum capacitated dominating set (D, P_D) over $G_{w(e_{mid})}$
7 **if** $|C'| \leqslant k$ **then**
8 $(C, P_C) \leftarrow (C', P_{C'})$
9 $high \leftarrow mid - 1$
10 **else**
11 $low \leftarrow mid + 1$
12 **end**
13 **end**
14 **return** (C, P_C)

Algorithm 3: The CVKCP as a series of MCDSPs.

5.1.2 k-center-based clustering

By Theorem 5.1.1 and Algorithm 3, CVCKP can be reduced to a series of MCDSPs. However, these subproblems are $\mathcal{NP}$-hard. Therefore, we must appeal to heuristics to achieve *good enough* feasible solutions in polynomial time. Algorithm 4 shows the proposed heuristic, which depends on *GreedyAssignment* (See Algorithm 5) and *DistanceBasedSelection* (See Algorithm 6).

The main cycle of OHFF (lines 4 to 15 of Algorithm 4) performs a binary search over the possible values of the size of the optimal solution. This way, at some point, a value $w(e_{mid})$ that is near-optimal will be explored. With such a value, the complete weighted input graph is pruned and turned into a simple graph. Thus, this algorithm seeks feasible capacitated dominating sets. Since the returned solution might have less than k vertices, the next cycles (lines 16 to 20 and 21 to 27) aim at completing it. Notice that the returned solutions consist of a set C of k vertices and a partition P_C (a clustering). The most important part of this algorithm is line 6, which performs a *GreedyAssignment* (Algorithm 5).

Input: A complete graph $G = (V, E)$, two positive integers k and U, and a non-decreasing list of the m edge weights of G, i.e., $w(e_1), w(e_2), ..., w(e_m)$, where $w(e_i) \leqslant w(e_{i+1})$

Output: A set of vertices $C \subseteq V$, such that $|C| = k$, and an assignment $P_C : V \setminus C \to C$

1 $high \leftarrow m$
2 $low \leftarrow 1$
3 $(C, P_C) \leftarrow (\emptyset, \emptyset)$
4 **while** $low \leqslant high$ **do**
5 $\quad mid \leftarrow \lfloor (high + low)/2 \rfloor$
6 $\quad (C', P_{C'}) \leftarrow$ *GreedyAssignment* $(G, k, w(e_{mid}), U)$
7 $\quad$ **if** $r(C', P_{C'}) \leqslant r(C, P_C)$ **then**
8 $\quad\quad (C, P_C) \leftarrow (C', P_{C'})$
9 $\quad$ **end**
10 $\quad$ **if** $r(C, P_C) \leqslant w(e_{mid})$ **then**
11 $\quad\quad high \leftarrow mid - 1$
12 $\quad$ **else**
13 $\quad\quad low \leftarrow mid + 1$
14 $\quad$ **end**
15 **end**
16 **while** $|C| < k$ **do**
17 $\quad v \leftarrow \arg\max\{d(u, P_C(u)) : u \in V \setminus C\}$
18 $\quad P_C \leftarrow P_C \setminus \{(v, P_C(v))\}$
19 $\quad C \leftarrow C \cup \{v\}$
20 **end**
21 **foreach** $c_i \in C$ **do**
22 $\quad X \leftarrow dom(P_{c_i}) \cup \{c_i\}$
23 $\quad c_j \leftarrow \arg\min\{\max\{d(u, v) : v \in X\} : u \in X\}$
24 $\quad P_C \leftarrow P_C \setminus P_{c_i}$
25 $\quad P_{c_j} \leftarrow \{(v, c_j) : X \setminus \{c_j\}\}$
26 $\quad P_C \leftarrow P_C \cup P_{c_j}$
27 **end**
28 **return** (C, P_C)

Algorithm 4: One-hop farthest-first (OHFF)

Input: A complete graph $G = (V, E, w)$, a positive integer k, a covering radius r, and a capacity U
Output: A set of vertices $C \subseteq V$, and an assignment $P_C : V \setminus C \to C$

$C \leftarrow \emptyset$
$P_C \leftarrow \emptyset$
$S \leftarrow copy(V)$
$G_r \leftarrow BottleneckGraph(G, r)$
foreach $v \in V$ **do**
  $score(v) \leftarrow |N_{G_r}(v)|$
end
for $i \leftarrow 1$ **to** k **and** $S \neq \emptyset$ **do**
  $v_f \leftarrow \arg\max\{d(v, C) : v \in S\}$
  if $\exists v \in N_{G_r}[v_f] \cap S \ : \ score(v) > U$ **then**
    $(c_i, f_{c_i}) \leftarrow DistanceBasedSelection(G, G_r, v_f, C, U, S)$
  else
    $c_i \leftarrow \arg\max\{score(u) : u \in N_{G_r}[v_f] \cap S\}$
    $P_{c_i} \leftarrow \{(u, c_i) : u \in N_{G_r}(c_i) \cap S\}$
  end
  $C \leftarrow C \cup \{c_i\}$
  $P_C \leftarrow P_C \cup P_{c_i}$
  foreach $v \in dom(P_{c_i}) \cup \{c_i\}$ **do**
    $S \leftarrow S \setminus \{v\}$
    foreach $u \in N_{G_r}(v)$ **do**
      $score(u) \leftarrow score(u) - 1$
    end
  end
end
while $S \neq \emptyset$ **do**
  $v \leftarrow$ any vertex in S
  $c_j \leftarrow \arg\min\{d(c_i, v) : c_i \in C \wedge |P_{c_i}| < U\}$
  $P_C \leftarrow P_C \cup \{(v, c_j)\}$
  $S \leftarrow S \setminus \{v\}$
end
return (C, f_C)

Algorithm 5: *GreedyAssignment*

Input: A complete graph $G = (V, E)$, a bottleneck graph $G_r = (V, E_r)$, a vertex $v_f \in V$, a set of centers C, a positive integer U, and the set of unvisited vertices S
Output: A center c' and an assignment $P_{c'} : V' \to \{c'\}$, where $V' \subseteq V \setminus C$

1 $(c', P_{c'}) \leftarrow null$
2 $d^* \leftarrow +\infty$
3 **foreach** $v \in N_{G_r}[v_f] \cap S \ : \ score(v) > U$ **do**
4 $\quad v_{ref} \leftarrow \arg\max\{d(u, C \cup \{v\}) : u \in S\}$
5 $\quad P_v \leftarrow \{(u, v) : u \in F_U\}$, where F_U is the set of the U farthest vertices in $N_{G_r}(v) \cap S$ to v_{ref}
6 $\quad u \leftarrow$ the $(U+1)^{th}$ farthest vertex in $N_{G_r}(v) \cap S$ to v_{ref}
7 $\quad$ **if** $d(v_{ref}, u) < d^*$ **then**
8 $\quad\quad d^* \leftarrow d(v_{ref}, u)$
9 $\quad\quad (c', P_{c'}) \leftarrow (v, P_v)$
10 $\quad$ **end**
11 **end**
12 **return** $(c', P_{c'})$

Algorithm 6: *DistanceBasedSelection*

The general idea of Algorithm 5 is to integrate different heuristic rules. First, each vertex is assigned a $score(v_i)$ (line 5), which reflects how many unassigned vertices are adjacent to v_i. Then, k vertices are selected iteratively (lines 7 to 23). A vertex can be selected if it is in the neighborhood of the farthest vertex from the partial solution ($N_{G_r}[v_f]$ at line 9). The intuition behind this heuristic is that the farthest vertex is critical, i.e., it is what defines the size of a feasible solution. Therefore, it must be assigned to some center as soon as possible. Afterward, the vertex with the higher score is added to the solution. However, if many vertices are capable of *covering* L or more vertices (line 9) *DistanceBasedSelection* (Algorithm 6) is executed. Finally, an extra heuristic consists of executing OHFF $|V(G)|$ times with a different initial vertex every time; we refer to this as OHFF+.

DistanceBasedSelection (Algorithm 6) explores a set of candidate vertices $\{v \in V(G_r) : v \in N_{G_r}[v_f] \wedge score(v) > L\}$. Since all of them are equally capable of covering L vertices, an extra heuristic must be implemented. In a nutshell, this extra heuristic seeks the vertex that minimizes the distance to the next farthest vertex. Thus, this procedure has to actually select each candidate vertex, compute the hypothetical farthest vertex (line 4), and then keep the vertex that minimizes the aforementioned distance (lines 6 to 10). Since the exploration of each vertex is independent of the others, this procedure was

parallelized using the multiple instruction multiple data model (MIMD). According to experimental results, this heuristic is very efficient and competitive when compared to state-of-the-art heuristics. In fact, it found near-optimal solutions for graph instances with up to 5000 vertices and $k = 40$ in less than 30 seconds [72]. by the way, the name of these heuristics come from the heuristic rules they implement: one-hop farthest-first. Namely, to explore the neighborhood (one-hop) of the farthest vertices.

Input: A weighted graph $G = (V, E, w)$, and two positive integers m and U
Output: A set of salespersons tours $p = \{p_1, p_2, \cdots, p_m\}$

```
1 p ← ∅
2 (C, P_C) ← k − center(G, m, U)
3 foreach c_i ∈ C do
4 |   X ← dom(P_{c_i}) ∪ {c_i}
5 |   p_i ← Farthest − Insertion(G[X])
6 |   p ← p ∪ {p_i}
7 end
8 return p_i
```

Algorithm 7: Two-Phase constructive heuristic.

5.1.3 Farthest-insertion for routing

In the routing literature, many algorithms have been proposed for the classical TSP. Among these proposals, there are some exact algorithms that guarantee to find the optimal solution. However, since TSP is $\mathcal{NP}$-hard, such algorithms may have an important limitation. Other proposals are approximation algorithms that do not guarantee finding an optimal solution but a solution inside a ratio of the optimal one. Likewise, heuristics do not guarantee optimality but are fast and very effective in finding near-optimal solutions. Metaheuristics are more elaborated procedures that use exploration and exploitation components to escape from local optimals during search. Among all these proposals, approximation algorithms and heuristics are preferred when running time is an important issue since both run in polynomial time.

Many approximation algorithms and heuristics have been proposed for routing problems. Among the route constructive procedures, there is a specific category called *insertion procedures*, which consists of constructing a route by iteratively choosing a vertex and inserting it in the route. Some classical heuristics that have been proposed are: nearest-insertion, cheapest-insertion, and farthest-insertion [73, 74]. From these, the farthest-insertion (a.k.a. furthest-point insertion) has shown to be more effective compared with the others. Besides, by using farthest-insertion, the TSP can be approx-

imated within an approximation factor of $\log n$ [75]. To the best of our knowledge, this bound has not been proven to be tight. The general idea of farthest-insertion is the following: the algorithm may start by choosing the vertices in the longest edge as the initial partial path. Then, the farthest vertex from the partial path is chosen and inserted inside the path in the position that increases the lowest possible value. The procedure is repeated until all the vertices are in the partial tour. This simple idea has proven practical, as solutions found by it are typically only 5% to 10% longer than optimal in some cases [76].

Input: A complete weighted graph $G = (V, E, w)$
Output: A salesperson tour p_i that contains all the vertices in $V(G)$
$p_i \leftarrow \max\arg\{w(e) : e \in E(G)\}$
while $V(G) \neq V(p_i)$ **do**
  $v \leftarrow \max\arg\{d(v, V(p_i)) : v \in V(G) \setminus V(p_i)\}$
  insert v in p_i at the position that adds the smallest cost
end
return p_i

Algorithm 8: farthest-insertion algorithm [73].

5.1.4 Computational experimentation and results

By using OHFF and OHFF+ at the clustering phase [77] and the farthest-insertion heuristic at the routing phase [73, 78], we were able to obtain a two-phase constructive procedure for the DFmTSP that considers upper bound constraints. We analyzed and compared this proposal against the state-of-the-art metaheuristic AC-PGA [1]. We tested the algorithms over some instances of TSPLIB benchmark [71]. Figures 5.1 and 5.2 show the obtained results over the instances kroA100 and kroA200, respectively, for different values of k. According to the experimentation performed, in some cases, our two-phase constructive proposal was able to obtain better solutions. At some times, the simpler version of the k-center-based heuristic (OHFF) was capable of getting better solutions than the AC-PGA metaheuristic. This supports the conjecture that using the cluster-first route-second approach for the DFmTSP can have practical advantages.

5.2 Metaheuristic algorithms for DFmTSP

This section introduces a metaheuristic algorithm for DFmTSPs with bounding constraints. Specifically, we perform a deep study of evolutionary algorithms and the behavior and importance of diversity of solutions during the solving process. We

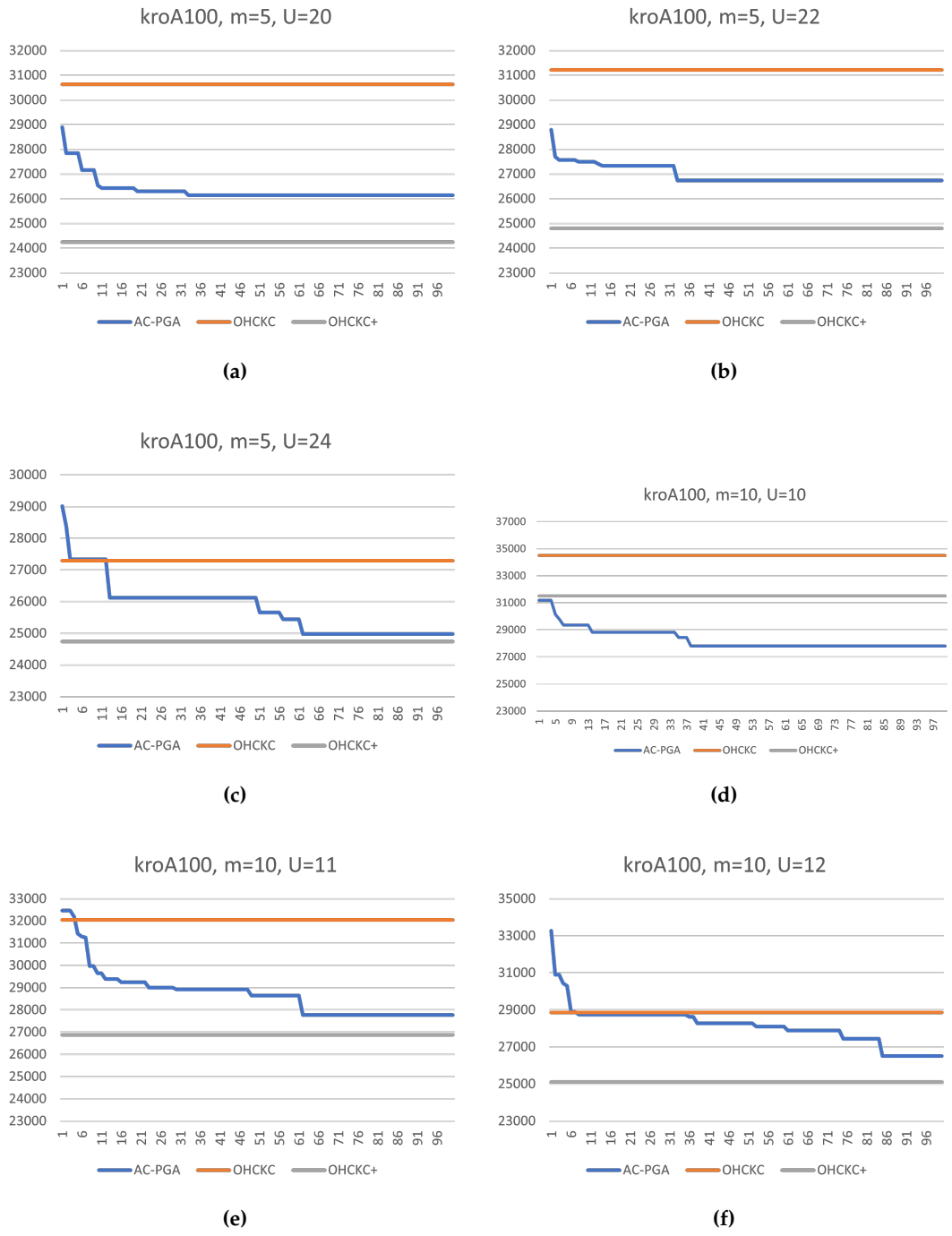


Figure 5.1: Convergence reported by algorithms OHFF, OHFF+ and AC-PGA for instance kroA100 for the minsum CP-DFmTSP.

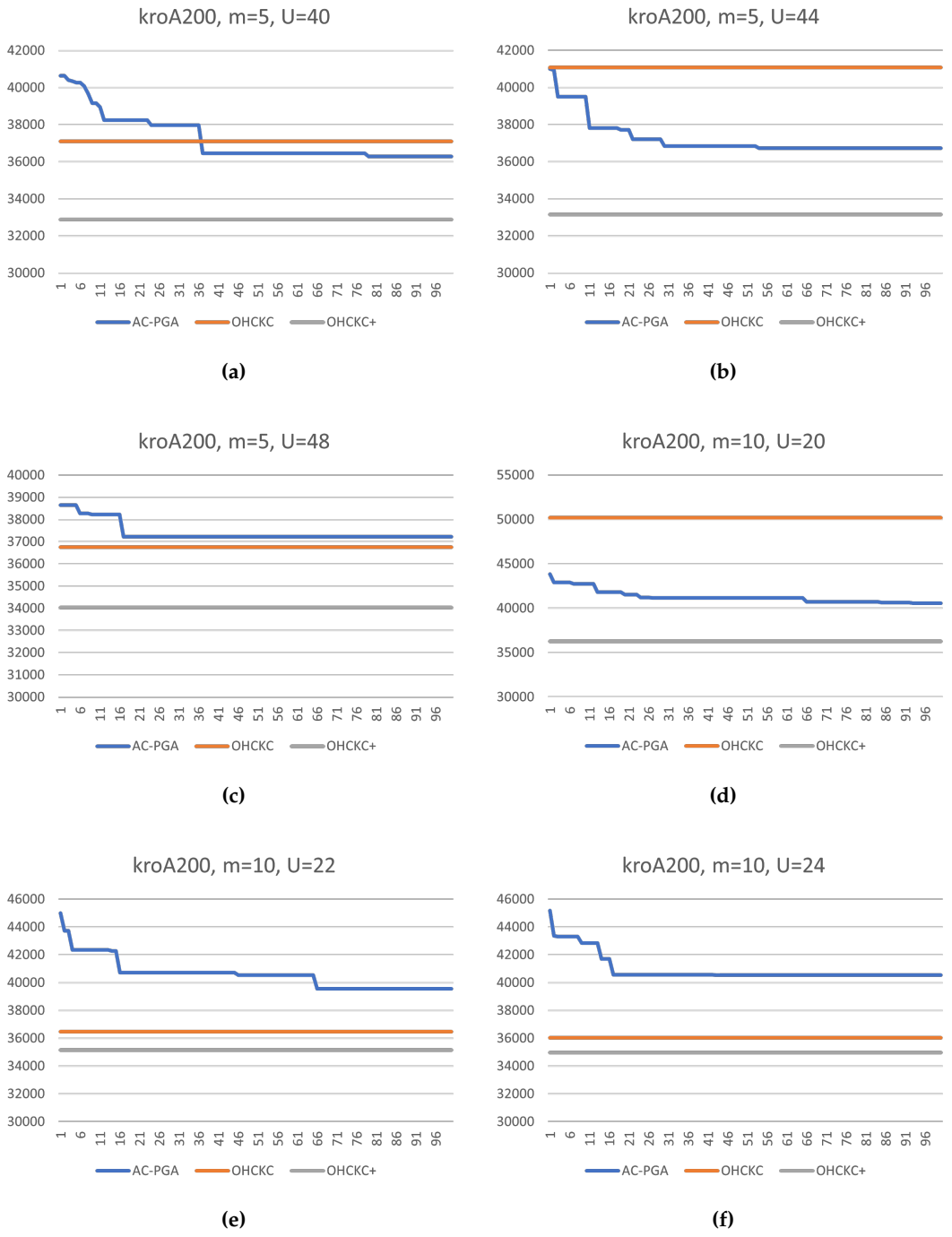


Figure 5.2: Convergence reported by algorithms OHFF, OHFF+ and AC-PGA for instance kroA200 for the minsum CP-DFmTSP.

stated the conjecture and the hypothesis that evolutionary computing algorithms for the DFmTSP can perform well if the diversity of the solutions is appropriately handled during the evolution process. For this purpose, we start the section by briefly introducing the concept of diversity in evolutionary algorithms and describing a state-of-the-art technique called Best Non-Penalized. Then, we introduce in detail the design of the proposed Memetic Algorithm (MA) for the DFmTSP. Finally, we performed an empirical evaluation of the proposed MA that proves the effectiveness of our proposal.

5.2.1 Diversity in evolutionary algorithms

Evolutionary algorithms are general problem-solving techniques usually used for search and optimization. These techniques are inspired by Darwin's evolution theory of species and natural selection. There are many types of evolutionary algorithms, such as Genetic/Memetic Algorithms, Genetic Programming, Evolutionary Programming, among others [79]. In general, evolutionary algorithms are part of a more extensive research field called metaheuristics, whose purpose is the study of algorithms for hard search/optimization problems that cannot be solved efficiently with conventional approaches. The study of evolutionary algorithms involves the design of mechanisms such as selection, crossover, mutation, and replacement. Such components are known as genetic operators, and their importance lies in the fact that they radically influence the effectiveness of evolutionary algorithms from a practical point of view. Besides, when designing evolutionary algorithms, it is essential to have a good balance of exploitation and exploration [80]. In this context, exploitation refers to improving and intensifying the candidate solutions, whereas exploration refers to how many solutions and in what way they are visited. Among the genetic operators of evolutionary algorithms, replacement is a crucial operator since it defines which candidate solutions survive for the next generation given the parents and offspring populations. In the literature, the classical replacement operators are generational, where offspring directly replace the parents' population, and truncation, which has a more significant selection pressure with the aim of a faster convergence. However, these replacement operators do not have proper diversity handling among the candidate solutions. Diversity handling techniques are usually used to keep a good balance of exploration during the evolution process. Maintaining proper diversity management in the evolution process may help to avoid early local optima because it tries to keep the diversity of the candidate solutions that are being explored and intensified. In the literature, proper diversity management has proven to be important in multi-objective and single-objective evolutionary algorithms [81], for continuous optimization [82], and

combinatorial optimization [83, 84].

Among the handling diversity techniques in the literature, there are two categories: implicit and explicit [84]. On the one hand, implicit techniques refer to evolutionary algorithms that work with structured populations, such as island-based and cellular schemes. Researchers have studied that these approaches have important implicit effects on diversity preservation, since at these models, the candidate solutions can only interact with other candidate solutions in a defined neighborhood in the structured population. This usually may delay the convergence. On the other hand, explicit techniques refer to explicitly controlling the diversity among the candidate solutions. Explicit diversity techniques have some advantages. We can properly control the balance of exploration and exploitation during the evolution process, as well as the decrease and increase of diversity.

One of the explicit diversity techniques that has proven effective is the the Best-Non-Penalized (BNP) [81, 84]. Overall, the BNP is the survivor replacement operator that aims to have proper diversity handling by choosing candidate solutions that are not close enough for the next generation. This is achieved by using a distance metric among the candidate solutions. At each generation, the BNP tries to select for the next generation, candidate solutions that are at least a certain distance away D, then D linearly decreases as the evolution process goes. In this way, the BNP ensures that at the beginning, no very close solutions are selected to survive. Thus, proper diversity is maintained and early local optima try to be avoided.

Algorithm 9 shows the pseudocode of the BNP. The notation is the following:

- $f(s)$ fitness value of a candidate solution s.
- Given a candidate solutions set P and a candidate solution $s \notin P$, $d(s, P)$ is the distance from s to its closest solution in P, i.e., $d(s, P) = \min\{d(s, k) : k \in P\}$.
- D_{init} is the initial expected diversity, and it can be initialized as the average diversity of the population.
- $state \in [0, 1]$ is the current state of the evolutionary algorithm running. It can be defined as $\frac{\text{elapsed time}}{\text{total time}}$ or $\frac{\text{current generation}}{\text{total generations}}$.

Given a population of parents P and the offspring population O, the BNP keeps the best candidate solution for the next generation. Then, it looks to select candidate solutions at a certain distance D, which linearly decreases as the evolutionary algorithm runs. In this way, the algorithm avoids selecting identical candidate solutions that may

cause the loss of diversity in the population. Therefore other issues may arise, like getting stuck in local optima.

Input: Parents population P, offspring population O, and $state$ of running
Output: New population P'

$A \leftarrow P \cup O$
$s \leftarrow$ best solution in A
$P' \leftarrow \{s\}$
$A \leftarrow A \setminus \{s\}$
$D \leftarrow D_{init} - D_{init} \cdot state$
while $|P'| \neq |P|$ **do**
  $s \leftarrow$any solution in A
  foreach $k \in A$ **do**
    if $d(s, P') < D$ **then**
      if $d(k, P') > d(s, P')$ **or** $\big(d(k, P') = d(s, P')$ **and** $f(k) < f(s)\big)$ **then**
        $s \leftarrow k$
      end
    else
      if $d(k, P') \geqslant D$ **then**
        if $f(k) < f(s)$ **or** $\big(f(k) = f(s)$ **and** $d(k, P') > d(s, P')\big)$ **then**
          $s \leftarrow k$
        end
      end
    end
  end
  $P' \leftarrow P' \cup \{s\}$
  $A \leftarrow A \setminus \{s\}$
end
return P'

Algorithm 9: Best Non-Penalized (BNP).

5.2.2 Memetic algorithm

Our proposal consists of a Memetic Algorithm (MA) with specifically designed components such as a crossover operator, a replacement operator, and an intensification phase that uses the Lin-Kernighan metaheuristic. One of the main novelties of our proposal is that the MA treats the DFmTSP just as an assignment problem, although it is a routing problem. That is, the MA looks only for the proper assignment of the

vertices to the routes without taking care of the order in which vertices are visited. Then, the intensification phase becomes relevant because it determines the order in which vertices should be visited in the paths. Thus, the evaluation of the quality of the paths is also determined in the intensification phase. We intend our presented results to be reproducible. Thus, we describe the details of the main components of our proposal, in addition to providing the source code of our implementations.

Now, we show the overall notation of the pseudocodes presented in the next subsections.

- $s = \{s_1, s_2, \cdots, s_m\}$ candidate solution composed by m salespersons paths.
- s_i a salesperson path (a sequence of vertices) of a candidate solution s.
- $V(s_i)$ set of vertices of the salesperson path s_i.
- s_i^p subpath (subsequence of vertices) of length p of a salesperson path s_i.
- $f(s)$ fitness value of a candidate solution s.
- $c(s_i)$ cost of a salesperson path s_i.

Algorithm 10 shows the general template of the proposed MA. The MA receives a feasible initial population randomly generated. In line 1, the received population is intensified and evaluated. This phase aims to define the order in which the vertices are traversed in the paths of the salespersons. The details of this intensification and evaluation phase is described later. In line 2, the best solution is saved as s^*. Then, while a stop condition is not met, the following actions take place:

First, given the population P, a selection operator is applied to choose the parents P' in line 4. The selection operators used are described later. In line 5, the offspring is created from the selected parents P'. Then, the offspring is created in line 5. Since we are working with bounding constraints, the offspring may violate such constraints. Thus, in line 6, a balancing method is applied to repair the offspring. Then, in line 7, the intensification and evaluation phase is executed again over the offspring population. Then, the survivor selection phase is executed in line 8. The BNP (Algorithm 9) is used for this purpose. Lines 9-12 keep track of the best found solution.

```
   Input: A population of candidate solutions P
   Output: Best found solution s*
1  IntensifyAndEvaluation(P)
2  s* ← best solution in P
3  while ¬StopCondition() do
4  |  P' ← SelectParents(P)
5  |  O ← Recombination(P')
6  |  Balancing(O)
7  |  IntensifyAndEvaluation(O)
8  |  P ← Replacement(P, O)
9  |  s ← best solution in P
10 |  if f(s) < f(s*) then
11 |  |  s* ← s
12 |  end
13 end
14 return s*
```

Algorithm 10: Memetic algorithm template.

5.2.3 Intensification phase with Lin-Kernighan

The intensification and evaluation phase (Algorithm 11) is an important component of our proposal. It uses the Lin-Kernighan heuristic and works as follows. Overall, this phase consists of two different ideas. On the one hand, the first idea is that segments of paths are interchanged among the salesperson paths. The motivation behind this idea is exploring solutions since new paths with different vertices are created. On the other hand, the second idea is about intensification/exploitation. The latter uses the Lin-Kernighan heuristic to improve the paths of the salespersons. Now, let us describe in more detail this intensification and evaluation phase.

First, the intensification is executed for a certain time t_{max}. This is ensured by the while loop of line 2. Then, the boolean function callLKH ensures that the runLKH (the Lin-Kernighan heuristic) is called exactly $n_{LKH} - 1$ times during the execution of the algorithm to intensify and evaluate the solution s. Note that one call of runLKH is performed after the while ends so that runLKH is called exactly n_{LKH} times. Then, the exploration phase starts. First, in line 6 a segment size p is selected. In lines 7 and 8 two salesperson paths of s are chosen randomly and copied. In lines 9-12, random subpaths are extracted from the selected paths and inserted into each other using the cheapest insertion heuristic. The cheapest insertion heuristic inserts a vertex in a path

in a specific position with the lowest cost increase. Then, in lines 13-15, if the cost of the new paths is improved, they become part of the solution s.

In summary, Algorithm 11 uses an exploration approach that interchanges segments of paths, and also, each certain time the paths are improved and intensified using the Lin-Kernighan heuristic. Note that $runLKH$ executes the Lin-Kernighan heuristic over each $s_i \in S$ for a running time t_{LKH}. Be aware that using the Lin-Kernighan heuristic may be heavy. Thus, a small value of $n_{LKH} \geqslant 2$ is recommended.

Input: A feasible solution s, time limit for intensification t_{max}, number of LKH calls to be applied to s during the intensification phase n_{LKH}, time limit for each LKH call t_{LKH}
Output: The intensified solution s

1 $t_{elap} \leftarrow$ elapsed time of intensification (initially 0)
2 **while** $t_{elap} < t_{max}$ **do**
3 **if** $callLKH(t_{elap}, t_{max})$ **then**
4 $runLKH(s, t_{LKH})$
5 **end**
6 $p \leftarrow$ choose segment size to be exchanged among the chosen paths
7 $s_i, s_j \leftarrow$ choose two different random paths in s
8 $k_i, k_j \leftarrow Copy(s_i), Copy(s_j)$
9 $k_i^p \leftarrow$ extract random subpath of length p from k_i
10 $k_j \leftarrow$ insert vertices in k_i^p to k_j by cheapest insertion
11 $k_j^p \leftarrow$ extract random subpath of length p from k_j
12 $k_i \leftarrow$ insert vertices in k_j^p to k_i by cheapest insertion
13 **if** $c(k_i) + c(k_j) < c(s_i) + c(s_j)$ **then**
14 $s_i, s_j \leftarrow k_i, k_j$
15 **end**
16 **end**
17 $runLKH(s, t_{LKH})$
18 **return** s

Algorithm 11: Intensify and Evaluation.

5.2.4 Genetic operators

The next important component of our proposal is the recombination operator. The recombination step consists of taking two parents and recombining them to create a new candidate solution that combines the features of the two parents. This is performed through a novel crossover operator used in the literature for other combinatorial

optimization problems that work with partitions. The operator is the Hungarian Based Crossover (HBX). This operator aims to maximize the number of vertices in the same partitions of both parents and offspring.

Algorithm 12 shows the pseudocode of the HBX. Now, let us describe it in more detail. First, line 1 starts by creating a complete weighted bipartite graph through Algorithm 13, where the vertices are the paths of the salespersons and the weights are the number of common vertices between such paths. Then, a maximum weighted matching M is obtained in line 2. This can be performed through the Hungarian algorithm that solves the assignment problem in polynomial time. The HBX uses the edges of M to create an offspring of the solutions s and k by prioritizing creating paths with vertices that are part of the same paths in the parents. The purpose is that paths of good quality are kept in the offspring. Each path of the offspring must be composed of exactly m intersections of the bipartite graph G. Besides, each edge of M can be used only once. Thus, to avoid repetitions, B (line 3) is used to track the used edges in the recombination process. S and K (lines 4-5) store eligible vertices that will be used to ensure that each offspring path will be composed of exact m intersections. o (line 6) is the created offspring from the recombination process. Then, each iteration of the for-loop of line 7 will select the vertices that compose the ith path of the offspring. Inside this for-loop, the following actions take place. At line 8 an edge in $\{s_i, k_j\} \in M$ (maximum weighted matching) is taken and removed from it, and in line 9 the ith path is initialized as the empty set. Then, lines 11-19 will be executed on even iterations, whereas lines 21-29 are executed in odd iterations. In even iterations, endpoint s_i is fixed, and all adjacent unused edges of s_i are used to compose the ith path (lines 12-15). Finally, if less than m edges were available to compose the ith path, lines 16-19 will ensure that the missing edges are considered to form the ith path. This is performed by taking endpoint k_j are considering all its unused adjacent edges to eligible vertices in S. In odd iterations (lines 21-29) run the same steps but fix endpoint k_j.

For better comprehension, Tables 5.1 and 5.2 show an example of two candidate solutions s and k and the weights of the complete bipartite weighted graph formed by such solutions. Figure 5.3 shows the maximum weighted matching in bold, and the execution of iterations of the HBX.

Input: Two candidate solutions s and k
Output: An offspring candidate solution o

$G \leftarrow$ Create bipartite graph with s and k `// Algorithm 13`
$M \leftarrow$ maximum weighted matching of G
$B \leftarrow \emptyset$ `// set of used edges`
$S \leftarrow \emptyset$ `// eligible paths of s`
$K \leftarrow \emptyset$ `// eligible paths of k`
$o \leftarrow \emptyset$ `// offspring solution`
for $i \leftarrow 1$ **to** m **do**
  $\{s_i, k_j\} \leftarrow$ extract and remove any edge in M
  $V(o_i) \leftarrow \emptyset$
  if $i \equiv 1 \mod 2$ **then**
    $K \leftarrow K \cup \{k_j\}$
    foreach $\{s_i, k_t\} \in E(G) \setminus B$ **do**
      $B \leftarrow B \cup \{\{s_i, k_t\}\}$
      $V(o_i) \leftarrow V(o_i) \cup V(s_i) \cap V(k_t)$
    end
    foreach $\{s_t, k_j\} \in \{\{s_t, k_j\} : s_t \in S\} \setminus B$ **do**
      $B \leftarrow B \cup \{\{s_t, k_j\}\}$
      $V(o_i) \leftarrow V(o_i) \cup V(s_t) \cap V(k_j)$
    end
  else
    $S \leftarrow S \cup \{s_i\}$
    foreach $\{s_t, k_j\} \in E(G) \setminus B$ **do**
      $B \leftarrow B \cup \{\{s_t, k_j\}\}$
      $V(o_i) \leftarrow V(o_i) \cup V(s_t) \cap S(k_j)$
    end
    foreach $\{s_i, k_t\} \in \{\{s_i, k_t\} : k_t \in K\} \setminus B$ **do**
      $B \leftarrow B \cup \{\{s_i, k_t\}\}$
      $V(o_i) \leftarrow V(o_i) \cup V(s_i) \cap V(k_t)$
    end
  end
  $o \leftarrow o \cup \{o_i\}$
end
return o

Algorithm 12: Hungarian Based Crossover (HBX).

Table 5.1: Two candidate solutions $s = \{s_1, s_2, s_3, s_4\}$ and $k = \{k_1, k_2, k_3, k_4\}$ for an example instance with $n = 24$ vertices, $m = 4$ salespersons, and $U = 6$.

$s_1 = (v_1, v_2, v_3, v_{13}, v_{14}, v_{15})$
$s_2 = (v_4, v_5, v_6, v_{16}, v_{17}, v_{18})$
$s_3 = (v_7, v_8, v_9, v_{19}, v_{20}, v_{21})$
$s_4 = (v_{10}, v_{11}, v_{12}, v_{22}, v_{23}, v_{24})$
$k_1 = (v_4, v_5, v_6, v_{13}, v_{19}, v_{22})$
$k_2 = (v_1, v_2, v_3, v_{16}, v_{20}, v_{23})$
$k_3 = (v_7, v_8, v_9, v_{14}, v_{17}, v_{24})$
$k_4 = (v_{10}, v_{11}, v_{12}, v_{15}, v_{18}, v_{21})$

Table 5.2: Weights of bipartite graph generated from s and k.

	k_1	k_2	k_3	k_4
s_1	1	3	1	1
s_2	3	1	1	1
s_3	1	1	3	1
s_4	1	1	1	3

Input: Two candidate solutions s and k
Output: A bipartite graph G

1 $s \leftarrow \{s_1, s_2, \cdots, s_m\}$
2 $k \leftarrow \{k_1, k_2, \cdots, k_m\}$
3 $E \leftarrow \left\{\{s_i, k_j\} : s_i \in s \wedge k_j \in k\right\}$
4 $w \leftarrow \left\{\left(\{s_i, k_j\}, |V(s_i) \cap V(k_j)|\right) : \{s_i, k_j\} \in E\right\}$ // $w : E \to \mathbb{R}$
5 $G \leftarrow (s \cup k, E, w)$ // build complete bipartite graph
6 **return** G

Algorithm 13: Create bipartite graph.

Input: Two partitions s and k
Output: Distance between s and k

1 $n \leftarrow$ number of vertices in s
2 $G \leftarrow$ Create bipartite graph with s and k // Algorithm 13
3 $M \leftarrow$ maximum weighted matching of G
4 **return** $n - \sum_{e \in M} w(e)$

Algorithm 14: Partitions distance.

After the recombination process, the generated offspring may violate the bounding constraints. Thus, a balancing phase is performed through Algorithm 15. This algorithm verifies if there are paths that violate the upper bound constraints. If so, a random vertex is extracted from such a path and inserted in a random path with availability.

Input: A candidate solution s and the maximum number of vertices allowed per salesperson $U \in \mathbb{Z}^+$
Output: The balanced candidate solution s

```
while ∃s_i ∈ s : |V(s_i)| > U do
    s_i ← any path in {s_i ∈ s : |V(s_i)| > U}
    v ← any vertex in V(s_i)
    V(s_i) ← V(s_i) \ {v}
    s_j ← any path in {s_j ∈ s : |V(s_j)| < U}
    V(s_j) ← V(s_j) ∪ {v}
end
return s
```

Algorithm 15: Balancing operator.

5.2.5 Computational experimentation and results

We tested the proposed algorithms over a set of instances from the literature [71]. The set of instances is made up of instances: kroA100, kroB100, kroC100, kroD100, kroE100, kroA150, kroB150, kroA200, kroB200, pr226, pr264, pr299, and pr439. The tested values of m are in $\{5, 10\}$. The tested bounding constraints are all tight. That is, the upper bound is set as $U = \lceil |V|/m \rceil$. All the proposed algorithms were implemented in C++ and executed over a hardware...

5.2.5.1 Parameter setting

We implemented the state-of-the-art ant colony-partheno genetic algorithm (AC-PGA) [1] and executed it over the used instances for comparison purposes. As far as we know, the AC-PGA is one of the best metaheuristics for the problem under consideration. The AC-PGA metaheuristic is a hybrid algorithm that combines an ant colony algorithm (ACO) with a partheno genetic algorithm (PGA). It was initially proposed for the minsum CP-DFmTSP with bounding constraints. A detailed description of the AC-PGA metaheuristic is beyond the scope of this paper. However, such details can be consulted in [1]. Table 5.5 shows the configuration we used for AC-PGA, which is based on the configuration recommended by their authors. The AC-PGA was implemented in C++ and the executions were performed on the already mentioned hardware.

Besides, we tested an implicit diversity mechanism. As mentioned before, cellular schemes also delay the convergence. Thus, a cellular memetic algorithm (cMA) version of our proposal was also executed over the tested instances. The cMA is quite similar

Table 5.3: Parameter setting of the AC-PGA metaheuristic [1].

Parameter	Value
Population size	100
AC-PGA iterations	adjustable
ACO iterations	100
ρ	0.1
α	2
β	8
γ	0.5

to our MA, with the difference that some operators differ due to the nature of the cellular approach. The cMA uses a 1D grid, and the neighborhood used for each cell is the EAST-WEST. The neighborhood is used for the selection procedure in the cMA, since our MA uses a binary tournament selection, the cMA uses a different mechanism. The cMA uses an update-policy known as kFLS [85]. In a nutshell, the kFLS is similar to the update-policy Fixed Line Sweep, which consists of updating a cell in a grid at each step time of the algorithm. Then, at the next step time, the adjacent cell is updated, and so on. The kFLS is similar but it updates k adjacent cells in the grid at each step time. According to the literature [85], this update-policy is able to keep a good balance between exploration and exploitation, which may delay the convergence in the algorithm.

5.2.5.2 Analysis of results

Table 5.4 shows the results obtained from the tested methods over the described instances. μ and σ represent the average and standard deviation of 30 independent runnings per each method. From this table, we can observe that the BNP outperforms the other approaches in all the tested cases.

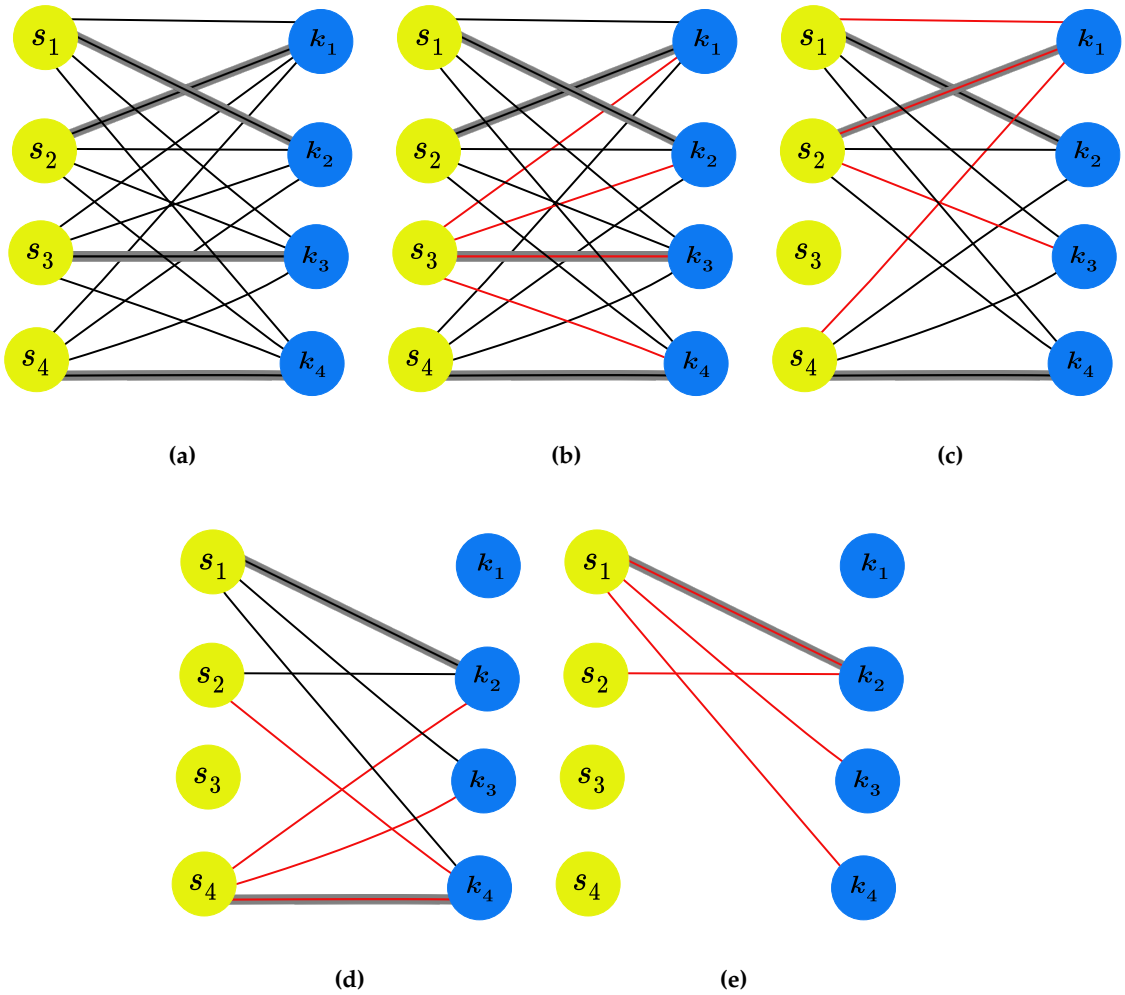


Figure 5.3: **(a)** is the bipartite complete weighted graph formed by solutions s and k, maximum weighted matching is in bold. **(b)-(e)** correspond to the iterations of the HBX. At each iteration, the paths of offspring will be composed of the intersections of paths of red edges. Thus, the paths of the offspring are composed by vertices $V(o_1) = \{v_7, v_8, v_9, v_{19}, v_{20}, v_{21}\}$, $V(o_2) = \{v_4, v_5, v_6, v_{17}, v_{13}, v_{22}\}$, $V(o_3) = \{v_{10}, v_{11}, v_{12}, v_{23}, v_{24}, v_{18}\}$, and $V(o_4) = \{v_1, v_2, v_3, v_{14}, v_{15}, v_{16}\}$.

Table 5.4: Results over some instances of the TSPLIB dataset.

instance	n	m	U	BNP		TRUNCATION		GEN		kFLS		AC-PGA	
				μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
kroA100	100	5	20	**22824.00**	0.00	22829.97	20.35	22829.10	21.52	**22824.00**	0.00	25160.23	462.14
		10	10	**24410.00**	0.00	24435.40	71.42	24433.60	70.28	**24410.00**	0.00	26453.73	462.62
kroB100	100	5	20	**23557.00**	0.00	23561.07	15.22	23571.23	25.80	23559.03	10.95	24643.90	454.43
		10	10	**24802.00**	0.00	24942.87	81.87	24926.83	75.77	24804.80	15.08	27319.33	404.46
kroA150	150	5	30	**27663.00**	0.00	27734.53	82.41	27742.80	65.06	27677.50	32.42	31438.53	444.93
		10	15	**28853.00**	0.00	29090.77	167.94	29053.70	139.03	28869.07	44.70	33950.93	597.52
kroB150	150	5	30	**27022.00**	0.00	27109.03	85.71	27118.27	102.28	27029.67	28.69	31241.93	433.97
		10	15	**28304.00**	0.00	28617.03	253.85	28563.13	205.00	28357.67	96.79	33730.37	723.66
kroA200	200	5	40	**30271.00**	0.00	30506.13	210.86	30491.63	210.65	30282.07	27.12	35766.53	512.47
		10	20	**31714.07**	8.42	31980.77	245.71	32090.20	182.30	31756.63	77.46	39130.67	591.65
kroB200	200	5	40	**30665.03**	20.97	31075.87	200.47	31022.30	185.49	30713.40	51.95	35776.13	443.86
		10	20	**31536.37**	59.20	32243.53	368.20	32012.43	249.09	31690.13	123.58	38004.83	844.48
pr226	226	5	46	**91378.00**	0.00	91891.47	761.86	91645.50	524.81	91406.00	84.00	105670.83	1584.06
		10	23	**95210.00**	0.00	96575.13	867.94	97116.10	1325.19	95575.33	331.30	112009.67	2079.92
pr264	264	5	53	**52956.00**	0.00	53010.77	238.01	53665.37	527.55	**52956.00**	0.00	58998.27	743.81
		10	27	**45241.00**	91.48	46061.50	489.31	46481.33	296.38	45348.97	240.08	53417.27	914.58
pr299	299	5	60	**49366.73**	8.23	49637.27	219.90	49495.90	143.07	49380.00	19.93	58404.97	782.82
		10	30	**50521.43**	68.21	51362.73	445.38	51259.50	531.21	50716.60	190.30	63171.87	960.41
pr439	439	5	88	**111023.73**	19.39	111327.10	497.98	111140.33	149.53	111100.47	102.01	133958.07	1702.51
		10	44	**113995.57**	171.32	117417.53	2082.71	115074.50	1114.00	115691.27	1044.45	143114.73	1242.44

Table 5.5 shows the results of applying the statistical test Wilcoxon. In this table, we confirm that the proposal that uses the explicit diversity management technique BNP obtained the best results, followed by the cellular approach, which uses an implicit management diversity technique. The proposals that do not use management diversity techniques obtained the worst scores on this performance test.

Table 5.5: Wilcoxon test.

Algorithm	Score
BNP	69
kFLS	42
GEN	-13
TRUNCATION	-18
AC-PGA	-80

Table 5.6 shows the best-found solutions for the tested instances. All of these solutions were found by the MA that uses the BNP management diversity technique. Such results intend to be a reference for the best-known solutions under the constraints for this specific problem studied in this paper.

Figure 5.4 shows the convergence behavior of the tested algorithms for some cases. From this figure, on the one hand, we observe that the AC-PGA proposal obtained the worst results. This may be due to the lack of a good balance between exploration and exploitation, which will be discussed later. On the other hand, we can observe that proposals that use any management diversity technique obtained the best results. In particular, the proposal that uses the BNP starts with worse solutions than the other methods, but after considerable progress in the evolution process, it eventually reaches and outperforms the others.

Figure 5.5 shows the diversity behavior of the evolution process for the tested methods. From this figure, we can observe some facts. We observe that diversity is lost prematurely for the methods that use TRUNCATION and GENERATIONAL, due to these do not use any management technique. The figure shows that the AC-PGA seems to have a considerable exploration. Nevertheless, it may cause a lack of exploitation such that no quality solutions are found by this algorithm in comparison with the other methods, as mentioned before. The kFLS method has more proper diversity management since it starts with good diversity and it is decreased among the evolution progress. However, in some cases, the diversity is lost rapidly and in some cases high diversity levels are maintained among the complete evolution process thus that it may affect the sensibility of the algorithm. Lastly, the proposal that uses the BNP starts

Table 5.6: Best found solutions.

instance	n	m	U	BNP best
kroA100	100	5	20	22824
		10	10	24410
kroB100	100	5	20	23557
		10	10	24802
kroA150	150	5	30	27663
		10	15	28853
kroB150	150	5	30	27022
		10	15	28304
kroA200	200	5	40	30271
		10	20	31709
kroB200	200	5	40	30644
		10	20	31512
pr226	226	5	46	91378
		10	23	95210
pr264	264	5	53	52956
		10	27	45178
pr299	299	5	60	49361
		10	30	50502
pr439	439	5	88	111017
		10	44	113668
pr1002	1002	5	201	261751
		10	101	263322
pr2392	2392	5	479	381921
		10	240	388300

with high diversity levels, and it is linearly decreased during the evolution process. This controlled diversity decrease allows for good exploration at the beginning and exploitation at the end of the algorithm, which leads to a good balance in finding good quality solutions.

Figure 5.6 shows the box plots obtained from the 30 independent runnings for each algorithm. From this figure, we can see similar observations than the previous ones. The AC-PGA tends to have worse solutions than the proposals. Besides, the methods that use management diversity techniques, the kFLS and BNP tend to find better quality solutions. These observations support the stated conjecture that proper diversity handling matters to approach the depot-free multiple traveling salesperson problem.

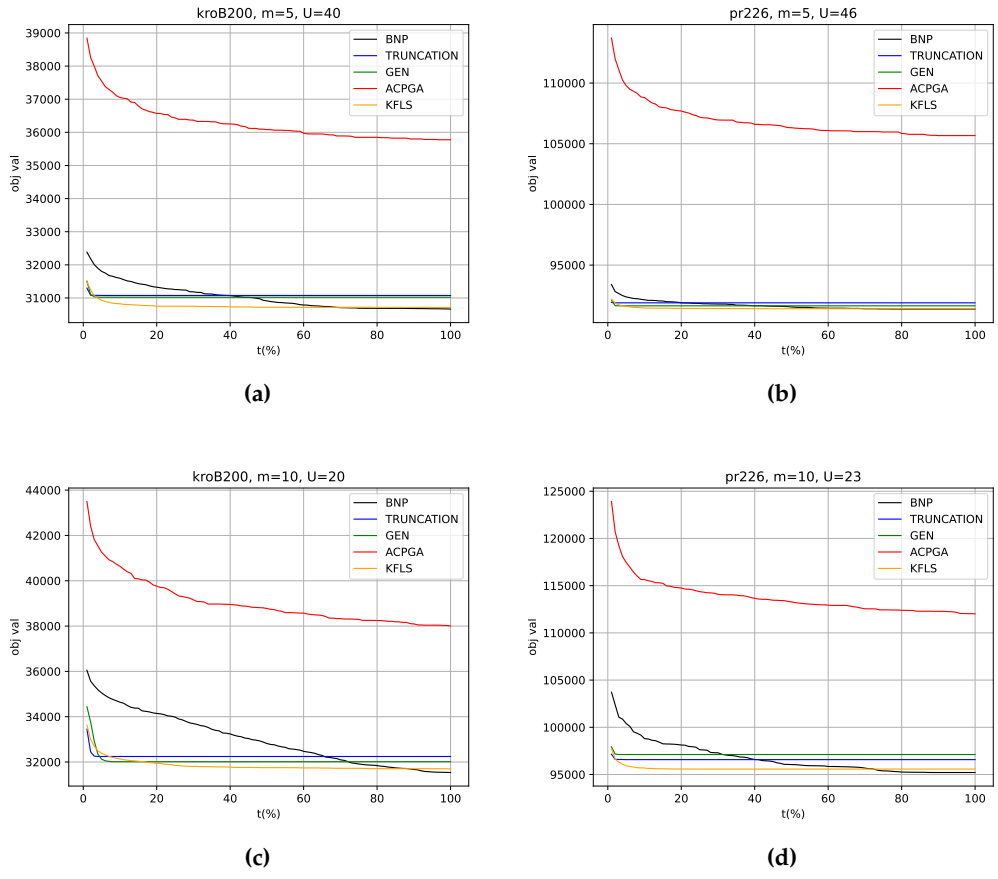


Figure 5.4: Analysis of fitness convergence.

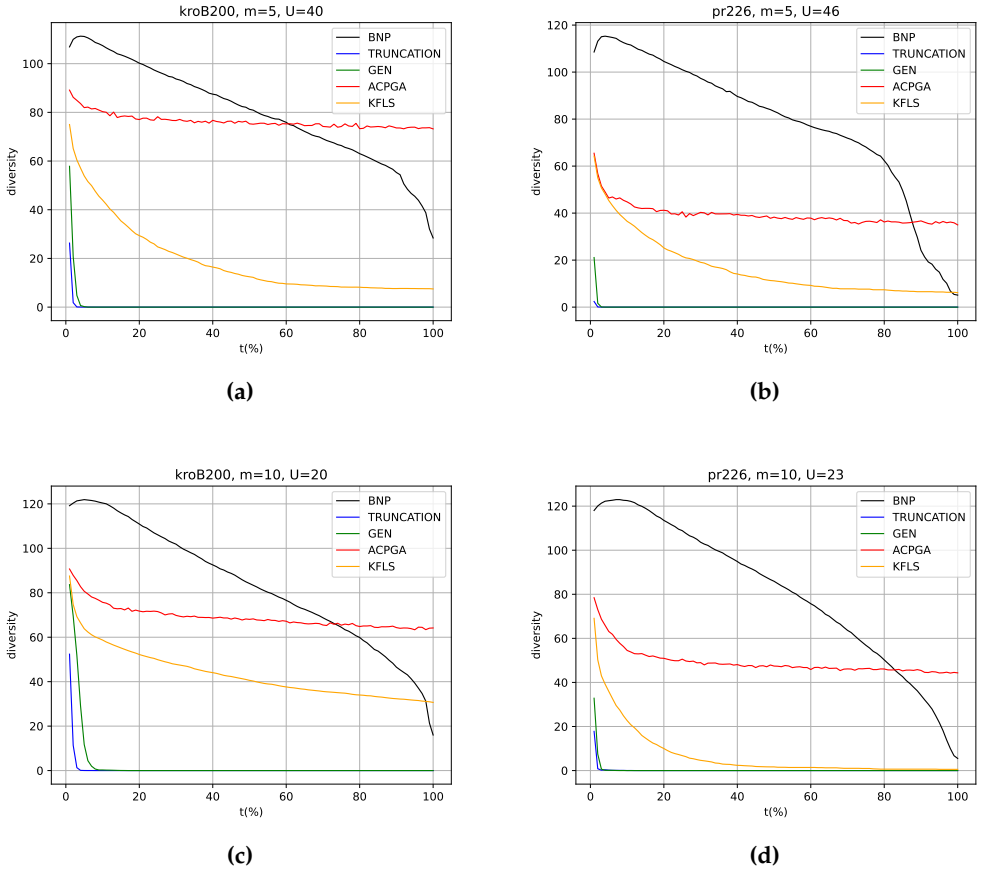


Figure 5.5: Analysis of diversity convergence.

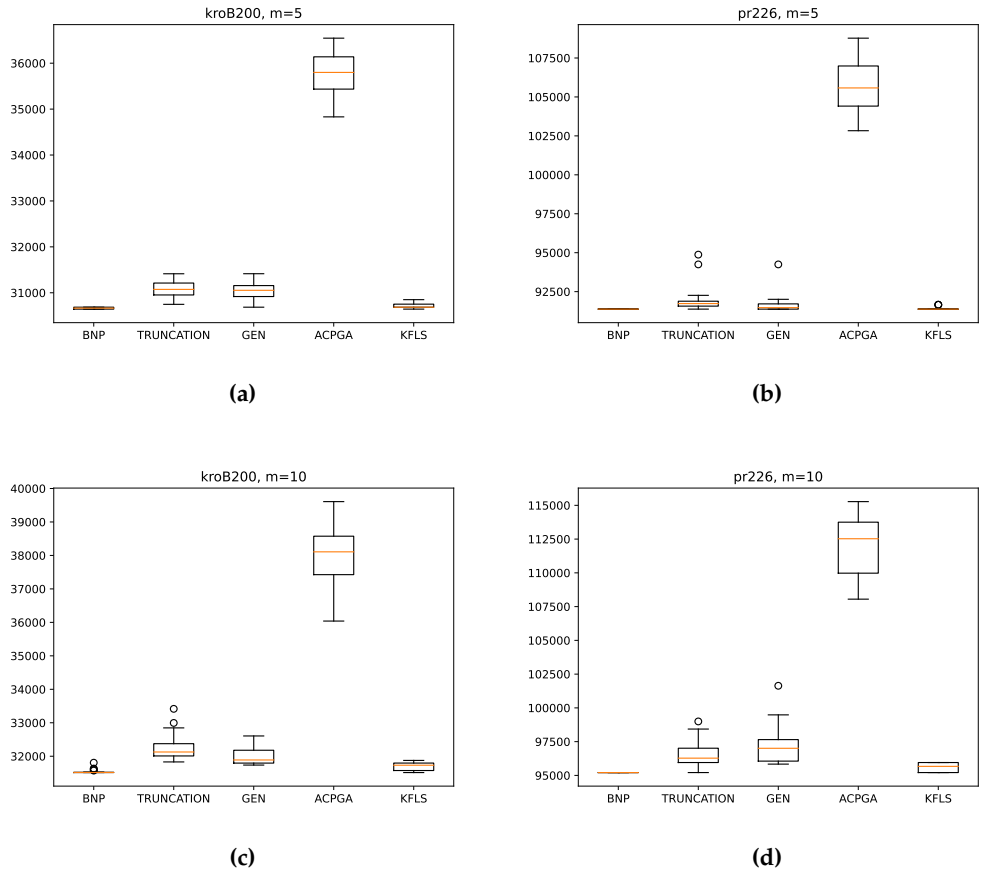


Figure 5.6: Distribution of found solutions by evolutionary algorithms.

Chapter 6

Conclusions and future work

This thesis studies the Depot-Free Multiple Traveling Salesperson Problem (DFmTSP) with bounding constraints, a variant of the Multiple Traveling Salesperson Problem (mTSP), which is an extension of the classical Traveling Salesperson Problem (TSP). We studied the problem from two different perspectives: mathematical modeling, and heuristics and metaheuristics.

On the one hand, we proposed mathematical models (IPs) that use the concept of dummy depots, which are fake vertices added to the input graphs. Using an optimization solver with classical exact algorithms, the proposed models were useful for solving relatively small instances. With this observation, we could validate some of the hypotheses and objectives of this project. On the other hand, we proposed a constructive heuristic that uses the cluster-first route-second approach. For the clustering phase, we proposed a heuristic for the CVKCP that exploits the relationship with the MCDSP. For the routing phase, we used the classical farthest-first heuristic, which has proven effective for routing problems. The main advantages of this proposal are practicality and speed. Also, an evolutionary computing approach was also studied. It consists of a Memetic Algorithm (MA) that uses exploration and exploitation components. The main difference between this MA and other proposals of the literature is that it works with the problem from a graph partition perspective and uses a state-of-the-art heuristic as the exploitation mechanism. The results show that this proposal outperforms the best metaheuristic in the literature for the specific problems studied in this thesis.

There are some interesting future directions for this work. For instance, additional mathematical models that use less than m dummy depots may be worked. Besides, the model proposed in Section 4.3 might be adapted to the minmax objective function. This model may be of special interest since it uses only $O(n^2)$ binary variables. Besides, we believe that the ideas explored in this work could also be useful for designing approximation and exact algorithms for the DFmTSP.

For the heuristics perspective, additional work concerning the two-phase constructive heuristic may be of interest. In this work, the scope of the two-phase heuristic proposal does not include lower-bound constraints. This is because the CVKP characteristics consider only an upper-bound in the number of assigned vertices to each center. Thus, variants of heuristics that consider lower-bound constraints may also be studied in the future. Finally, other metaheuristic approaches such as GRASP could be studied [86]. GRASP metaheuristic is of special interest because it stands out as a good tool for combinatorial optimization problems that involve graphs and greedy algorithms.

Bibliography

[1] C. Jiang, Z. Wan, and Z. Peng, "A new efficient hybrid algorithm for large scale multiple traveling salesman problems," *Expert Systems with Applications*, vol. 139, p. 112867, jan 2020.

[2] R. M. Karp, *Reducibility among combinatorial problems*. Springer, 1972.

[3] L. A. Hemaspaandra, "Sigact news complexity theory column 36," *ACM SIGACT News*, vol. 33, no. 2, pp. 34–47, 2002.

[4] K. Karabulut, H. Öztop, L. Kandiller, and M. F. Tasgetiren, "Modeling and optimization of multiple traveling salesmen problems: An evolution strategy approach," *Computers & Operations Research*, vol. 129, p. 105192, 2021.

[5] R. J. Trudeau, *Introduction to graph theory*. New York: DOVER PUBLICATIONS, INC, 1993.

[6] O. Naud, J. Taylor, L. Colizzi, R. Giroudeau, S. Guillaume, E. Bourreau, T. Crestey, and B. Tisseyre, "Support to decision-making," in *Agricultural Internet of Things and Decision Support for Precision Smart Farming*, pp. 183–224, Elsevier, 2020.

[7] C. H. Papadimitriou and K. Steiglitz, *Combinatorial Optimization. Algorithms and Complexity*. Dover, 1 ed., 1998.

[8] R. Diestel, "Graph theory 3rd ed," *Graduate texts in mathematics*, vol. 173, p. 33, 2005.

[9] G. B. Dantzig and M. N. Thapa, *Linear Programming. 1, Introduction {Springer Series in Operations Research}*. Springer-Verlag New York Incorporated, 1997.

[10] D. G. Luenberger, Y. Ye, *et al.*, *Linear and nonlinear programming*, vol. 2. Springer, 1984.

[11] M. Conforti, G. Cornuejols, and G. Zambelli, *Integer Programming*. Springer International Publishing, 1 ed., 2014.

[12] M. S. Daskin, *Network and Discrete Location Models, algorithms, and Applications*. John Wiley & Sons, Inc., 2013.

[13] V. V. Vazirani, *Approximation algorithms*. Springer Science & Business Media, 2013.

[14] H. A. Taha, "Operations research an introduction," 2007.

[15] R. Z. Farahani and M. Hekmatfar, *Facility location: concepts, models, algorithms and case studies*. Springer Science & Business Media, 2009.

[16] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Operations research*, vol. 12, no. 3, pp. 450–459, 1964.

[17] G. Laporte, S. Nickel, and F. S. da Gama, *Location Science*. Springer International Publishing, 2019.

[18] M. Daskin, "Network and discrete location: models, algorithms and applications," *Journal of the Operational Research Society*, vol. 48, no. 7, pp. 763–764, 1997.

[19] T. Ilhan, F. Ozsoy, and M. Pinar, "An efficient exact algorithm for the vertex p-center problem and computational experiments for different set covering subproblems," *Bilkent University, Department of Industrial Engineering, Technical Report*, 2002.

[20] H. Calik and B. C. Tansel, "Double bound method for solving the p-center location problem," *Computers & operations research*, vol. 40, no. 12, pp. 2991–2999, 2013.

[21] J. Garcia-Diaz, R. Menchaca-Mendez, R. Menchaca-Mendez, S. Pomares Hernández, J. C. Pérez-Sansalvador, and N. Lakouari, "Approximation algorithms for the vertex k-center problem: Survey and experimental evaluation," *IEEE Access*, vol. 7, pp. 109228–109245, 2019.

[22] S. Khuller and Y. J. Sussmann, "The capacitated k-center problem," *SIAM J. Discret. Math.*, vol. 13, p. 403–418, May 2000.

[23] B. LaLonde and P. H. Zinszer, "Customer service meaning and measurement, national council of physical distribution management, chicago, il," *Search in*, 1976.

[24] B. L. Golden and E. A. Wasil, "Or practice—computerized vehicle routing in the soft drink industry," *Operations research*, vol. 35, no. 1, pp. 6–17, 1987.

[25] M. S. Daskin, "Logistics: an overview of the state of the art and perspectives on future research," *Transportation Research Part A: General*, vol. 19, no. 5-6, pp. 383–398, 1985.

[26] G. Dantzig, R. Fulkerson, and S. Johnson, "Solution of a large-scale traveling-salesman problem," *Journal of the operations research society of America*, vol. 2, no. 4, pp. 393–410, 1954.

[27] A. Mor and M. G. Speranza, "Vehicle routing problems over time: a survey," *Annals of Operations Research*, vol. 314, no. 1, pp. 255–275, 2022.

[28] G. B. Dantzig, D. R. Fulkerson, and S. M. Johnson, "On a linear-programming, combinatorial approach to the traveling-salesman problem," *Operations Research*, vol. 7, no. 1, pp. 58–66, 1959.

[29] C. E. Miller, A. W. Tucker, and R. A. Zemlin, "Integer Programming Formulation of Traveling Salesman Problems," *J. ACM*, vol. 7, no. 4, pp. 326–329, 1960.

[30] K. Daniel, *Thinking, fast and slow*. 2017.

[31] D. S. Hochbaum, "Approximation algorithms for np-hard problems," *ACM Sigact News*, vol. 28, no. 2, pp. 40–52, 1997.

[32] G. Pólya and J. H. Conway, *How to solve it: A new aspect of mathematical method*. Princeton University Press Princeton, 1975.

[33] M. Hjeij and A. Vilks, "A brief history of heuristics: how did research on heuristics evolve?," *Humanities and Social Sciences Communications*, vol. 10, no. 1, pp. 1–15, 2023.

[34] C. Blum and A. Roli, "Metaheuristics in combinatorial optimization: Overview and conceptual comparison," *ACM Comput. Surv.*, vol. 35, p. 268–308, Sept. 2003.

[35] M. Gendreau and J.-Y. Potvin, eds., *Handbook of Metaheuristics*. Springer International Publishing, 3 ed., 2019.

[36] R. Mart, P. M. Pardalos, and M. G. Resende, *Handbook of heuristics*. Springer Publishing Company, Incorporated, 2018.

[37] K. Sörensen and F. Glover, "Metaheuristics," *Encyclopedia of operations research and management science*, vol. 62, pp. 960–970, 2013.

[38] R. Martí, M. Sevaux, and K. Sörensen, "50 years of metaheuristics," *European Journal of Operational Research*, 2024.

[39] D. Weyland, "A rigorous analysis of the harmony search algorithm: How the research community can be misled by a "novel" methodology," *International Journal of Applied Metaheuristic Computing (IJAMC)*, vol. 1, no. 2, pp. 50–60, 2010.

[40] J. H. Holland, *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, 1975. second edition, 1992.

[41] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Computers & operations research*, vol. 13, no. 5, pp. 533–549, 1986.

[42] T. A. Feo and M. G. Resende, "A probabilistic heuristic for a computationally difficult set covering problem," *Operations research letters*, vol. 8, no. 2, pp. 67–71, 1989.

[43] N. Mladenovic, "A variable neighborhood algorithm-a new metaheuristic for combinatorial optimization," in *papers presented at Optimization Days*, vol. 12, 1995.

[44] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial intelligence through simulated evolution," 1966.

[45] W. Vent, "Rechenberg, ingo, evolutionsstrategie — optimierung technischer systeme nach prinzipien der biologischen evolution. 170 s. mit 36 abb. frommann-holzboog-verlag. stuttgart 1973. broschiert," *Feddes Repertorium*, vol. 86, pp. 337–337, 1975.

[46] J. Koza and R. Poli, *Genetic Programming*, pp. 127–164. 01 2005.

[47] J. A. Svestka and V. E. Huckfeldt, "COMPUTATIONAL EXPERIENCE WITH AN M-SALESMAN TRAVELING SALESMAN ALGORITHM," *Management Science*, vol. 19, no. 7, pp. 790–799, 1973.

[48] I. Kara and T. Bektas, "Integer linear programming formulations of multiple salesman problems and its variations," *European Journal of Operational Research*, vol. 174, pp. 1449–1458, nov 2006.

[49] I. Branco and J. D. Coelho, "The hamiltonian p-median problem," *European Journal of Operational Research*, vol. 47, no. 1, pp. 86–95, 1990.

[50] O. Cheikhrouhou and I. Khoufi, "A comprehensive survey on the multiple traveling salesman problem: Applications, approaches and taxonomy," *Computer Science Review*, vol. 40, p. 100369, 2021.

[51] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, 2006.

[52] J. A. Cornejo-Acosta, J. García-Díaz, J. C. Pérez-Sansalvador, and C. Segura, "Compact Integer Programs for Depot-Free Multiple Traveling Salesperson Problems," *Mathematics*, vol. 11, no. 13, 2023.

[53] M. Bellmore and S. Hong, "Transformation of multisalesman problem to the standard traveling salesman problem," *Journal of the ACM (JACM)*, vol. 21, no. 3, pp. 500–504, 1974.

[54] M. Rao, "A note on the multiple traveling salesmen problem," *Operations Research*, vol. 28, no. 3-part-i, pp. 628–632, 1980.

[55] Y. GuoXing, "Transformation of multidepot multisalesmen problem to the standard travelling salesman problem," *European Journal of Operational Research*, vol. 81, no. 3, pp. 557–560, 1995.

[56] P. M. França, M. Gendreau, G. Laporte, and F. M. Müller, "The m-traveling salesman problem with minmax objective," *Transportation Science*, vol. 29, no. 3, pp. 267–275, 1995.

[57] M. J. Assaf, *Transformations for variants of the travelling salesman problem and applications*. PhD thesis, 2017.

[58] A. E. Carter and C. T. Ragsdale, "A new approach to solving the multiple traveling salesperson problem using genetic algorithms," *European Journal of Operational Research*, vol. 175, no. 1, pp. 246–257, 2006.

[59] S. N. Kabadi and A. P. Punnen, "The bottleneck tsp," in *The traveling salesman problem and its variations*, pp. 697–735, Springer, 2007.

[60] R. A. Russell, "Technical note—an effective heuristic for the m-tour traveling salesman problem with some side conditions," *Operations Research*, vol. 25, no. 3, pp. 517–524, 1977.

[61] R. I. Bolaños, E. M. Toro Ocampo, and M. Granada Echeverri, "A population-based algorithm for the multi travelling salesman problem," *International Journal of Industrial Engineering Computations*, vol. 7, no. 2, pp. 245–256, 2016.

[62] X. Xu, H. Yuan, M. Liptrott, and M. Trovati, "Two phase heuristic algorithm for the multiple-travelling salesman problem," *Soft Computing*, vol. 22, p. 6567–6581, 10 2018.

[63] M. Latah, "Solving Multiple TSP Problem by K-Means and Crossover based Modified ACO Algorithm," *International Journal of Engineering Research & Technology (IJERT)*, vol. 5, pp. 430–434, 02 2016.

[64] N. Sathya and A. Muthukumaravel, "Two phase hybrid ai-heuristics for multiple travelling salesman problem," *International Journal of Applied Engineering Research*, vol. 12, pp. 12659–12664, 01 2017.

[65] A. Otman, H. Khatir, R. Mouhssine, and M. Ezziyyani, *An Effective Parallel Approach to Solve Multiple Traveling Salesmen Problem: Volume 5: Advanced Intelligent Systems for Computing Sciences*, pp. 647–664. 01 2019.

[66] A. Steven, G. F. Hertono, and B. D. Handari, "Implementation of clustered ant colony optimization in solving fixed destination multiple depot multiple traveling salesman problem," in *2017 1st International Conference on Informatics and Computational Sciences (ICICoS)*, pp. 137–140, 2017.

[67] M. Yousif and B. Al-Khateeb, "A novel metaheuristic algorithm for multiple traveling salesman problem," *Journal of Advanced Research in Dynamical and Control Systems*, vol. 10, pp. 2113–2122, 01 2018.

[68] S. P. Anbuudayasankar, K. Ganesh, and S. Mohapatra, *Models for Practical Routing Problems in Logistics*. Springer International Publishing, 1 ed., 2014.

[69] H. Zhou, M. Song, and W. Pedrycz, "A comparative study of improved GA and PSO in solving multiple traveling salesmen problem," *Applied Soft Computing*, vol. 64, pp. 564–580, 2018.

[70] Gurobi Optimization, LLC, "Gurobi Optimizer Reference Manual," 2022.

[71] G. Reinelt, "TSPLIB—A Traveling Salesman Problem Library," *ORSA Journal on Computing*, vol. 3, no. 4, pp. 376–384, 1991.

[72] J. A. Cornejo Acosta, J. García Díaz, R. Menchaca-Méndez, and R. Menchaca-Méndez, "Solving the Capacitated Vertex K-Center Problem through the Minimum Capacitated Dominating Set Problem," *Mathematics*, vol. 8, p. 1551, sep 2020.

[73] D. Rosenkrantz, R. Stearns, and P. M. L. II, "AN ANALYSIS OF SEVERAL HEURISTICS FOR THE TRAVELING SALESMAN PROBLEM," *SIAM J. Comput.*, vol. 6, pp. 563–581, 09 1977.

[74] W. J. Cook, *In Pursuit of the Traveling Salesman*. New Jersey: Princeton University Press, 1 ed., 2012.

[75] D. S. Johnson and L. A. McGeoch, *Experimental Analysis of Heuristics for the STSP*, pp. 369–443. Boston, MA: Springer US, 2007.

[76] S. S. Skiena, *The Algorithm Design Manual*. Springer Cham, 3 ed., 2020.

[77] J. A. Cornejo-Acosta, J. García-Díaz, J. C. Pérez-Sansalvador, R. Z. Ríos-Mercado, and S. E. Pomares-Hernández, "A constructive heuristic for the uniform capacitated vertex k-center problem," *ACM J. Exp. Algorithmics*, vol. 28, aug 2023.

[78] B. Dezső, A. Jüttner, and P. Kovács, "LEMON — an Open Source C++ Graph Template Library," *Electronic Notes in Theoretical Computer Science*, vol. 264, pp. 23–45, 07 2011.

[79] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2nd ed., 2015.

[80] M. Črepinšek, S.-H. Liu, and M. Mernik, "Exploration and exploitation in evolutionary algorithms: A survey," *ACM Comput. Surv.*, vol. 45, jul 2013.

[81] C. Segura, C. A. Coello Coello, G. Miranda, and C. León, "Using multi-objective evolutionary algorithms for single-objective optimization," *4OR*, vol. 11, pp. 201–228, Sept. 2013.

[82] J. Chacón Castillo and C. Segura, "Differential evolution with enhanced diversity maintenance," *Optimization Letters*, vol. 14, pp. 1471–1490, Sept. 2020.

[83] O. M. González, C. Segura, S. I. V. Peña, and C. León, "A memetic algorithm for the capacitated vehicle routing problem with time windows," in *2017 IEEE Congress on Evolutionary Computation (CEC)*, pp. 2582–2589, 2017.

[84] C. Segura, A. H. Aguirre, S. I. V. Peña, and S. B. Rionda, *The Importance of Proper Diversity Management in Evolutionary Algorithms for Combinatorial Optimization*, pp. 121–148. Cham: Springer International Publishing, 2017.

[85] J. A. Cornejo-Acosta and J. García-Díaz, "A First Approach to Asynchronous-Synchronous Tradeoff in 1D Cellular Genetic Algorithms," *Research in Computing Science*, vol. 150, no. 12, 2021.

[86] M. G. C. Resende and C. C. Ribeiro, *Optimization by GRASP: Greedy Randomized Adaptive Search Procedures*. New York: Springer, 2016.

Appendix

Acronyms

- **ACO** Ant Colony Optimization
- **cEA** Cellular Evolutionary Algorithm
- **CVKCP** Capacitated Vertex k-center Problem
- **EA** Evolutionary Algorithm
- **EC** Evolutionary Computing
- **GA** Genetic Algorithm
- **GRASP** Greedy Randomized Adaptive Search Procedures
- **ILP** Integer Linear Program
- **IP** Integer Program
- **MA** Memetic Algorithm
- **MCDSP** Minimum Capacitated Dominating Set Problem
- **MDSP** Minimum Dominating Set Problem
- **MILP** Mixed Integer Linear Program
- **MmTSP** Multiple depot Multiple Traveling Salesperson Problem
- **mTSP** Multiple Traveling Salesperson Problem
- **PGA** Partheno Genetic Algorithm
- **SECs** Subtour Elimination Constraints
- **SIMD** Single-Instruction Multiple-Data
- **SmTSP** Single depot Multiple Traveling Salesmen Problem

- **TS** Tabu Search
- **TSP** Traveling Salesperson Problem
- **VDP** Vehicle Dispatch Problem
- **VKCP** Vertex k-center Problem
- **VNS** Variable Neighborhood Search