ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

DOTTORATO DI RICERCA IN

INGEGNERIA BIOMEDICA, ELETTRICA E DEI SISTEMI

Ciclo 38

Settore Concorsuale: 01/A6 - RICERCA OPERATIVA

Settore Scientifico Disciplinare: MAT/09 - RICERCA OPERATIVA

MODELS AND ALGORITHMS FOR ROUTING AND NETWORK OPTIMIZATION PROBLEMS

Presentata da: Federico Michelotto

Coordinatore Dottorato

Michele Monaci

Supervisore

Michele Monaci

Co-supervisore

Daniele Vigo

Esame finale anno 2026

Abstract

This thesis explores three major optimization problems in transportation and communication systems that address modern practical challenges.

First, we introduce the *Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds*, a new variant of the TSP that models coordinated truck-drone delivery with dynamically adjustable drone speeds. By formulating the problem as a Mixed Integer Linear Program under convex drone energy-per-meter functions, we develop the first exact solution method. Results demonstrate that optimizing drone speed can reduce delivery completion times by an average of 32%, highlighting the importance of adaptive drone speed control in last-mile logistics. To handle larger instances, a genetic algorithm is proposed, achieving significant improvements over existing heuristics.

Next, we address the *Capacitated Vehicle Routing Problem*, investigating whether it is possible to independently optimize different parts of a CVRP solution while maintaining feasibility. We uncover a counterintuitive structural property of the CVRP that enables a novel *sequence-based decomposition* approach, providing new insights into parallel optimization for routing problems.

Finally, we study the *Routing and Wavelength Assignment* problem in optical networks under a shared protection policy. We formulate Integer Linear Programming relaxations and develop an iterated local search metaheuristic. Computational experiments on realistic instances show that the proposed algorithms achieve feasible solutions with tight optimality gaps, effectively balancing survivability and efficiency in optical communication networks.

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Professor Michele Monaci, and my co-supervisor, Professor Daniele Vigo, for giving me the opportunity and the freedom to work on challenging and stimulating research problems.

I am also thankful for the valuable insights and support provided by Professors Enrico Malaguti and Valentina Cacchiani.

A heartfelt thank you to my colleagues and friends, Luca Accorsi, Francesco Cavaliere, and Antonio Punzo, for their companionship and for sharing this journey with me.

Lastly, I thank Professors Roberto Roberti, Domenico Salvagnin, and Matteo Fischetti from the University of Padova for their kind invitations to valuable seminars, which provided significant learning opportunities.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

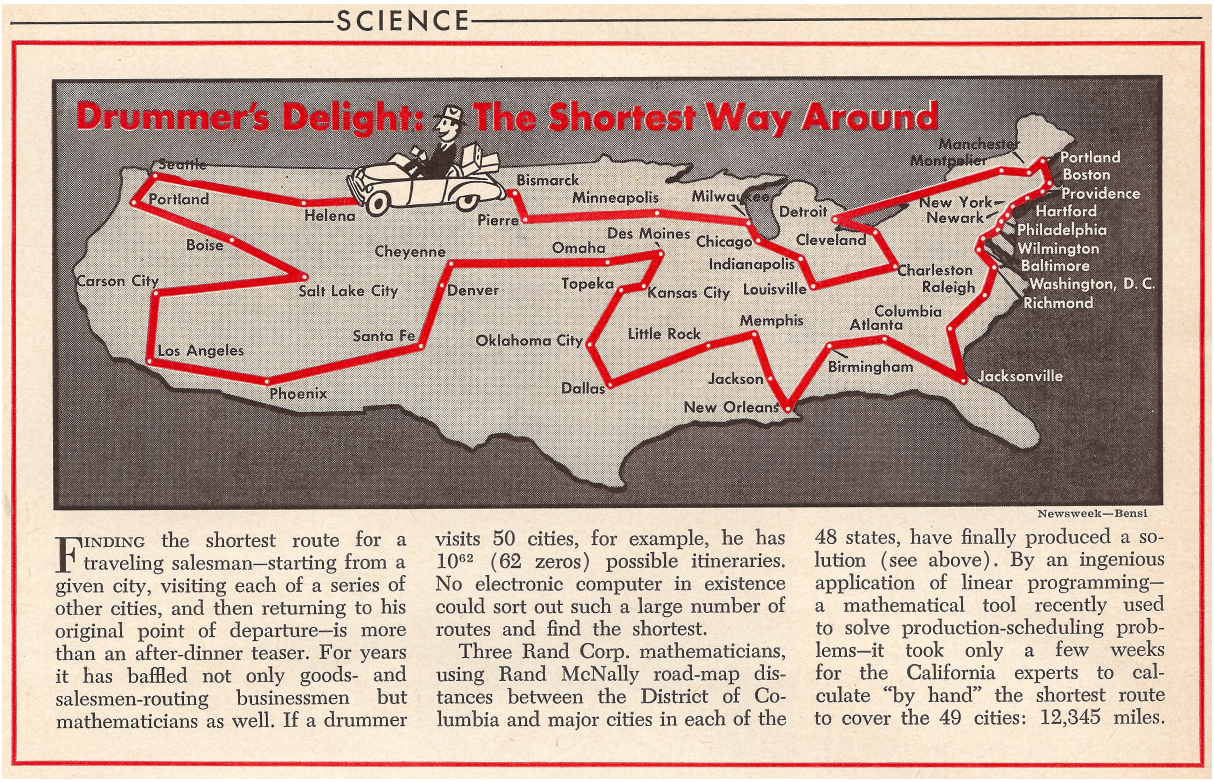
To introduce some fundamental concepts in operations research and, at the same time, the work developed during my PhD, I would like to start with what is arguably the single most influential paper in the field: "*Solution of a Large-Scale Traveling-Salesman Problem*" by Dantzig, Fulkerson and Johnson (1954).

The *Traveling Salesman Problem* (TSP) is one of the most popular problems studied in operations research. The TSP simply asks: *What is the shortest route (closed path) that allows a salesman to visit a list of cities and return to the point of origin?* The TSP has a surprisingly long history. Its earliest known mention dates back to 1832, in a German guidebook for traveling salesmen containing practical suggestions on how to plan efficient trips. However, the first written mathematical approaches to the problem appeared only a century later, in 1930, in a work by Karl Menger.

In the symmetric version of the TSP, the cost of going from node a to node b is the same as going from node b to node a. Whereas, in the asymmetric TSP, this is not true in general. Given a problem with n cities, it is easy to observe by fixing the starting city (without loss of generality) that the number of possible solutions for the asymmetric TSP is $(n-1)!$ and $(n-1)!/2$ for the symmetric TSP. This exponential growth quickly makes exhaustive search infeasible. To give an idea about the complexity of the problem, when the number of cities n is 65, there are about 10^{90} different solutions for the TSP, and to put this number into perspective, just think that 10^{90} is the estimated number of atoms in the observable universe Guth (2001)! The TSP has numerous practical applications, from route planning and logistics to circuit design, genome mapping, telescope scheduling, and many others. Unfortunately, the TSP belongs to the class of $\mathcal{NP}$-hard problems, which means that – assuming $\mathcal{P} \neq \mathcal{NP}$ – no polynomial-time algorithms exist for this problem. Operations research aims precisely to address such challenges: finding optimal, or near-optimal solutions when not possible, to $\mathcal{NP}$-hard combinatorial optimization problems.

One of the first TSP instances that sparked interest among researchers was the so-called "48-states problem", introduced by Hassler Whitney in the early 1930s during a seminar talk at Princeton University, in which the cities to be visited were the 48 capital cities of the United States (Alaska and Hawaii were not yet states at the time). Nearly 25 years later, in 1954, George Dantzig, Ray Fulkerson and Selmer Johnson – hereafter, DFJ – published the first optimal solution for the 48-states problem, actually, a variant with 49 cities, one for each state, not necessarily capitals, and Washington D. C – reported as the 49-city problem. The DFJ paper "*Solution of a Large-Scale Traveling-Salesman Problem*", whose concise abstract is reported below, does not provide a complete procedure on how to compute an optimal solution, but rather a mathematical proof showing that a specific tour is optimal, that is, that a shorter tour is not possible. The optimal tour found is shown in Figure 1.1.

> *It is shown that a certain tour of 49 cities, one in each of the 48 states and Washington, D. C., has the shortest road distance.*



SCIENCE

Drummer's Delight: The Shortest Way Around

FINDING the shortest route for a traveling salesman—starting from a given city, visiting each of a series of other cities, and then returning to his original point of departure—is more than an after-dinner teaser. For years it has baffled not only goods- and salesmen-routing businessmen but mathematicians as well. If a drummer visits 50 cities, for example, he has 10^{62} (62 zeros) possible itineraries. No electronic computer in existence could sort out such a large number of routes and find the shortest.

Three Rand Corp. mathematicians, using Rand McNally road-map distances between the District of Columbia and major cities in each of the 48 states, have finally produced a solution (see above). By an ingenious application of linear programming—a mathematical tool recently used to solve production-scheduling problems—it took only a few weeks for the California experts to calculate "by hand" the shortest route to cover the 49 cities: 12,345 miles.

Figure 1.1: Dantzig, Fulkerson, and Johnson 49-city tour, Newsweek, July 26, 1954.

To achieve this result, they tackled the 49-city problem by using *Linear Programming*, a mathematical framework used to model optimization problems in which both the constraints and the objective function to be optimized are defined by linear relations. A linear program (LP) is an optimization problem with N variables and $M \leq N$ constraints,

that, in *standard form*, is expressed as:

$$\min \sum_{i=1}^{N} c_i x_i \tag{1.1a}$$

$$\sum_{i=1}^{N} a_{ij} x_i = b_j \qquad \forall\, j \in [1, M] \tag{1.1b}$$

$$x_i \geq 0 \qquad \forall\, i \in [1, N] \tag{1.1c}$$

Here $x_i, i \in [1, N]$, are the N non-negative decision variables of the problem. The objective coefficients $c_i \in \mathbb{R}, i \in [1, N]$, specify the contribution of each variable to the objective function and thus determine the cost of the solution. The constants $a_{ij}, b_j \in \mathbb{R}, i \in [1, N], j \in [1, M]$, define M linearly independent constraints which characterize the feasible region of the decision variables.

Linear Programming theory – developed independently by Kantorovich in 1939, by Koopmans in 1947 and by Dantzig in 1947 – says that if the N-dimensional polyhedron corresponding to the LP feasible region is non-empty and finite, i.e., a polytope, then an optimal solution lies at one of the vertices. A *basic solution* is a solution obtained by setting to zero $N - M$ variables (*non-basic variables*) and solving the uniquely determined system of M equations in M variables (*basic variables*), thus there are at most $\binom{N}{M}$ basic solutions. If the solution's variables are non-negative, it is called *basic feasible solution* and each basic feasible solution corresponds to a vertex of the polytope, and vice versa. Thus, computing an optimal solution is equivalent to finding the minimum-cost vertex of the polytope representing the feasible region. To compute such a vertex, Dantzig developed in 1947 the *simplex method*.

Linear Programming theory says that it is possible to describe the objective function with respect to any basic solution $\overline{x} = (\overline{x}_1, \overline{x}_2, ..., \overline{x}_N)$ as a function of the non-basic variables as follows.

$$\sum_{i=1}^{N} c_i x_i = \sum_{i=1}^{N} c_i \overline{x}_i + \sum_{i=1}^{N} \delta_i x_i \tag{1.2}$$

Where the values $\delta_1, \delta_2, ..., \delta_N$ – nowadays called *reduced costs* – are uniquely determined by defining $\delta_i = 0$ if $x_i, i \in [1, N]$, is a basic variable. The reduced cost of a non-basic variable x_i determines the change in the objective function if x_i is increased by one unit, or in other terms, is the partial derivative of the objective function with respect to x_i, computed in $\overline{x}$. The simplex method starts from a vertex (basic feasible solution) and, relying on equation (1.2), iteratively moves to an adjacent vertex that improves (or at least does not worsen) the objective function. If the problem is bounded, the algorithm continues until all reduced costs are non-negative, indicating that the current vertex is optimal.

Although theoretically the simplex method can require an exponential number of steps to converge in the worst-case, in practice it remains the most effective and fastest algorithm for solving the vast majority of LPs. Because of its widespread use in mathematics and industrial applications, the simplex method has been recognized as one of the top ten algorithms of the twentieth century.

However, no one has yet succeeded in formulating the TSP as a linear program, and the common belief is that this is not possible. Moreover, achieving such a formulation using a polynomial number of variables and constraints with respect to the number of cities would demonstrate that $\mathcal{P} = \mathcal{NP}$ – a result that would resolve one of the Millennium Prize Problems, among the seven most profound and challenging open questions in mathematics. Such a breakthrough would not only earn its discoverer a one-million-dollar prize but also a place in the annals of mathematical history.

Instead, by imposing integer-valued decision variables (*integrality constraints*), the TSP can be formulated as an *Integer Linear Program* (ILP). Unfortunately, ILPs are non-convex optimization problems since the feasible region consists of a discrete set of points, and therefore are typically much more complex problems than LPs.

Given $n \in \mathbb{N}$ cities, indexed from 1 to n, and the distances $c_{ij} \in \mathbb{N}$ between each pair of cities $i, j \in [1, n]$, DFJ noted that the symmetric TSP could be formulated as follows.

$$\min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij} \tag{1.3a}$$

$$\sum_{i=1}^{j-1} x_{ij} + \sum_{i=j+1}^{n} x_{ji} = 2 \qquad \forall\, j \in [1, n] \tag{1.3b}$$

$$\sum_{\substack{i,j \in S \\ i<j}} x_{ij} \leq |S| - 1 \qquad \forall S \subset [1, n] \,:\, |S| > 1 \tag{1.3c}$$

$$x_{ij} \in \{0, 1\} \qquad \forall\, i, j \in [1, n] \,:\, i < j \tag{1.3d}$$

Here, the binary decision variables $x_{ij} \in \{0, 1\}, i, j \in [1, n], i < j$, indicate if cities i and j are visited consecutively ($x_{ij} = 1$) or not ($x_{ij} = 0$). Inequalities (1.3b) ensure that each city is visited in a closed path. Inequalities (1.3c) ensure that no subtours can occur, i.e., closed paths that do not visit all cities, and for this reason they are called nowadays *subtour elimination constraints* or *SECs* (originally called *loop conditions* in the DFJ paper). Finally, the objective function to be minimized is the total distance traveled (1.3a).

The issue with this formulation is twofold. First, the number of subtour elimination constraints (1.3c) is exponential with respect to the number of cities n, precisely, $2^n - n - 2$ (almost a million of billions for the 49-city problem), making such a formulation

impractical even today for even smaller values of n. Second, in 1954 no method was yet invented to solve ILPs.

So, how DFJ were able to find an optimal solution for the 49-city problem? They started by temporarily ignoring both problematics, i.e., substituting the non-convex integrality constraints (1.3d) with the linear constraints $x_{ij} \geq 0 \; \forall \; i, j \in [1, n]$, and removing the subtour elimination constraints (1.3c). Thus, leading to the following LP.

$$\min \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij} \tag{1.4a}$$

$$\sum_{i=1}^{j-1} x_{ij} + \sum_{i=j+1}^{n} x_{ji} = 2 \qquad \forall \; j \in [1, n] \tag{1.4b}$$

$$x_{ij} \geq 0 \qquad \forall \; i, j \in [1, n] \; : \; i < j \tag{1.4c}$$

Of course, this formulation does not model the TSP since the corresponding optimal solution could have fractional values and could contain subtours. However, it is much easier to solve, making it more appealing than (1.3). Moreover, it can be proven that all vertices of the polytope described by (1.4) are integer points. Note that each solution for (1.3) is also a solution for (1.4) and with a cost not greater (in this case, equal) than the one in (1.3), making the formulation (1.4) a *relaxation* of (1.3). The main property of a relaxation is that its optimal solution's cost is a lower bound on the optimal solution's cost of the original formulation.

The approach used by DFJ to solve the 49-city problem can be summarized as follows.

1. *Initialize the simplex method for the LP in* (1.4) *with a TSP solution* $\overline{\boldsymbol{x}}$ *that looks promising (recall that all TSP solutions are basic feasible solutions, i.e., vertices, for this LP).*
2. *If all reduced costs with respect to the basic feasible solution* $\overline{\boldsymbol{x}}$ *are non-negative,* $\overline{\boldsymbol{x}}$ *is an optimal tour. Exit.*
3. *Apply an iteration of the simplex method to obtain a new improving (or non-worsening) basic feasible solution* $\boldsymbol{x}$ *adjacent to* $\overline{\boldsymbol{x}}$ *in the LP polytope.*
4. *If* $\boldsymbol{x}$ *is an integer tour, update* $\overline{\boldsymbol{x}}$ *with* $\boldsymbol{x}$*. Otherwise, there are three possibilities:* $\boldsymbol{x}$ *could have subtours, variables with a value greater than 1, and/or fractional values. In all cases, augment the LP by adding one or more constraints that remove* $\boldsymbol{x}$ *from the feasible region without removing any TSP solution, in a way that* $\overline{\boldsymbol{x}}$ *is still a vertex of the new LP polytope.*
5. *Go to 2.*

They started with an initial TSP solution $\overline{\boldsymbol{x}}$ with a cost of 699 (12345 miles). During the process, they added 25 constraints among 7 SECs, 16 upper bound constraints

($x_{ij} \leq 1$), and 2 other valid constraints to remove fractional solutions, without finding an improving integer tour. After that, the augmented LP was such that all reduced costs with respect to $\bar{x}$ were non-negative but one with value 0.5. But, since distances are integers, and decision variables cannot be greater than 1, the objective function must be an integer, concluding that 699 is the minimum distance for the 49-city problem. Moreover, by continuing the procedure, they were able to prove that $\bar{x}$ is also the unique optimal solution. Remarkably, all computations were computed by hand.

In retrospect, it is of particular interest to report their modest conclusions:

> *It is clear that we have left unanswered practically any question one might pose of a theoretical nature concerning the traveling-salesman problem; however, we hope that the feasibility of attacking problems involving a moderate number of points has been successfully demonstrated, and that perhaps some of the ideas can be used in problems of similar nature.*

Indeed, not only the methodological approach of DFJ is the backbone of the current state-of-the-art solution method for the TSP, but it also anticipated several key ideas that later became central to modern mixed-integer optimization. The DFJ approach would now be called a primal *cutting plane method*, later formalized in the 1960s, and inspired the dual cutting plane method introduced by Gomory in 1958. Furthermore, their use of reduced cost arguments (only briefly mentioned) anticipated modern *reduced cost fixing* techniques used for pruning and tightening mixed-integer models. These innovations collectively provided the foundations for the development of the *branch-and-bound* method (Land and Doig, 1960) and later the *branch-and-cut* method, both of which are the cornerstone of contemporary integer programming solvers.

Currently, the largest TSP instance solved consists of a problem with 85900 cities, related to a VLSI (Very Large-Scale Integration) chip design application. The instance was solved by Applegate et al. (2009) in 2006 using the Concorde code, an advanced software based on the branch-and-cut technique.

Chapters 2 and 3 address two variants of the TSP, respectively, the *Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds*, and the *Capacitated Vehicle Routing Problem*. Finally, Chapter 4 addresses a *Routing and Wavelength Assignment Problem*, a routing problem in optical networks. The following sections give an introduction to these three works.

1.1 Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds

Chapter 2 investigates the *Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds* (FSTSP-VDS), a novel extension of the classical TSP which reflects the realities of emerging hybrid delivery systems combining trucks and Unmanned Aerial Vehicles (UAVs or drones). The motivation for studying the FSTSP-VDS is to improve the efficiency of last-mile delivery, the most time-consuming and expensive stage of logistics, in terms of delivery times, operational costs, gas emissions, and customer satisfaction.

The FSTSP-VDS models a last-mile delivery system where the two vehicles, a truck and a drone, cooperate together with the aim of serving a set of customers in the shortest time. The truck transports the drone, acting as a mobile depot to launch and retrieve it, extending the drone's reach far beyond its flight range, and while the drone is flying, the truck can serve other customers. Drones are typically faster and are not restricted to the road network or affected by traffic; however, their flight range and payload capacity are much more limited compared to trucks. The FSTSP is defined on a complete directed graph $\mathcal{G} = (V, A)$. The vertex set V is defined as $V = \{0, 0'\} \cup N$, where both 0 and $0'$ represent a single depot and $N = \{1, 2, \dots, n\}$ a set of n customers to serve. Each customer $i \in N$, requires a parcel of weight w_i that must be delivered either by the truck or by the drone, initially located at the depot. The drone can carry only one parcel per trip and has a payload capacity W_{max}. Therefore, any parcel with a weight exceeding W_{max} must be delivered by the truck.

Unlike the standard Flying Sidekick Traveling Salesman Problem (FSTSP), which assumes a constant drone speed and range, in the FSTSP-VDS, the drone's flight speed is treated as a continuous decision variable. This extension allows for a realistic modeling of energy consumption, which varies non-linearly with respect to speed and payload due to the aerodynamic drag. We demonstrate that if the drone's energy-per-meter function (i.e., the energy consumed by the drone to travel horizontally one meter) is convex with respect to speed, it is possible to model the problem as a Mixed Integer Linear Programming (MILP) problem, even if the drone's power consumption is non-convex with respect to speed.

Based on that, we describe the first exact method for the FSTSP-VDS, capable of solving to optimality instances with up to ten customers. The findings reveal that optimizing the drone speed can reduce completion times by 32% on average, offering strong evidence for the importance of dynamic speed control in drone-assisted logistics. To handle larger instances, a genetic algorithm is proposed, showing significant improvements over existing heuristics.

1.2 Capacitated Vehicle Routing Problem

Chapter 3 focuses on the *Capacitated Vehicle Routing Problem* (CVRP), a generalization of the TSP, first introduced as the *truck dispatching problem*, by Dantzig and Ramser (1959). The CVRP is a fundamental problem in operations research, as it models many real-world applications that involve delivery, collection, or transportation with vehicle capacity limits.

The CVRP is defined on a complete undirected graph $\mathcal{G} = (V, E)$, where V is the node set defined as $V = \{0\} \cup N$, 0 represents the depot and $N = \{1, 2, \dots, n\}$ a set of $n \in \mathbb{N}$ customers to serve, each with a known positive demand $q_i, i \in [1, n]$. A non-negative traveling cost $c_{ij} = c_{ji} \geq 0$ is associated to each edge $(i, j) \in E$. A fleet of identical vehicles with a capacity Q are located at the depot to fulfill customer demands. A feasible vehicle route $R = (0, v_1, v_2, ..., v_r, 0)$ with $r \in \mathbb{N}$, is a simple tour in $\mathcal{G}$ starting from the depot, visiting nodes $v_1, v_2, ..., v_r$ and returning back to the depot such that the total demand of the visited customers does not exceed the vehicle capacity Q. The CVRP asks to find a set of feasible routes that minimize the overall traveling cost such that each customer is served exactly once.

Chapter 3 addresses the following question. Is it possible to concurrently optimize different parts of a CVRP solution, such that the recombination of the optimized partial solutions is still a feasible solution? The answer is clearly affirmative if the partial solutions to be optimized consist of a disjoint subset of routes, as it happens in the *route-based decomposition scheme*. Otherwise, the answer is unknown. This work addresses this question and unveils a counterintuitive property of the CVRP. A new decomposition scheme based on this property is proposed, named *sequence-based decomposition*.

1.3 Routing and Wavelength Assignment Problem

Modern global communication relies on high-capacity, high-speed backbone infrastructures provided by optical fiber networks. These systems leverage *Wavelength Division Multiplexing* (WDM) technology, which allows multiple data streams to be transmitted simultaneously over a single fiber by using distinct optical frequencies (wavelengths).

Chapter 4 focuses on the *Routing and Wavelength Assignment* (RWA) problem under a *shared protection policy*, an optimization problem concerned with allocating wavelengths in an optical network to serve a set of connection requests while ensuring robustness against single-link failures. Two edge-disjoint paths must be assigned to each connection, or commodity: a *working path*, used under normal operating conditions, and a *recovery path*, activated when a failure occurs on the working path. The classical *dedicated path*

protection scheme, which reserves exclusive resources for each connection, guarantees survivability but may be highly inefficient. Conversely, the *global re-routing* scheme maximizes efficiency but introduces complex control requirements and potential service disruptions. To balance these aspects, the *shared protection* scheme allows to share wavelengths among different connections, provided that no interference arises under any single-link failure scenario.

The RWA problem addressed is defined on an undirected multigraph $\mathcal{G} = (V, E)$ where V is a set of nodes and E is a multiset of edges, representing an existing optical fiber network. Each edge $e \in E$ corresponds to an optical fiber connection with capacity u, expressed as the number of different available wavelengths, and parallel edges (if any) represent alternative independent links between pairs of connected nodes. A set $\mathcal{K}$ of origin-destination pairs defines the connection requests to be served. The assignment of a wavelength of an edge $e \in E$ induces a positive cost ℓ_e. The problem asks to find a minimum cost routing that allows to serve all connection requests.

We discuss alternative relaxations for the problem based on Integer Linear Programming formulations, and introduce a metaheuristic algorithm based on the iterated local search paradigm. Computational experiments on realistic data show that our algorithms are able to produce feasible solutions with tight optimality gaps.

Chapter 2

Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds

[1]

Thanks to technological advances, many last-mile delivery companies have launched projects that combine unmanned aerial vehicles, commonly known as drones, with classic ground vehicles to reduce delivery times and gas emissions and improve customer satisfaction. However, most drone-aided optimization problems in the literature assume a non-realistic scenario in which the drone speed is constant and the drone endurance is independent of its speed and payload. This work addresses this gap by considering a realistic non-linear non-convex drone power consumption model with variable drone speeds. We propose an exact and heuristic approach for the *Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds* (FSTSP-VDS), an extension of the flying sidekick traveling salesman problem in which a truck is combined with a drone that can fly at variable speeds to deliver parcels to customers to minimize the makespan. If the drone energy-per-meter function is convex, the FSTSP-VDS can be formulated as a *Mixed Integer Linear Programming* (MILP) problem. We propose a MILP model that can solve to optimality instances with up to ten customers. We also show that, when a realistic drone energy model is used, fixing the drone speed can increase the average makespan by 32%. We also propose a genetic algorithm that outperforms the current state-of-the-art by 7% on average and finds optimal solutions for constant-speed instances with up to 99 customers.

[1] This chapter is based on Michelotto and Roberti (2025)

2.1 Introduction

Over the past few years, there has been a growing interest in delivery systems that include *Unmanned Aerial Vehicles* (UAV) such as drones. For example, large delivery companies (e.g., UPS and Amazon) started several projects about using drones in last-mile delivery to reduce delivery times and gas emissions and improve customer satisfaction (see, e.g., Banker (2022)). For these reasons, the research community is studying different routing problems in which conventional vehicles are supported by one or more drones. The first problem of this category is the *Flying Sidekick Traveling Salesman Problem* (FSTSP) introduced by Murray and Chu (2015), an extension of the popular *Traveling Salesman Problem* (TSP) in which a truck is coupled with a drone to serve a set of customers while minimizing the makespan, i.e., the overall completion time.

A common assumption of most drone-aided routing problems, as in the FSTSP, is that the drone endurance is constant regardless of its speed and payload (see Macrina et al. (2020) and Liang and Luo (2022) for an overview). This hypothesis wrongly suggests that increasing the drone flying speed to the maximum allowable speed has only beneficial effects, e.g., increasing the drone range. In reality, because of the air resistance (drag), the power consumption is a non-linear function with respect to the vehicle speed. Therefore, when dealing with more realistic drone power consumption models, changing (not only reducing) the drone flying speed could increase the drone range and endurance and serve more customers with the drone, or serve the customers faster, reducing the makespan.

In this paper, we study the *Flying Sidekick Traveling Salesman Problem with Variable Drone Speeds* (FSTSP-VDS) proposed by Raj and Murray (2020) in which, unlike the FSTSP, the drone speed is not fixed a priori but can be selected from a continuous set for each arc traversed by the drone. However, Raj and Murray (2020) consider the general case with one or more drones, whereas we focus on the single-drone case.

Our main scientific contributions are the following:

- we show that if the drone energy-per-meter function is convex with respect to the drone speed, the minimum and maximum travel time to launch the drone, serve a customer, and retrieve the drone at a rendezvous location, can be computed by solving a convex optimization problem;
- we present a compact *Mixed Integer Linear Programming* (MILP) formulation for the FSTSP-VDS under the condition that the drone energy-per-meter function is convex. This formulation can solve instances with ten customers when cast within a general-purpose solver;

- we show that, when a realistic drone energy model is used, fixing the drone speed produces a substantial increase in the makespan up to 32% on average;
- we propose a genetic algorithm for the FSTSP-VDS that outperforms the state-of-the-art heuristic by a significant margin and when the drone speed is constant, finds optimal or near-optimal solutions (on average within 0.26% from the optimal ones) on instances with up to 99 customers (the largest instances tested).

The remainder of the paper is organized as follows. Section 2.2 reviews the literature about drone-aided routing problems related to our work. Section 2.3 formally defines the FSTSP-VDS. Section 2.4 investigates the properties of convex energy-per-meter functions. Section 2.5 presents a compact MILP formulation for the FSTSP-VDS. Section 2.6 describes our genetic algorithm. Section 2.7 reports the computational results of the compact MILP formulation and the genetic algorithm. Section 2.8 discusses some conclusions.

2.2 Literature Review

The recent growing interest of the research community in drone-aided routing problems does not allow a complete review of all scientific papers on this topic. Thus, we limit our review to papers about the FSTSP or the *Traveling Salesman Problem with Drone* (TSP-D). For an exhaustive review of drone-aided routing problems, we refer the reader to the surveys of Macrina et al. (2020) and Liang and Luo (2022). In the next sections, we summarize the relevant works on the FSTSP and the TSP-D, respectively, following the definitions of Murray and Chu (2015) for the FSTSP and Agatz et al. (2018) for the TSP-D, who first defined the two problems. The FSTSP is often referred to as the basic TSP-D.

2.2.1 The Flying Sidekick Traveling Salesman Problem

The FSTSP is the first drone-aided routing problem studied in the literature (Murray and Chu (2015)). The FSTSP is an extension of the classical TSP in which a truck is coupled with a fixed-speed drone to serve a set of customers and return both vehicles to the depot in the minimum time, including potential waiting times related to the truck and drone synchronization. In particular, the drone can be launched from the truck at a customer location or the depot only, serve a customer, and then return to the truck at a different location, with the constraint that if the drone arrives before the truck at the rendezvous location, it cannot land and wait, but it must hover until the arrival of the truck, hence, consuming energy. Some customers may be unsuitable for drone delivery, for example, because the parcel is too heavy or because the customer location does not

allow a safe landing, so some customers can only be served by truck. In addition, each node must be visited exactly once, and the time to launch and retrieve the drone and the time to deliver the packages must be considered.

Murray and Chu (2015) propose a MILP formulation and a heuristic for the FSTSP. Ha et al. (2018) propose a greedy randomized adaptive search procedure based on a procedure that optimally splits a TSP solution into an FSTSP solution, given the relative order between customers. Es Yurek and Ozmutlu (2018) propose an iterative decomposition-based approach. de Freitas and Penna (2020) propose a variable neighborhood search heuristic algorithm. Ha et al. (2020) propose a hybrid genetic heuristic. Several exact methods based on MILP formulations have been proposed, e.g., branch-and-bound (Dell'Amico et al. (2021*a*), Dell'Amico et al. (2021*b*), Dell'Amico et al. (2022)), column-and-row generation (Boccia et al. (2021)), branch-and-price (Roberti and Ruthmair (2021)), branch-and-cut (Boccia et al. (2023)) and column-generation-based (Blufstein et al. (2024)).

2.2.2 The Traveling Salesman Problem with Drone

The TSP-D proposed by Agatz et al. (2018) differs from the FSTSP. Indeed, a node can be visited more than once, the drone can be launched and retrieved in the same location, the drone can land and wait, and drone launches, retrieval times, and parcel delivery times are neglected. Examples of heuristic approaches are the one of Agatz et al. (2018), based on a split procedure similar to the one of Ha et al. (2018), and the variable neighborhood search of El-Adle et al. (2023). Exact approaches for the TSP-D are based on dynamic programming (Bouman et al. (2018)), branch-and-bound (Agatz et al. (2018), Poikonen et al. (2019)), branch-and-cut (Schermer et al. (2020)), branch-and-price (Roberti and Ruthmair (2021)), and Benders decomposition (Vásquez et al. (2021)).

2.2.3 Non-Constant Drone Speed

In this section, we discuss the works related to drone-aided routing problems in which the drone speed is not fixed a priori. Raj and Murray (2020) introduce the FSTSP-VDS with multiple drones, an extension of the FSTSP in which drones are allowed to fly at different speeds, selected from a continuous set; the drone energy consumption is characterized by the model of Liu et al. (2017), and the authors propose a three-phase heuristic. Campuzano et al. (2023) introduce the drone-assisted variable speed asymmetric traveling salesman problem in which flight times depend not only on the payload weight but also on the drone speed and the weather conditions. The drone energy consumption is computed using the model of Liu et al. (2017), and the drone speed can be selected from a discrete set of three speed levels. The authors propose

a MILP formulation and two heuristics based on variable neighborhood search and tabu search. Tamke and Buscher (2023) introduces the vehicle routing problem with drones and drone speed selection, a variant of the classical vehicle routing problem in which trucks are equipped with multiple drones to minimize the overall operational cost. The drone speeds are not fixed but have to be selected from a discrete set of five speed levels, and the drone energy consumption is modeled as in Stolaroff et al. (2018). The authors propose a MILP formulation.

To our knowledge, there are no exact approaches in the literature for drone-aided routing problems in which a non-linear model characterizes the drone energy consumption and the drone speed can be selected from a continuous set.

2.3 Problem Description

The FSTSP-VDS can be formally described as follows. A complete directed graph $\mathcal{G} = (V, A)$ is given. The vertex set V is defined as $V = \{0, 0'\} \cup N$, where both 0 and $0'$ represent a single depot (respectively the starting and final location of truck-and-drone route) and $N = \{1, 2, \dots, n\}$ a set of n customers to serve. A single truck, equipped with a single drone, is located at the depot. Each customer $i \in N$ must be served exactly once, either by the truck or the drone. The arc set A is defined as $A = \{(i, j) \,|\, i, j \in N : i \neq j\} \cup \{(0, j) \,|\, j \in N\} \cup \{(i, 0') \,|\, i \in N\}$. In the following, we use the notation $N_0 = N \cup \{0\}$ and $N_0' = N \cup \{0'\}$.

A parcel of weight w_i must be delivered to customer $i \in N$. The drone can carry only one parcel at a time and has a payload capacity W_{max}, so parcels of weight greater than W_{max} must be delivered by the truck. The discrete set of payloads that can be carried by the drone is defined as $\mathcal{W} = \{0\} \cup \{w_i \leq W_{max}, i \in N\}$. The drone can leave and return to the truck at customer locations or the depot only, but it cannot be launched and retrieved at the same node. While the drone is flying or serving a customer, the truck can serve other customers. The drone can traverse different arcs at different flying speeds, but the flying speed must remain constant during the flight and be in the interval $[v^D_{min}, v^D_{max}]$. The distance that the truck (drone, resp.) must cover to traverse arc $(i, j) \in A$ is indicated with d^T_{ij} (d^D_{ij}, resp.) – typically, $d^T_{ij} \neq d^D_{ij}$ as the truck and the drone travel on different pathways. The time for the truck to traverse arc $(i, j) \in A$ is indicated with t^T_{ij}. The minimum and maximum feasible drone travel times to takeoff from node $i \in N_0$, deliver the parcel at node $j \in N$, and land at node $k \in N_0'$ (*drone leg* in the following), including potential hovering time over the landing node k, are denoted respectively with t^D_{ijk} and T^D_{ijk}, and are set to $+\infty$ in case of infeasibility. The set of all feasible drone legs is defined by $Z = \{(i, j, k) \,|\, i \in N_0, j \in N, k \in N_0' : t^D_{ijk} \in \mathbb{R}\}$.

The drone has a fixed battery capacity of B Joules, and the drone endurance, expressed in seconds, is a function of the parcel weight and the drone's speed. A fully charged battery is installed in the drone before every launch. The parameter h, expressed in meters, defines the drone cruise altitude. Drone takeoff and landing operations consist of vertical flights at a constant speed, respectively equal to v_t^D and v_l^D. We refer the reader to Section 2.3.1 for a complete description of the drone power consumption model employed in this work. The drone must rejoin the truck at a landing location before it runs out of battery. If the drone arrives before the truck at the rendezvous location, the drone cannot land and wait, but it must hover above the rendezvous location until the truck arrives, consuming energy. Vice versa, if the truck arrives before the drone, the truck must wait for the drone. Before the takeoff, the drone must be prepared, and this operation takes s_L seconds. When the drone lands at the rendezvous location, it takes s_R seconds to retrieve the drone. Hence, for each drone takeoff and landing, the additional time required to prepare and retrieve the drone must be considered. The time to deliver a parcel, once the vehicle arrives at a customer location $i \in N$, equals σ_i^T seconds for the truck and σ_i^D seconds for the drone. Without loss of generality, we define $\sigma_0^T = \sigma_{0'}^T = \sigma_0^D = \sigma_{0'}^D = 0$. Parcel delivery at rendezvous locations can occur before or after the truck and drone synchronization. Similarly, parcel delivery at takeoff locations can occur before or after the drone launch. Hence, if a location is both a rendezvous and a takeoff location, the parcel delivery can be done in three different ways: (i) before the truck and drone synchronization, (ii) after the drone launch, or (iii) after the truck and drone synchronization but before the drone launch. An illustrative example of the three possible delivery scenarios is depicted in Figure 2.1. In all cases, customers visited by the truck and the drone are always served by the former.

The goal of the FSTSP-VDS is to find a truck-drone tour of minimum makespan serving all customers either by truck or drone, taking into account potential waiting times due to the synchronization between the two vehicles.

In the following, we will use the following definitions. A *truck customer* (*drone customer*, resp.) is a customer visited by the truck (drone, resp.) alone. On the other hand, if a customer is visited by both the truck and the drone, we call it *combined customer*. A *truck arc* (*drone arc*, resp.) is traversed by the truck (drone, resp.) alone. A *combined arc* is traversed by the truck while the drone is onboard. A *truck leg* is a concatenation of truck arcs traversed by the truck alone in between two consecutive combined customers. A *drone leg* is a concatenation of exactly two consecutive drone arcs traversed by the drone alone between two consecutive combined customers. A *combined leg* is a concatenation of combined arcs traversed by the truck while the drone is onboard that consists of combined customers only. An *operation* consists of a truck leg and a drone leg in between the same pair of combined customers. Notice that a FSTSP-VDS solution is a

concatenation of operations and combined legs.

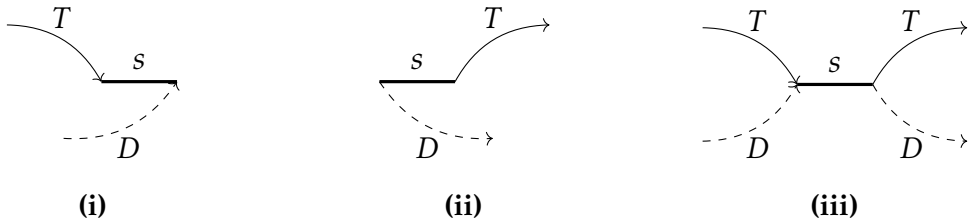


Figure 2.1: Timelines of the three possible ways of serving a combined customer, where s denotes the delivery operation, T the truck arrival, and D the drone arrival: from left to right, before the truck and drone synchronization, after the drone launch, and after the truck and drone synchronization but before the drone launch.

2.3.1 Drone Endurance

In this work, the drone endurance is characterized by the power consumption model of Liu et al. (2017), developed for small multi-rotor unmanned aircraft systems – the same model employed in the work of Raj and Murray (2020). The model characterizes the drone power consumption with respect to speed v (in m/s) and payload weight w (in kg), during the three flight stages of the drone, i.e., during the vertical takeoff or landing (2.1a), the horizontal cruising (2.1b), and the stationary hovering (2.1c):

$$P^{TL}(v,w) = k_1(D+w)g\left[\frac{v}{2} + \sqrt{\left(\frac{v}{2}\right)^2 + \frac{(D+w)g}{k_2^2}}\right] + c_2\left((D+w)\,g\right)^{3/2} \tag{2.1a}$$

$$P^{C}(v,w) = (c_1+c_2)\left[\left((D+w)g - c_5\left(v\cos\theta\right)^2\right)^2 + \left(c_4v^2\right)^2\right]^{3/4} + c_4v^3 \tag{2.1b}$$

$$P^{H}(w) = (c_1+c_2)\left((D+w)\,g\right)^{3/2} \tag{2.1c}$$

The coefficients k_1, k_2, c_1, c_2, c_4, c_5 are the parameters of the power consumption model of Liu et al. (2017), and their values can be found in Table 2.1. The other parameters, D and θ, define the physical characteristics of the drone. In particular, D is the drone frame weight, which is assumed to be 1.5 kg, and θ is the angle of attack of the drone, which equals 10 degrees. Finally, g is the gravitational acceleration (9.8 m/s^2).

Coefficient	k_1	k_2	c_1	c_2	c_4	c_5
Value	0.8554	0.3051	2.8037	0.3177	0.0296	0.0279
Unit	-	$\sqrt{\text{kg/m}}$	$\sqrt{\text{m/kg}}$	$\sqrt{\text{m/kg}}$	kg/m	N s/m

Table 2.1: Coefficient values for the power consumption model of Liu et al. (2017).

The Liu et al. (2017) drone power consumption during horizontal cruising and the corresponding drone range, with respect to different speeds and parcel weights, are illustrated in Figure 2.2 and Figure 2.3. Figure 2.2 shows that the drone power consumption $P^C(v, w)$ is non-linear and non-convex; indeed, consider the speeds 1.0 m/s and 6.0 m/s and a payload of 0.0 kg, we have $P^C(\frac{1}{2}(1.0 + 6.0), 0.0) = 171.36\ W > \frac{1}{2}(P^C(1.0, 0.0) + P^C(6.0, 0.0)) = 170.66\ W$.

However, since the FSTSP-VDS requires a constant drone flying speed during each flight, we are interested in studying how much energy is consumed by the drone to travel horizontally one meter at a given speed – energy-per-meter in the following – and how this quantity varies with respect to the drone speed. Figure 2.4 shows the drone energy-per-meter consumption, defined as $E^{PM}(v, w) = (1/v)P^C(v, w)$, with respect to speed v for different payload weights w.

In Sections 2.4 and 2.5, we show that, if the energy-per-meter function $E^{PM}(v, w)$ is convex for each parcel weight $w \in \mathcal{W}$, the FSTSP-VDS can be formulated as a compact MILP.

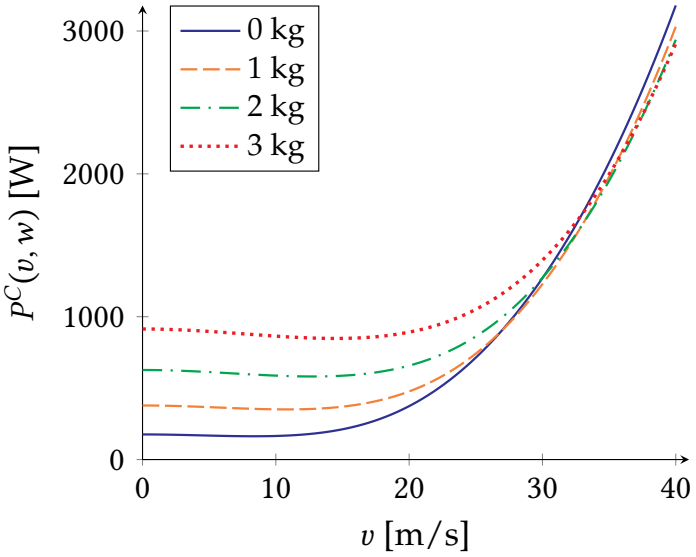


Figure 2.2: Liu et al. (2017) drone power consumption with respect to the drone speed v, for different payload weights w.

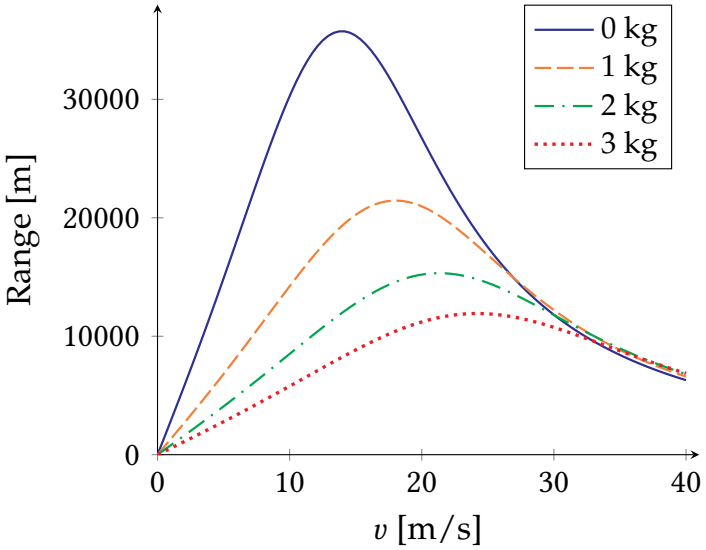


Figure 2.3: Liu et al. (2017) drone range in meters, computed as $vB/P^C(v, w)$, with respect to the drone speed v, for different payload weights w, given a battery with capacity $B = 500$ kJ.

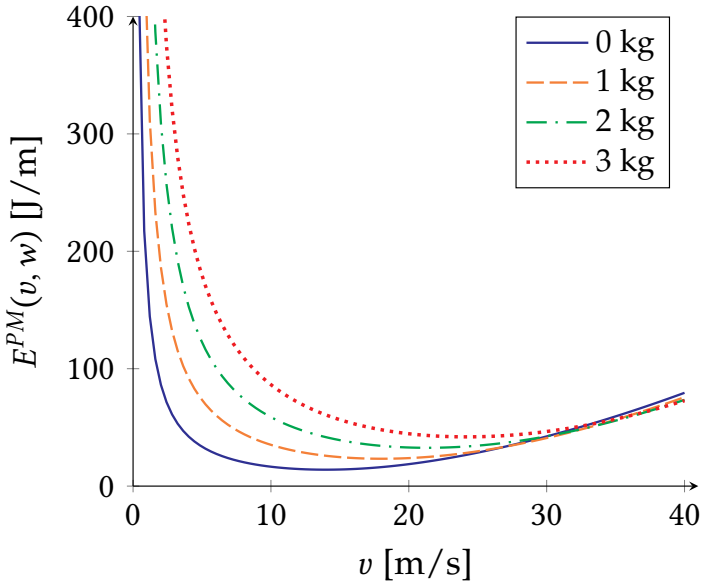


Figure 2.4: Liu et al. (2017) energy-per-meter function $E^{PM}(v, w)$ with respect to the drone speed v, for different payload weights w.

2.3.2 Example

Figure 2.5 illustrates a solution of a FSTSP-VDS instance with five customers (the circles) and the depot, that is, nodes 0 and 0′ (the rectangle). In the example, truck arcs (drone arcs, resp.) are depicted by solid (dashed, resp.) lines and combined arcs by double lines. Initially, the truck moves with the drone onboard towards customer 3 taking time t^T_{03}. Then, the drone is launched to serve customer 2 while the truck moves towards

customer 1. After serving customer 2, the drone moves towards customer 1 and rejoins with the truck. The drone leg travel time must be between t^D_{321} and T^D_{321}. The truck leg travel time $t^T_{3\to1}$ depends on when customers 3 and 1 are served. In particular, if the parcel delivery to customer 3 occurs before the drone launch and customer 1 is served after the drone and truck synchronization, $t^T_{3\to1}$ equals t^T_{31}. If instead customer 3 is served after the drone launch, σ^T_3 must be added to t^T_{31}. Moreover, if customer 1 is served before the drone and truck synchronization, σ^T_1 must also be added. After the truck and the drone synchronization, the drone is launched to serve customer 5 while the truck serves customer 4. Finally, the truck and drone return to the depot and rejoin. As before, the drone leg travel time must be between $t^D_{150'}$ and $T^D_{150'}$, and the truck leg travel time $t^T_{1\to0'}$ depends on when customer 1 is served. If customer 1 is served before the drone launch, $t^T_{1\to0'}$ equals $t^T_{14}+\sigma^T_4+t^T_{40'}$. Otherwise, if customer 1 is served after the drone launch, $t^T_{1\to0'}$ equals $\sigma^T_1+t^T_{14}+\sigma^T_4+t^T_{40'}$.

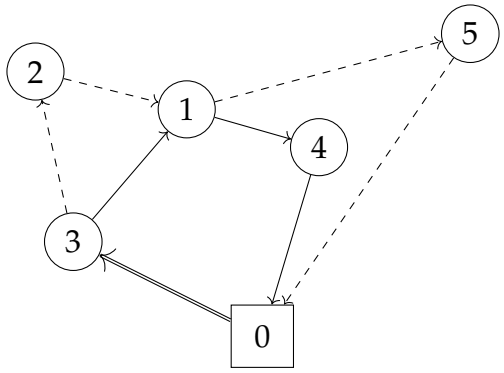


Figure 2.5: A solution of a FSTSP-VDS instance with five customers.

Below we list the six possible completion times depending on when combined customers 3 and 1 are served. In particular, T_1, T_2, T_3 (T_4, T_5, T_6, resp.) correspond to the completion times when customer 3 is served before (after) the drone launch, and customer 1 is served respectively, after synchronization but before the drone launch, before synchronization, and after the drone launch:

$$
\begin{aligned}
T_1 &= t^T_{03}+\sigma^T_3+\max\{t^D_{321},t^T_{31}\}+\sigma^T_1+\max\{t^D_{150'},t^T_{14}+\sigma^T_4+t^T_{40'}\}+2(s_L+s_R)\\
T_2 &= t^T_{03}+\sigma^T_3+\max\{t^D_{321},t^T_{31}+\sigma^T_1\}+\max\{t^D_{150'},t^T_{14}+\sigma^T_4+t^T_{40'}\}+2(s_L+s_R)\\
T_3 &= t^T_{03}+\sigma^T_3+\max\{t^D_{321},t^T_{31}\}+\max\{t^D_{150'},\sigma^T_1+t^T_{14}+\sigma^T_4+t^T_{40'}\}+2(s_L+s_R)\\
T_4 &= t^T_{03}+\max\{t^D_{321},\sigma^T_3+t^T_{31}\}+\sigma^T_1+\max\{t^D_{150'},t^T_{14}+\sigma^T_4+t^T_{40'}\}+2(s_L+s_R)\\
T_5 &= t^T_{03}+\max\{t^D_{321},\sigma^T_3+t^T_{31}+\sigma^T_1\}+\max\{t^D_{150'},t^T_{14}+\sigma^T_4+t^T_{40'}\}+2(s_L+s_R)\\
T_6 &= t^T_{03}+\max\{t^D_{321},\sigma^T_3+t^T_{31}\}+\max\{t^D_{150'},\sigma^T_1+t^T_{14}+\sigma^T_4+t^T_{40'}\}+2(s_L+s_R)
\end{aligned}
$$

Serving a combined customer with the drone on the ground is never more convenient than serving it while the drone is airborne, but when the drone's endurance is finite, the latter case could be not feasible whereas the former case could be.

2.4 Drone Travel Times Computation

In this section, we show that if the energy-per-meter function is convex for each $w \in \mathcal{W}$, the set of feasible drone leg travel times of each drone leg $(i, j, k) \in Z$ is also convex. Thereby, we can forget about drone speeds and bound the minimum and maximum feasible drone leg travel times; this allows us to formulate the FSTSP-VDS as a MILP. Then, we present two convex optimization problems to compute respectively, the minimum and the maximum drone leg travel times of a generic drone leg $(i, j, k) \in Z$, respectively, t^D_{ijk} and T^D_{ijk}. From now on, we assume that the energy-per-meter function is convex for each $w \in \mathcal{W}$. For the sake of clarity, we ignore w.l.o.g. the constant quantities related to the takeoff and landing operations (s_L and s_R), and the time required to serve the drone customer (σ^T_j).

Let us introduce the following set of constraints:

$$E^{PM}(v^D_{ij}, w_j)\, d^D_{ij} + E^{PM}(v^D_{jk}, 0)\, d^D_{jk} + P^H(0)\, t_h \leq B \tag{2.2a}$$

$$t_h \geq 0 \tag{2.2b}$$

$$v^D_{ij}, v^D_{jk} \in [v^D_{min}, v^D_{max}] \tag{2.2c}$$

Constraint (2.2a) imposes that the drone energy consumption to traverse the drone leg (i, j, k) at speeds v^D_{ij} and v^D_{jk} plus the energy required to hover for a time t_h seconds must not exceed the battery capacity of B Joules. Constraints (2.2b) and (2.2c) model the feasible range of the variables v^D_{ij}, v^D_{jk}, and t_h. Since the left-hand side of (2.2a) is a sum of convex functions, the feasible region of (2.2) is convex.

Therefore, to compute t^D_{ijk} one can solve the convex optimization problem corresponding to minimizing $d^D_{ij}/v^D_{ij} + d^D_{jk}/v^D_{jk} + t_h$ subject to (2.2). Furthermore, since the feasible region of (2.2) is convex, the set of feasible drone leg travel times of the above convex optimization problem is also convex. To compute T^D_{ijk}, we propose the following convex optimization problem. The total energy consumed by the drone to travel horizontally d meters in t seconds with a payload of w kg, $E^d(t, w)$ in the following, can be expressed as follows:

$$E^d(t, w) = t\, P^C(d/t, w) = d\,(t/d)\, P^C(d/t, w) = d\, E^{PM}(d/t, w) \tag{2.3}$$

Now observe that the function $g(x) = \frac{1}{x}, x > 0$ is a strict order reversing map (or monotone decreasing function), i.e., $\forall x_1, x_2 > 0 : x_1 < x_2 \implies g(x_1) > g(x_2)$. Therefore, if $E^{PM}(x, w)$ is convex in the interval $x \in [v^D_{min}, v^D_{max}]$ for any $w \in \mathcal{W}$, $E^{PM}(d/x, w)$ is convex in the interval $x \in [d/v^D_{max}, d/v^D_{min}]$ for any $w \in \mathcal{W}$. Consequently, $E^d(x, w)$ is convex in the interval $x \in [d/v^D_{max}, d/v^D_{min}]$ for any $w \in \mathcal{W}$. From the above observations, we define the

following convex optimization problem to compute T^D_{ijk}:

$$\max\ t^D_{ij} + t^D_{jk} + t_h \tag{2.4a}$$
$$E^{d^D_{ij}}(t^D_{ij}, w_j) + E^{d^D_{jk}}(t^D_{jk}, 0) + t_h\, P^H(0) \leq B \tag{2.4b}$$
$$t^D_{ij} \geq d^D_{ij}/v^D_{max} \tag{2.4c}$$
$$t^D_{jk} \geq d^D_{jk}/v^D_{max} \tag{2.4d}$$
$$t_h \geq 0 \tag{2.4e}$$

The objective function (2.4a) aims at maximizing the sum of the travel times to traverse the arcs (i, j) and (j, k), in t^D_{ij} and t^D_{jk} seconds respectively, plus the potential hovering time of t_h seconds over node k. Constraint (2.4b) ensures that the energy consumption is not greater than B Joules. Constraints (2.4c) - (2.4e) model the range of the decision variables t^D_{ij}, t^D_{jk} and t_h.

Finally, we propose the following convex optimization problem to find the minimum energy consumption to traverse a drone leg $(i, j, k) \in Z$ in a target travel time of T seconds:

$$\min\ E^{d^D_{ij}}(t^D_{ij}, w_j) + E^{d^D_{jk}}(t^D_{jk}, w_j) + (T - t^D_{ij} - t^D_{jk})P^H(0) \tag{2.5a}$$
$$t^D_{ij} + t^D_{jk} \leq T \tag{2.5b}$$
$$t^D_{ij} \geq d^D_{ij}/v^D_{max} \tag{2.5c}$$
$$t^D_{jk} \geq d^D_{jk}/v^D_{max} \tag{2.5d}$$

The objective function (2.5a) aims at minimizing the energy to traverse the arcs (i, j) and (j, k), in t^D_{ij} and t^D_{jk} seconds respectively, plus the energy required to hover over node k for $T - t^D_{ij} - t^D_{jk}$ seconds. Constraint (2.5b) ensures that the travel time to arrive in k must be not greater than T. Constraints (2.5c) and (2.5d) define the range of the decision variables t^D_{ij} and t^D_{jk}.

The models outlined in this section allow determining in pre-processing a range $[t^D_{ijk}, T^D_{ijk}]$ of feasible drone leg travel times for each feasible drone leg $(i, j, k) \in Z$. In Section 2.5, we show how such ranges can be used to build a MILP model for the FSTSP-VDS when the drone energy-per-meter function is convex.

2.5 Compact Formulation

In this section, we describe a compact formulation for the FSTSP-VDS, under the assumption that the energy-per-meter is convex. The resulting formulation is compact (i.e., polynomial number of constraints and variables) and can be solved with a generic MILP solver such as CPLEX or Gurobi.

Let $x_{ij}^T \in \{0,1\}$ be a binary variable equal to 1 if the truck traverses the arc $(i,j) \in A$, and let $x_{ij}^D \in \{0,1\}$ be a binary variable equal to 1 if the drone traverses the arc $(i,j) \in A$ (no matter if it is onboard truck or airborne). Let $y_i^T, y_i^D, y_i^C \in \{0,1\}$ be three binary variables equal to 1 if $i \in N$ is a truck, drone, or combined customer, respectively. Let $z_{ijk} \in \{0,1\}$ be a binary variable equal to 1 if the drone traverses the drone leg $(i,j,k) \in Z$.

Let $\alpha_i \in \{0,1\}$ be a binary variable equal to 1 if the truck delivery at node $i \in V$ occurs right after the truck arrival and before the truck and drone synchronization if i is a rendezvous node (case 2.1(i)). Let $\beta_i \in \{0,1\}$ be a binary variable equal to 1 if the truck delivery at node $i \in V$ occurs after the drone launch – an operation must start from i – (case 2.1(ii)). Let $\gamma_i \in \{0,1\}$ be a binary variable equal to 1 if the truck delivery at node $i \in V$ occurs after the synchronization between the truck and drone and before the drone launch if i is a takeoff node – an operation must end in i – (case 2.1(iii)). Finally, let $a_i \in \mathbb{R}_+$ be a real variable denoting the latest arrival time between the truck and drone at node $i \in V$, including the time to serve i if $\alpha_i = 1$, without considering the time required to prepare the launches and retrievals of the drone and the parcel deliveries occurred in between successive operations. If i is a drone customer, a_i is not defined.

For clarity, we summarize here the model parameters introduced in Section 2.3: the truck travel time t_{ij}^T to traverse arc $(i,j) \in A$; the minimum and maximum drone leg travel times t_{ijk}^D, T_{ijk}^D, for each feasible drone leg $(i,j,k) \in Z$ (including takeoff at node i, landing at node j, serving customer j, takeoff at node j and landing at node k); the minimum and maximum drone flying speed v_{min}^D, v_{max}^D; the time required to launch and retrieve the drone s_L, s_R; and the time required to serve a customer $i \in N$ with the truck and with the drone σ_i^T, σ_i^D. Finally, the parameter M (set equal to the minimum truck-only makespan) is used to disable a constraint if a binary decision variable is not triggered.

The FSTSP-VDS can be formulated as follows:

$$\min \quad a_{0'} + (s_L + s_R) \sum_{i \in N} y_i^D + \sigma_i^T \sum_{i \in N} \gamma_i, \tag{2.6a}$$

$$\sum_{(i,j) \in A} x_{ij}^T = \sum_{(j,i) \in A} x_{ji}^T = y_i^T + y_i^C \qquad i \in N \tag{2.6b}$$

$$\sum_{(0,j) \in A} x_{0j}^T = \sum_{(i,0') \in A} x_{i0'}^T = 1 \tag{2.6c}$$

$$\sum_{(i,j) \in A} x_{ij}^D = \sum_{(j,i) \in A} x_{ji}^D = y_i^D + y_i^C \qquad i \in N \tag{2.6d}$$

$$\sum_{(0,j) \in A} x_{0j}^D = \sum_{(i,0') \in A} x_{i0'}^D = 1 \tag{2.6e}$$

$$y_i^T + y_i^D + y_i^C = 1 \qquad i \in N \tag{2.6f}$$

$$y_i^C = \alpha_i + \beta_i + \gamma_i \qquad i \in N \tag{2.6g}$$

$$\beta_i \leq \sum_{(i,j,k) \in Z} z_{ijk} \qquad i \in N_0 \tag{2.6h}$$

$$\gamma_k \leq \sum_{(i,j,k) \in Z} z_{ijk} \qquad k \in N_0' \tag{2.6i}$$

$$a_i + t_{ij}^T + \sigma_i^T \beta_i + \sigma_j^T (\alpha_j + y_j^T) \leq a_j + M(1 - x_{ij}^T) \qquad (i,j) \in A \tag{2.6j}$$

$$a_i + \sum_{(i,j,k) \in Z} (t_{ijk}^D + M) z_{ijk} \leq a_k + M \qquad i \in N_0, k \in N_0' \tag{2.6k}$$

$$a_i + \sum_{(i,j,k) \in Z} (T_{ijk}^D - M) z_{ijk} \geq a_k - M \qquad i \in N_0, k \in N_0' \tag{2.6l}$$

$$\sum_{(i,j,k) \in Z} z_{ijk} = y_j^D \qquad j \in N \tag{2.6m}$$

$$\sum_{(k,i,j) \in Z} z_{kij} + \sum_{(i,j,k) \in Z} z_{ijk} \leq x_{ij}^D \qquad (i,j) \in A \tag{2.6n}$$

$$M(1 - x_{ij}^D) + a_j \geq a_i + y_j^D \left(\frac{d_{ij}^D}{v_{max}^D} \right) \qquad (i,j) \in A \tag{2.6o}$$

$$x_{ij}^T \geq x_{ij}^D + y_i^C + y_j^C - 2 \qquad (i,j) \in A \tag{2.6p}$$

$$x_{ij}^T, x_{ij}^D \in \{0,1\} \qquad (i,j) \in A \tag{2.6q}$$

$$y_i^T, y_i^D, y_i^C \in \{0,1\} \qquad i \in N \tag{2.6r}$$

$$z_{ijk} \in \{0,1\} \qquad (i,j,k) \in Z \tag{2.6s}$$

$$\alpha_i, \beta_i, \gamma_i \in \{0,1\} \qquad i \in V \tag{2.6t}$$

$$a_i \in \mathbb{R}_+ \qquad i \in V \tag{2.6u}$$

The objective function (2.6a) aims at minimizing the total tour duration to serve all customers and return to the depot. Precisely, the objective function is equal to the variable $a_{0'}$, plus the time required to launch and retrieve the drone $s_L + s_R$ multiplied by the number of drone legs in solution $\sum_{i \in N} y_i^D$, plus the time required to serve the customers

in between successive operations $\sigma_i^T \sum_{i \in N} \gamma_i$, as described by case 2.1(iii)). Constraints (2.6b) are flow conservation constraints for the truck and link the x_{ij}^T variables with the variables y_i^T and y_i^C. Constraints (2.6c) ensure that the truck leaves and returns to the depot exactly once. Constraints (2.6d)–(2.6e) are equivalent to constraints (2.6b)–(2.6c) but apply to the drone. Constraints (2.6f) ensure that each customer is a drone, truck, or combined customer and is visited exactly once. Constraints (2.6g) define how each combined customer is served. Constraints (2.6h) and (2.6i) ensure the compatibility between service types and operations. Constraints (2.6j) act as subtour elimination constraints for the truck and set the truck arrival time at each node. Constraints (2.6k) and (2.6l) set the minimum and maximum drone travel times for each drone leg $(i, j, k) \in Z$. Constraints (2.6m) link the variables z_{ijk} with the variables y_j^D. Constraints (2.6n) impose that a drone leg can contain arc $(i, j) \in A$ only if arc (i, j) is traversed by the drone. Constraints (2.6o) set a lower bound on the arrival time for the customers served by the drone. Constraints (2.6p) ensure that an arc traversed only by the drone must be part of a drone leg. Constraints (2.6q)–(2.6u) model the range of the decision variables.

The following inequality can significantly strengthen the continuous relaxation of formulation (2.6):

$$a_{0'} \geq \sigma_0^T + \sum_{(i,j) \in A} (t_{ij}^T + \sigma_j^T) x_{ij}^T \tag{2.7}$$

Constraint (2.7) specifies that $a_{0'}$ cannot be lower than the sum of the truck travel times of the arcs traversed by the truck plus the truck delivery times of all customers visited by the truck.

2.6 Genetic Algorithm

The genetic algorithm we propose for the FSTSP-VDS is inspired by the Hybrid Genetic Search with Adaptive Diversity Control (HGSADC) metaheuristic of Vidal et al. (2012), one of the most effective heuristics for the Capacitated Vehicle Routing Problem. Moreover, genetic algorithms have demonstrated good performance on problems similar to the FSTSP-VDS, see for example Ha et al. (2020) and Chen et al. (2025). The high-level scheme of our genetic algorithm is shown in Algorithm 1.

The method initially defines a random population of A^η individuals, each of which encodes one or more FSTSP-VDS solutions. At each iteration, A^γ new individuals (offspring) are created by combining two parent individuals together (crossover), selected according to their *biased fitness* (individual's cost adjusted by its diversity contribution, see Section 2.6.3). Then, a local search phase is applied (education). At the end of each iteration, the fitness of all individuals is updated, and A^γ individuals that belong to the

Algorithm 1: Genetic Algorithm(A^{η}, A^{γ}, A^{epochs}).

```
Data:
    A^η: population size
    A^γ: number of offspring at each iteration
    A^epochs: max number of iterations
Generate a random population of A^η individuals;
Evaluate the biased fitness of all individuals in the population;
while iterations < A^epochs do
    for i = 1 to A^γ do
        Select two parent individuals P1 and P2 according to their biased fitness;
        Generate offspring C from P1 and P2 (crossover);
        Apply local search procedures to C (education);
    end
    Evaluate the biased fitness of all individuals in the population;
    Replace A^γ individuals in the population with the new offspring;
end
return the best individual;
```

current population are replaced with the new individuals. Finally, after A^{epochs} iterations, the procedure returns the best individual found.

Section 2.6.1 describes how individuals encode solutions. Section 2.6.2 describes how the best solution associated with an individual is retrieved. Section 2.6.3 defines how individuals are evaluated with respect to the entire population. Section 2.6.4 describes how parents are selected and the crossover procedure to generate a new individual. Section 2.6.5 describes the local search procedures applied during the education phase. Finally, Section 2.6.6 describes how the population is managed during the evolution process.

2.6.1 Individual Encoding

Each FSTSP-VDS solution can be characterized (possibly, not uniquely) by a permutation of the customers set N representing the precedence relationship among customers. Similarly to what is done by Ha et al. (2018) and Agatz et al. (2018), we define the following encoding rule. Let p be a permutation of N. A solution is encoded by p if, for each $i, j \in N$ such that i is visited before j with the same vehicle, i occurs before j in p. From now on, we refer to a chromosome sequence as a sequence with prefix 0 followed by a permutation p and $0'$. An example of this encoding is illustrated in Figure 2.6, which shows the five solutions encoded by the chromosome sequence $(0, 1, 2, 0')$.

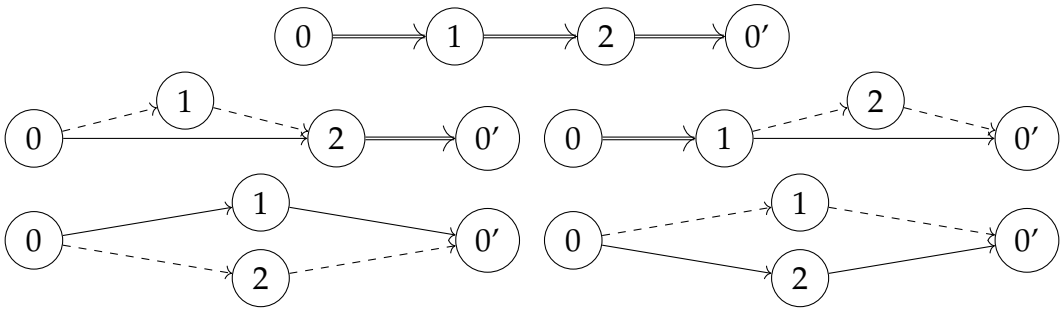


Figure 2.6: The five solutions encoded by the chromosome sequence $(0, 1, 2, 0')$.

Note that, since the precedence relationship among customers visited by the same vehicle is determined by the permutation sequence, there is a bijection between the set of operations and the set of drone legs, that is, an operation is uniquely determined by a drone leg.

The advantage of such encoding relies on the fact that, as we will show in Section 2.6.2, we can retrieve the best solution among all the solutions encoded by a permutation sequence in time $\Theta(|N|^3)$. Furthermore, we also propose an effective linear approximated method that runs in time $\Theta((A^b)^2|N|)$, where A^b is a parameter that sets the maximum truck leg length allowed, to limit the computational time of this procedure when dealing with large instances.

2.6.2 Chromosome Evaluation

In this section, we describe a dynamic programming algorithm, named *split*, that given a chromosome sequence, computes a minimum makespan solution among all the solutions encoded by the chromosome.

Let ρ be a chromosome such that $\rho = (\rho_0 = 0, \rho_1, ..., \rho_n, \rho_{n+1} = 0')$. Let $f_k(\rho)$ and $g_k(\rho)$ denote respectively the minimum time to reach ρ_k in ρ without serving it and the minimum time to reach and serve ρ_k, under the precedence relationship defined by the chromosome sequence ρ, when ρ_k is visited by both the truck and drone.

We denote with $\pi_\rho^{u,u}(i,k)$ the minimum time to go from ρ_i not yet served, to ρ_k without serving it, either with a combined arc if $k = i + 1$ or with an operation that starts from ρ_i and ends in ρ_k if $k > i + 1$. The definitions of $\pi_\rho^{u,s}(i,k)$, $\pi_\rho^{s,u}(i,k)$ and $\pi_\rho^{s,s}(i,k)$ follow by analogy, where the first superscript is equal to s if the starting node has been already served, u otherwise, and the second superscript is equal to s if the ending node must be served, u otherwise. Table 2.2 summarizes the notation for the above definitions.

It is easy to see that $f_k(\rho)$ and $g_k(\rho)$ satisfy the Bellman equations (2.8a) and (2.8b). The minimum time to reach ρ_k without serving it (2.8a), can be computed as the minimum

Notation	Start node ρ_i	End node ρ_k	Description
$\pi_\rho^{u,u}(i,k)$	Unserved	Unserved	Minimum time to go from ρ_i (not yet served) to ρ_k without serving ρ_k
$\pi_\rho^{u,s}(i,k)$	Unserved	Served	Minimum time to go from ρ_i (not yet served) to ρ_k and serve ρ_k
$\pi_\rho^{s,u}(i,k)$	Served	Unserved	Minimum time to go from ρ_i (already served) to ρ_k without serving ρ_k
$\pi_\rho^{s,s}(i,k)$	Served	Served	Minimum time to go from ρ_i (already served) to ρ_k and serve ρ_k

Table 2.2: Notation summary for π_ρ

time to go from ρ_i plus the minimum time to go from ρ_i to ρ_k, for each $i < k$, both when ρ_i has been already served ($g_i(\rho) + \pi_\rho^{s,u}(i,k)$) and when ρ_i has not yet been served ($f_i(\rho) + \pi_\rho^{u,u}(i,k)$). The same reasoning applies to compute the minimum time to reach and serve ρ_k (2.8b), whereas, instead of $\pi_\rho^{u,u}(i,k)$ and $\pi_\rho^{s,u}(i,k)$, we have respectively $\pi_\rho^{u,s}(i,k)$ and $\pi_\rho^{s,s}(i,k)$.

$$f_k(\rho) = \begin{cases} 0 & \text{if } k = 0 \\ \min\limits_{0 \le i < k} \left\{ \min\left\{ f_i(\rho) + \pi_\rho^{u,u}(i,k);\ g_i(\rho) + \pi_\rho^{s,u}(i,k) \right\} \right\} & \text{if } k \ge 1 \end{cases} \tag{2.8a}$$

$$g_k(\rho) = \begin{cases} \sigma_{i_0}^T & \text{if } k = 0 \\ \min\limits_{0 \le i < k} \left\{ \min\left\{ f_i(\rho) + \pi_\rho^{u,s}(i,k),\ g_i(\rho) + \pi_\rho^{s,s}(i,k) \right\} \right\} & \text{if } k \ge 1 \end{cases} \tag{2.8b}$$

Algorithm 2 shows the pseudocode of the dynamic programming algorithm that can be derived from the recurrence equations (2.8a) and (2.8b) to compute a minimum makespan solution among all the solutions encoded by a given chromosome sequence. The algorithm begins by initializing $f_i(\rho)$ and $g_i(\rho), 0 \le i < |\rho| - 1$, as the truck arrival times associated with the truck-only solution. Then, starting from $i = 0$ to $i = |\rho| - 2$, all the minimum makespan operations starting from ρ_i and ending in $\rho_k, k > i$ are evaluated, and the quantities $f_k(\rho)$ and $g_k(\rho)$ are updated accordingly. The correctness of the algorithm relies on the fact that at the beginning of each iteration i, the sub-problems referenced by $f_i(\rho)$ and $g_i(\rho)$ have already been solved.

The initialization phase (Lines 1 to 6) requires $\Theta(|\rho|)$ operations. Line 9 requires $\Theta(|\rho|)$ operations, since there are exactly $k - i - 1$ drone legs that start from ρ_i, end in ρ_k, and that satisfy the precedence relationship defined by ρ. Lines 10 and 11 require $\Theta(1)$ operations. Hence, Algorithm 2 runs in time $\Theta(|N|^3)$, since $|\rho| = |N| + 2$.

If we fix the maximum truck leg length to A^b, the time complexity of Algorithm 2 is reduced to $\Theta((A^b)^2|N|)$ because now Line 9 requires $\Theta(A^b)$ operations, and Line 9 is now

Algorithm 2: $split(\rho)$

Data:
$\rho = (\rho_0, \rho_1, \dots, \rho_{|\rho|-1})$: chromosome sequence

`// Initialization (truck-only solution)`

$f_0(\rho) \leftarrow 0.0$
$g_0(\rho) \leftarrow \sigma^T_0$
for $i = 1$ **to** $|\rho| - 1$ **do**
  $f_i(\rho) \leftarrow g_{i-1}(\rho) + t^T_{i_{i-1}, i_i}$
  $g_i(\rho) \leftarrow f_i(\rho) + \sigma^T_{\rho_i}$
end

`// Recursive Phase`

for $i = 0$ **to** $|\rho| - 2$ **do**
  for $k = i + 1$ **to** $|\rho| - 1$ **do**
    compute $\pi^{u,u}_\rho(i,k), \pi^{u,s}_\rho(i,k), \pi^{s,u}_\rho(i,k), \pi^{s,s}_\rho(i,k)$
    $f_k(\rho) \leftarrow \min\{f_k(\rho), f_i(\rho) + \pi^{u,u}_\rho(i,k), g_i(\rho) + \pi^{s,u}_\rho(i,k)\}$
    $g_k(\rho) \leftarrow \min\{g_k(\rho), f_i(\rho) + \pi^{u,s}_\rho(i,k), g_i(\rho) + \pi^{s,s}_\rho(i,k)\}$
  end
end
$best\_sol \leftarrow$ backtrack optimal solution
return $g_{|\rho|-1}(\rho), best\_sol$

executed $\Theta(A^b|N|)$ times rather than $\Theta(|N|^2)$ times. The idea of limiting the truck leg length stems from observing that solutions with low parallelism between the truck and the drone due to long truck legs are typically associated with a high makespan since they tend to be structurally more similar to TSP solutions.

2.6.3 Biased Fitness

A typical problem of genetic algorithms is premature convergence, which happens when the best individual in the population is far from optimality and the algorithm does not improve anymore because all individuals in the population resemble each other in terms of genetic material. Therefore, a key element of genetic algorithms is maintaining a diversified population in the evolution process to avoid premature convergence (see Fogel (1994)).

In population-based metaheuristics, a value representing the "quality" of each individual with respect to the entire population is defined, usually expressed as a function of the individual's cost, also called *fitness*, and one or more penalty terms. This value is therefore called *biased fitness* and is used to guide the "natural selection" during the

evolution process. Inspired by the work of Vidal et al. (2012), we define the *diversity contribution* $\Delta(P)$ for an individual P, as the average distance to its A^{close} neighbors in the population, represented by the set $N_{close}(P)$.

Let $\Gamma^T(P)$ and $\Gamma^D(P)$ be the arrays of successors of, respectively, the truck and the drone routes of the best solution among all the ones encoded by the individual P, such that, $\Gamma^T_i(P)$ is the next node visited after i in the truck route, and $\Gamma^D_i(P)$ is the next node visited after i in the drone route. Then, the distance $\delta(P_1, P_2)$ between two individuals P_1 and P_2 is defined as the Hamming distance between the array of successors $\Gamma^T(P_1)$ and $\Gamma^T(P_2)$, plus the Hamming distance between the array of successors $\Gamma^D(P_1)$ and $\Gamma^D(P_2)$. We assume that if node i is not visited by the truck or the drone, $\Gamma^T_i(P) = -1$ and $\Gamma^D_i(P) = -1$ respectively. Therefore, the diversity contribution $\Delta(P)$ of an individual P is computed according to Equation (2.9) where $\mathbf{1}(x)$ is the indicator function equal to 1 if x is true, 0 otherwise:

$$\begin{aligned}\Delta(P) &= \frac{1}{A^{close}} \sum_{P_n \in N_{close}(P)} \delta(P, P_n) \\ &= \frac{1}{A^{close}} \sum_{P_n \in N_{close}(P)} \sum_{i=0}^{|N|} \left(\mathbf{1}(\Gamma^T_i(P) \neq \Gamma^T_i(P_n)) + \mathbf{1}(\Gamma^D_i(P) \neq \Gamma^D_i(P_n)) \right)\end{aligned} \tag{2.9}$$

Let $fit(P)$ and $dc(P)$ be the positions of the individual P in the fitness ranking and the diversity contribution ranking in ascending and descending order respectively, with respect to the entire population. Then, we define the *biased fitness* $BF(P)$ of an individual P as

$$BF(P) = fit(P) + \lambda\, dc(P) \tag{2.10}$$

where $\lambda \in [0, 1]$ is a trade-off parameter used to control the relative importance of the diversity contribution over the fitness contribution of an individual. The *biased fitness* is thus a measure that balances the evolutionary pressure based on the fitness of the individuals (elitism) with the genetic diversity of the population, by penalizing individuals very "similar" to their neighbors and individuals with low fitness.

2.6.4 Parent Selection and Crossover

An important concept in genetics is genetic recombination, in which the genetic material of different organisms is combined to generate offspring with combinations of traits that differ from those found in both parents (see Fogel (1994)). This process, also known as *crossing over* (or crossover), works by breaking and rejoining chromosome segments. In the classic genetic algorithm paradigm, the biological crossover operation is simulated by combining the genetic information of two parent individuals. Various types of

crossover have been proposed in the literature. In this work, we focus on the single-point crossover. The idea of the single point crossover is to define a random crossover point c such that the resulting offspring is the concatenation of the first c elements of the first parent, with the last $n - c$ elements of the second parent, where n is the length of the two chromosomes.

However, when a single-point crossover is applied, duplicates can occur. Therefore, after copying the first c elements of the first parent, we copy the last $n - c$ elements of the second parent without inserting duplicates. The leftover elements are then shuffled and added to the offspring chromosome one by one in a position that minimizes the cost associated with the current partial chromosome. Let r be the left-over element we want to add. The minimum cost position can be found by calling the *split* procedure $O(|N|)$ times, every time on a different chromosome sequence that differs for the position of the element r. Since there can be at most $|N|/2$ left-over elements, and each element must be evaluated in at most $|N|$ different positions – each of which requires $O(|N|^3)$ operations – the time complexity of this crossover procedure is $O(|N|^5)$. If the approximated *split* procedure is chosen, the time complexity can be reduced to $O(b^2|N|^3)$.

For instances with hundreds of customers, the crossover procedure can become a bottleneck, so we propose a relaxed version with a reduced time complexity. Rather than evaluating the *split* procedure on the whole chromosome sequence, we only evaluate a partial chromosome sequence centered at r including at most A^τ elements, e.g., $A^\tau = 30$. In this way, the *split* procedure requires $O(b^2 A^\tau)$ operations, and the whole crossover procedure has a complexity of $O(b^2 A^\tau |N|^2)$.

With probability 0.5, instead of concatenating the last $n - c$ elements of the second parent after the first c elements of the first parent, we concatenate the first c elements of the second parent before the last $n - c$ elements of the second parent.

Finally, parents are selected using a binary tournament procedure, that randomly (with a uniform probability) selects two individuals from the population and returns the one with the best biased fitness.

2.6.5 Education

When an offspring is generated, it undergoes an *education* phase, which consists of different local search procedures applied to the best solution associated with the individual to improve the solution cost. The local search procedures devised for the education phase apply the following operations in this exact order:

- **Reverse operations**: for each operation, check if the solution cost decreases by reversing the operation, and if so, apply the inversion;

- **Swap customers**: for each operation, check if the solution cost can be improved by swapping the takeoff node or the rendezvous node with one of its neighbor nodes. If so, apply the most improving substitution;
- **Optimize truck and combined legs**: for each truck/combined leg with at most three truck customers, optimize the order of the truck customers. For truck/combined legs with more than three customers, apply the most improving two-opt move until there are no more improvements.

2.6.6 Population Management

In the last sections, we have presented a crossover operator that describes how two individuals combined generate an offspring, an education operator that describes the local search procedures an offspring undergoes when it is created, and how parents are selected. The population management operator complements these operators to propagate good and promising solutions while enhancing the diversity in the population. By following the work of Vidal et al. (2012), our population management mechanism consists of three components: *initialization*, *diversification*, and *survivor selection*.

In the *initialization* phase, A^{η} individuals are randomly generated by assigning to each of them a random chromosome sequence. The *diversification* phase is activated when the genetic algorithm cannot improve the best solution for A^{max} consecutive iterations. This stage aims to add new genetic material to the population to escape from the local minimum where the algorithm is currently stuck. In this phase, $A^{r} \cdot A^{\eta}$ individuals with the worst biased fitness are replaced with new random individuals.

As anticipated in Section 2.6.3, one of the main challenges of population-based metaheuristics is to avoid premature convergence. This problem is particularly exacerbated, when, as in our case, the repair phase in the crossover operator and the education operator are greedy procedures. The result is a general tendency to favor individuals with high fitness at the expense of the diversity of the population. For this purpose, the *biased fitness* measure has been defined to prevent premature convergence by discarding individuals with a low diversity contribution. Then, our *survivor selection* policy replaces after each iteration, the A^{γ} individuals with the worst biased fitness in the population with the offspring just generated.

Other than that, after every epoch we check if there are *clones* in the population, that is, if two individuals P_1, P_2 are such that $\delta(P_1, P_2) \leq 1 + A^{c} \cdot 2|V|$. If two clones are found, we replace the individual with the worst fitness with a new offspring.

2.7 Computational Results

In this section, we discuss the computational results achieved by the exact solution method (hereafter CF) described in Section 2.5 and by the metaheuristic solution method (hereafter GA) described in Section 2.6. This section is organized as follows. Section 2.7.1 describes the test instances used in the computational experiments. Section 2.7.2 presents the results of CF. Section 2.7.3 analyzes the drone speeds associated with an optimal FSTSP-VDS solution. Section 2.7.4 provides insights on fixed vs variable drone speeds. In Section 2.7.5, we compare the results achieved by GA with those achieved by Raj and Murray (2020) on single-drone FSTSP-VDS instances with up to 100 customers, available at https://github.com/optimatorlab/mFSTSP-VDS. In Section 2.7.6, we assess GA on two FSTSP variants with up to 99 customers in which the drone speed is assumed to be constant. Finally, in Section 2.7.7, we evaluate the impact of the main key components proposed in GA.

The computational tests of Raj and Murray (2020) have been performed on an Intel i7-6700 processor with 16 GB RAM running Ubuntu 14.04. Both CF and GA have been implemented in C and compiled with GCC 11.4 and with the optimization flag -O3. All experiments have been conducted on a 64-bit desktop computer with an AMD Ryzen 5 PRO 4650GE processor running at 3.3GHz and with 16 GB of RAM on a GNU/Linux Ubuntu 22.04.2 LTS operating system. The MILP formulation of CF was solved with CPLEX 22.1.1 with a time limit of one hour. We use the default parameters of CPLEX, except for CPX_PARAM_THREADS, set equal to 1 to use a single thread, CPX_PARAM_EPINT (set to 0 to decrease integrality tolerance), CPX_PARAM_EPRHS (set to 1e-6 to decrease feasibility tolerance) and CPX_PARAM_EPGAP (set to 1e-5 to decrease relative MIP gap tolerance).

The minimum and maximum travel times of all feasible drone legs are computed in preprocessing as described in Section 2.4 using the derivative-free optimization algorithm COBYLA of the open-source non-linear-optimization library NLopt 2.7.1 of Johnson (2007). We set a feasibility tolerance equal to 1e-6, a relative tolerance on the optimization variables equal to 1e-6, and a relative optimality tolerance of 1e-8. We impose a maximum number of NLopt iterations equal to 500 and 800 respectively for the minimum and maximum drone leg travel time optimization problems. If an optimization problem terminates before reaching the maximum number of iterations, we set the optimal (minimum or maximum) drone leg travel time equal to the returned optimal solution's cost. Otherwise, if we are solving the minimization (maximization) problem, we define an upper (lower) bound as the best feasible solution found during the optimization procedure and a lower (upper) bound as the travel time required to traverse the drone leg at the maximum (minimum) speed. We then apply the bisection

method until the difference between the lower and upper bound is not greater than 1e-6 by solving at each step the convex optimization problem (2.5) to determine the minimum energy required to traverse a drone leg in a target time. In this case, we solve (2.5) without imposing any limit on the number of iterations. To reduce the computational burden of this phase, we apply the bisection procedure just described only for the exact method CF or if NLopt returns an error, otherwise we define the minimum/maximum drone leg travel time as the best feasible solution's cost found during the optimization procedure.

All computing times reported in this section are in seconds. Due to the problem complexity, CF was tested only on FSTSP-VDS instances with 10 customers. For each instance tested, both CF and GA are executed 10 times with a different random seed from 0 to 9. The parameters of GA used in the experiments have been defined after some preliminary testing on a subset of instances and can be found in Table 2.3. The source code and the solutions obtained in these experiments are available at http://github.com/federicomichelotto/FSTSP-VDS.

Parameter	Meaning	Value
A^{epochs}	Number of iterations	2000
A^{η}	Population size	400
A^{γ}	Number of offspring in a generation	20
A^{close}	Number of close individuals considered for distance evaluation	15
A^{λ}	Biased fitness trade-off parameter	0.92
A^{max}	Max iterations without improvement	400
A^{b}	Max truck leg length	12
A^{τ}	Max length of partial chromosomes to consider during crossover	30
A^{r}	Fraction of individuals to reset during diversification	0.9
A^{c}	Max percentage of chromosome sharing	0.03

Table 2.3: GA parameters.

2.7.1 Test Instances

The FSTSP-VDS instances used in these experiments are the ones introduced by Raj and Murray (2020), available at https://github.com/optimatorlab/mFSTSP-VDS. For this set of instances, the maximum payload capacity is 5 lbs, and parcel weights are randomly chosen to be integer between 1 and 5 lbs for 85% of the customers, and more

than 5 lbs for the remaining customers. Thus, the set of possible drone payloads is equal to $\mathcal{W} = \{0, 1, 2, 3, 4, 5\}$. For each $w \in \mathcal{W}$, the second order derivative of $E^{PM}(v, w)$ with respect to v, is a function whose numerator and denominator are strictly positive functions for $v > 0$, and consequently, $E^{PM}(v, w)$ is convex for $v > 0$ for each $w \in \mathcal{W}$. Therefore, for each drone leg, the set of feasible drone leg travel times is a convex set.

The test set consists of two sets with customer locations in the Seattle and Washington area. The first set consists of 10 problems for each of four levels of the number of customers (10, 25, 50, and 100) generated on a fixed service area of 918.0 km^2, for a total of 40 problem instances. The second set consists of 10 problems, each one with 50 customers, for each of four levels of service area (57.4, 229.5, 516.4, and 918.0 km^2), for a total of 40 problems. In the following, we will refer to these two datasets as dataset A and B. In all instances, customer locations are generated from a uniform distribution on the road network within the service area. For each level of customers and each level of service area, 5 instances featured a centrally-located depot and 5 instances contained a depot at the periphery. The drone battery capacity is equal to $B = 500$ kJ. The takeoff and landing drone speeds are constants and set respectively to 10 m/s and 5 m/s. The cruise altitude h is set to 50 meters. Truck and drone delivery times σ^T and σ^D are set to 30 and 60 seconds respectively. Drone launch and retrieval times, s_L and s_R, are set to 60 and 30 seconds respectively. The distance d between two locations is computed using the Haversine distance formula as follows

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{\theta_2 - \theta_1}{2}\right) + \cos(\theta_1)\cos(\theta_2)\sin^2\left(\frac{\varphi_2 - \varphi_1}{2}\right)}\right)$$

Where θ_1, φ_1 represent the latitude and longitude of the first location, θ_2, φ_2 the latitude and longitude of the second location, and $R = 6378100.0$ denotes the radius of the earth in meters. The Liu et al. (2017) model is used to characterize the drone power consumption with respect to speed and payload weight, whose parameters are defined in Section 2.3.1. We test all instances with different maximum drone speeds $v^D_{max} = 10, 20, 30, 40$ m/s, for a total of 320 instances. The minimum drone speed v^D_{min} is set equal to 0.1 m/s.

The FSTSP instances used in these experiments are the ones used by Blufstein et al. (2024) (available at `https://github.com/fsoulignac/tspd-release`), an adaptation of the instances originally introduced by Poikonen et al. (2019) for the TSP-D. In particular, the test set consists of 250 instances with $n = 9, 19, \ldots, 99$ customers, 25 for each problem size. The instances were created by randomly locating the depot and the customers on a 50-by-50 grid using a uniform distribution on both axes. Given two locations i and j and the corresponding x, y coordinates (x_i, y_i) and (x_j, y_j), truck travel times are computed according to the Manhattan (taxicab) metric, i.e., $t^T_{ij} = \lfloor |x_i - x_j| + |y_i - y_j| \rfloor$, drone travel

times are instead computed according to the Euclidean distance metric at a speed that is μ times faster than the truck, i.e., $t_{ij}^D = \lfloor \frac{1}{\mu}\sqrt{(x_i - x_j)^2 + (y_i - j_j)^2} \rfloor$. Each instance is solved with $\mu = 1, 2, 3$, for a total of 750 instances.

2.7.2 Compact Formulation

In this section, we evaluate the proposed exact method CF on the ten instances with 10 customers of dataset A. Table 2.4 evaluates CF for different values of v_{max}^D: 10, 20, 30 and 40 m/s. Column instance reports the name of the instance evaluated. All instances begin with the prefix 20191230T1, which, for clarity, has been omitted from the table. Columns root and Δ report respectively the integrality gap, and the gap between the optimal solution obtained and the optimal solution for the FSTSP-VDS when $v_{max}^D = 40$ m/s. Specifically, the integrality gap is measured as $(z^* - LB0)/z^*$, where z^* is the optimal solution cost and $LB0$ is the linear relaxation cost. Column time reports the overall computing time, including preprocessing.

First of all, all instances tested have been solved to optimality by CF. The average computing time goes from 160.27 seconds when $v_{max}^D = 10$ m/s to 64.04 seconds when $v_{max}^D = 40$ m/s. The average integrality gap is about 50% independently of the value of v_{max}^D. Clearly, Δ is higher for lower values of v_{max}^D, and it is respectively 19.60%, 3.08% and 0.18% for v_{max}^D values equal to 10, 20 and 30 m/s. Furthermore, we evaluated the impact of serving the combined customers just after the truck arrival, i.e., $\alpha = 1$ in CF, rather than allowing the three possible ways depicted in Figure 2.1. In this case, the makespan increases by 0.38% on average, up to a maximum of 1.26%.

Instance	$v_{max}^D = 10$			$v_{max}^D = 20$			$v_{max}^D = 30$			$v_{max}^D = 40$		
	root	Δ	time	root	Δ	time	root	Δ	time	root	Δ	time
45624016194	53.56	28.57	302.82	45.87	7.98	34.16	41.55	0.00	14.50	41.55	0.00	12.68
45645377021	42.27	15.11	71.19	46.37	4.19	149.41	44.12	0.00	87.38	44.12	0.00	145.74
45706863992	41.70	22.12	80.02	56.45	1.94	69.14	55.61	0.00	14.31	55.61	0.00	11.07
45728368390	48.78	20.50	189.96	43.22	4.18	152.17	40.84	0.00	72.14	40.84	0.00	84.50
45749863540	40.36	11.75	159.23	36.71	0.95	78.91	36.11	0.00	30.80	36.11	0.00	31.34
45854314056	47.77	16.25	251.93	53.41	1.29	173.42	52.81	0.00	150.00	52.81	0.00	134.67
45916460302	53.90	30.71	299.20	56.61	7.13	170.68	54.33	1.79	71.18	53.51	0.00	85.13
45938067895	57.92	15.05	109.05	52.06	0.15	118.49	51.99	0.00	91.32	51.99	0.00	78.53
45959409904	47.06	28.85	42.61	48.22	2.43	67.30	46.97	0.02	29.07	46.96	0.00	30.20
50020711011	44.30	7.06	96.71	52.29	0.52	66.31	52.04	0.00	29.73	52.04	0.00	26.49
All	47.76	19.60	160.27	49.12	3.08	108.00	47.64	0.18	59.04	47.55	0.00	64.04

Table 2.4: Computational results of CF for different maximum drone speeds.

2.7.3 Optimal Drone Speeds

Observe that, in general, a drone leg $(i, j, k) \in Z$ can be completed in $t \in [t^D_{ijk}, T^D_{ijk}]$ seconds (considering the potential hovering above the rendezvous node) by more than one drone speed configuration (v^D_{ij}, v^D_{jk}), and that the set of all possible drone speed configurations is convex if the drone energy-per-meter function is convex (see Section 2.4 and in particular, formulation (2.5)). Therefore, reporting the drone speeds associated with a FSTSP-VDS solution is not always possible. This section addresses this point and aims to provide meaningful insights.

Figure 2.7 depicts the optimal solution found for the ten-customer instance 45624016194 of dataset A. Table 2.5 shows the main features of the operations in this solution. Columns i, j and k denote, respectively, the takeoff node, the drone customer, and the rendezvous node. Column $t^T_{i\to k}$ denotes the truck leg travel time to go from i to k, including the delivery times to serve all customers in between, and, for the sake of simplicity, also customer k since all rendezvous customers in this solution are served before the truck and drone synchronization ($\alpha_k = 1$). Columns t^D_{ijk} and T^D_{ijk} denote, respectively, the minimum and maximum drone leg travel times. Therefore, the optimal drone leg travel time for each operation must be equal to $\max\{t^D_{ijk}, t^T_{i\to k}\}$ seconds, and the maximum of these two values is highlighted in bold. Column w^T denotes the waiting time of the truck for the drone arrival at the rendezvous node k, computed as $\max\{t^T_{i\to k} - t^D_{ijk}, 0.0\}$. Columns $\bar{v}^D_{ij}$ and $\bar{v}^D_{jk}$ denote the feasible drone speed configurations that allows to complete the drone leg $(i, j, k) \in Z$ (considering the potential hovering above the rendezvous node) in exactly $\max\{t^D_{ijk}, t^T_{i\to k}\}$ seconds, in a way that the energy consumption is minimized. To compute these quantities, we solved the convex optimization problem (2.5). Finally, column V^D_{ij} (V^D_{jk}, resp.) denotes the convex set of drone speeds to go from i to j (from j to k, resp.) such that there exists at least a feasible drone speed to go from j to k (from i to j, resp.) that allows to complete the drone leg $(i, j, k) \in Z$ (considering the potential hovering above the rendezvous node) in exactly $\max\{t^D_{ijk}, t^T_{i\to k}\}$ seconds without exceeding the battery capacity. To compute these convex sets, we implemented a bisection procedure that recursively solves a sub-problem of 2.5 in which either t^D_{ij} or t^D_{jk} is fixed. Note that the cartesian product $V^D_{ij} \times V^D_{jk}$ is a super-set of the convex set of all feasible drone speed configurations. All times and speeds reported in Table 2.5 are expressed, respectively, in seconds and meters/second.

The solution's makespan is equal to the sum of all times reported in Figure 2.7 (truck travel and waiting times), plus the time required to launch the drone ($s_L = 60$ seconds) and retrieve it ($s_R = 30$ seconds) multiplied by the number of drone legs in the solution (5), yielding a total of 4754.38 seconds.

Notice that the duration of the first three operations is given by t^D_{ijk} since the truck must wait for the drone, and the possible speed ranges are quite small. On the other hand, in

the last two operations whose duration is equal to $t^T_{i\to k}$, since the drone must wait for the truck, and the possible drone speed ranges are much larger.

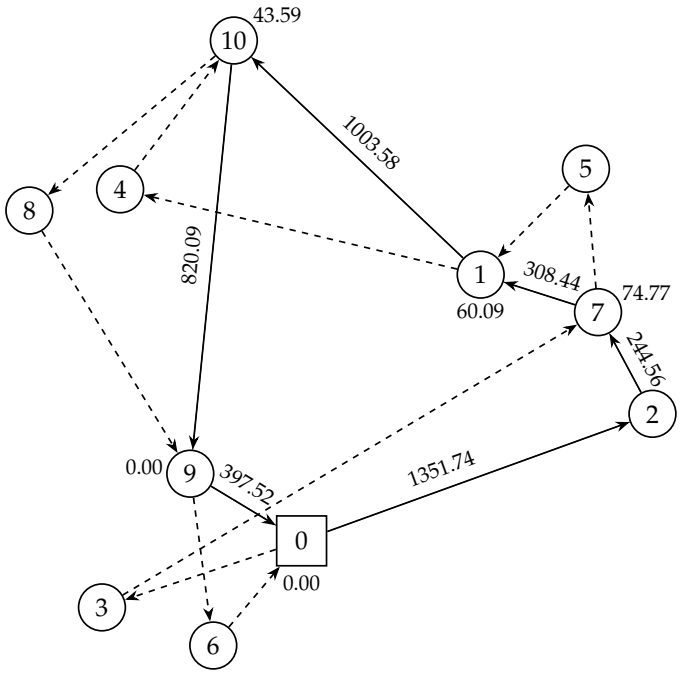


Figure 2.7: Optimal solution for the instance 45624016194 with a makespan of 4754.38 seconds. Values above the edges and close to the rendezvous nodes denote, respectively, the truck travel times and truck waiting times for the drone arrival, in seconds. Since all rendezvous customers are served before the truck and drone synchronization in this solution, truck delivery times have been included in the truck travel times. Times related to the drone launching and retrieval operations are not shown and are equal to $s_L + s_R = 90$ seconds for each drone leg.

i	j	k	$t^T_{i\to k}$	t^D_{ijk}	T^D_{ijk}	w^T	$\bar{v}^D_{ij}$	$\bar{v}^D_{jk}$	V^D_{ij}	V^D_{jk}
0	3	7	1596.30	**1671.07**	2162.50	74.77	20.58	16.14	[20.58, 20.58]	[16.14, 16.14]
7	5	1	308.44	**368.53**	2485.57	60.09	34.11	32.96	[34.10, 34.11]	[32.96, 32.96]
1	4	10	1003.58	**1047.17**	2473.26	43.59	22.60	21.60	[22.60, 22.60]	[21.60, 21.60]
10	8	9	**820.09**	779.51	2434.47	0.00	22.10	24.87	[20.64, 29.73]	[19.92, 26.71]
9	6	0'	**397.52**	389.72	2390.55	0.00	30.03	34.54	[29.25, 36.74]	[27.57, 35.87]

Table 2.5: Main features of the operations corresponding to the optimal solution depicted in Figure 2.7.

2.7.4 Fixed vs Variable Drone Speeds

Figure 2.8 illustrates the advantage of not fixing a priori the drone speed over the ten instances with ten customers solved with CF. The figure shows how the gap with respect to the optimal FSTSP-VDS solution with $v^D_{max} = 40$ m/s is affected by varying the value

of v^D_{max}, both when the drone speed is fixed and when it is not. All instances have been solved to optimality. The red line represents the average gap when the drone speed is not fixed, i.e., the Δ values of Table 2.4. The blue line denotes the average gap when the drone speed is fixed to the maximum allowed speed v^D_{max}. When v^D_{max} is 10 or 20 m/s, fixing or not the drone speed produces a gap difference of 0.0% and 1.99% respectively. Instead, when v^D_{max} is 30 or 40 m/s, fixing the drone speed to v^D_{max} results in a gap increase of 22.82% and 32.06%, respectively. The best results when the drone speed is fixed have been obtained with a speed of 20 m/s; however, even in this case, the gap with respect to the optimal FSTSP-VDS solution with $v^D_{max} = 40$ m/s is 5.07%. Finally, we also evaluated the case in which the drone speed for each drone flight is set equal to the speed that maximizes the drone range (it depends only on the payload weight) and obtained a gap of 7.79%, even larger than always flying at 20 m/s.

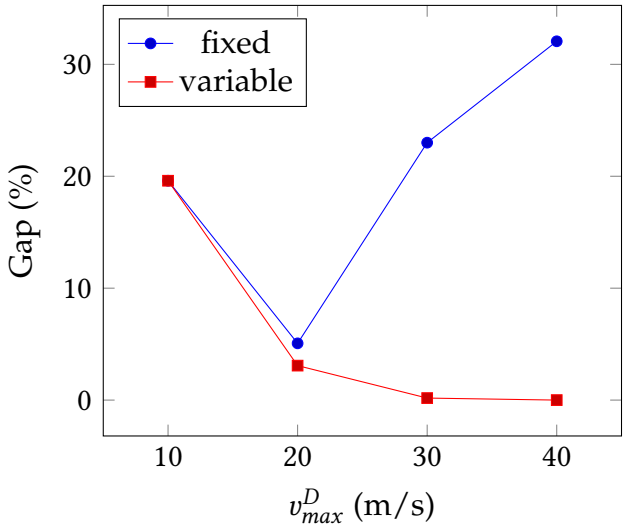


Figure 2.8: Average gap with respect to the optimal FSTSP-VDS solutions with $v^D_{max} = 40$ m/s, for different maximum drone speeds. In blue, the drone speed is fixed and equal to v^D_{max}. In red, the drone speed is variable and can vary between v^D_{min} and v^D_{max}.

2.7.5 Genetic Algorithm on FSTSP-VDS instances

In this section, we assess GA on FSTSP-VDS instances. To the best of our knowledge, the only known solution method for the FSTSP-VDS in literature is the heuristic method of Raj and Murray (2020), hereafter RM. Therefore, we refer to the best-known solution as the best solution between our method GA and RM for the instances with more than ten customers, and the optimal solutions obtained by CF for the instances with ten customers.

Tables 2.6 and 2.7 compare GA with RM respectively on the dataset A and B. Columns min and avg report, respectively, the aggregate average gap and the aggregate minimum

gap, in percentage, with respect to best-known solutions, columns preprocessing and time report the average preprocessing time required to compute the minimum and maximum drone leg time of all drone legs, and the average overall computing time, including preprocessing.

For all instances, the best-known solution was obtained by GA. For all instances with 10 customers, GA was able to find an optimal solution, for each seed tested. Across the 160 instances of dataset A, the aggregate average gap is 0.16% for the proposed method GA and 7.78% for RM. The computing time of GA ranges from a few seconds on the instances with ten customers up to about 10 minutes on the instances with 100 customers. The computing time of RM is significantly lower than GA up to 50 customers, but it reduces as the instance size increases, and on the largest instances with 100 customers, the two methods have comparable computing times, showing that GA scales better than RM. The preprocessing time grows exponentially and requires hundreds of seconds on the largest set of instances with 100 customers. No significant differences can be observed on dataset B.

		GA				RM	
$\|N\|$	v^D_{max}	min	avg	preprocessing	time	avg	time
10	10	0.00	0.00	0.29	6.94	4.55	0.32
	20	0.00	0.00	0.09	6.40	4.87	0.28
	30	0.00	0.00	0.11	6.41	6.73	0.20
	40	0.00	0.00	0.12	6.42	7.07	0.20
25	10	0.00	0.00	4.09	43.55	7.93	3.56
	20	0.00	0.01	1.30	27.30	7.15	3.24
	30	0.00	0.07	1.60	27.43	7.09	2.81
	40	0.00	0.00	1.86	28.24	7.34	2.83
50	10	0.00	0.08	35.78	157.14	9.21	28.92
	20	0.00	0.15	11.54	91.05	10.47	31.01
	30	0.00	0.22	14.20	90.56	7.82	28.43
	40	0.00	0.29	16.65	91.83	8.68	28.50
100	10	0.00	0.37	274.66	735.30	9.74	491.46
	20	0.00	0.45	92.97	411.99	8.42	580.25
	30	0.00	0.46	109.78	400.82	8.41	465.52
	40	0.00	0.48	128.89	412.08	9.06	454.70
All	10	0.00	0.11			7.86	
	20	0.00	0.15			7.72	
	30	0.00	0.19			7.51	
	40	0.00	0.19			8.04	
All		0.00	0.16			7.78	

Table 2.6: Comparison between GA and RM for different maximum drone speeds on dataset A.

		GA				RM	
$\|N\|$	v^D_{max}	min	avg	preprocessing	time	avg	time
50	10	0.00	0.17	84.74	191.21	7.73	36.03
	20	0.00	0.17	14.16	91.20	7.72	34.33
	30	0.00	0.19	24.06	96.81	5.92	28.96
	40	0.00	0.22	31.00	102.91	6.12	27.99
All		0.00	0.19			6.87	

Table 2.7: Comparison between GA and RM for different maximum drone speeds on dataset B.

Instance	z	time	gap_10	gap_20	gap_30	gap_40	gap_range
20191230T145624016194	4694.47	12.59	28.57	9.60	18.86	45.32	12.26
20191230T145645377021	4657.36	144.69	15.11	6.77	28.08	28.08	6.70
20191230T145706863992	4339.44	10.99	22.12	1.94	16.09	35.68	7.26
20191230T145728368390	5263.51	83.80	20.50	5.04	23.91	37.03	9.97
20191230T145749863540	5370.37	31.16	11.75	2.60	10.20	15.57	4.26
20191230T145854314056	5996.43	133.29	16.25	4.78	22.83	25.45	4.83
20191230T145916460302	3886.97	84.45	30.71	7.13	30.57	40.32	11.99
20191230T145938067895	6225.81	77.96	15.05	5.06	21.04	30.04	6.88
20191230T145959409904	4380.39	29.92	28.85	6.59	46.81	49.11	11.72
20191230T150020711011	4890.98	26.29	7.06	1.18	11.63	19.94	2.07
All		63.51	19.60	5.07	23.00	32.66	7.79

Table 2.8: Comparison between variable and constant drone speed on optimally solved FSTSP-VDS instances with 10 customers.

2.7.6 Genetic Algorithm on FSTSP instances

In this Section we evaluate the computational results of GA on FSTSP instances with respect to the solutions found by the exact solution method of Blufstein et al. (2024) (hereafter PFA). Table 2.9 reports the computational results of GA on two FSTSP variants evaluated by Blufstein et al. (2024), named, BASE and MHD. In the BASE variant, the drone endurance is infinite, a node must be visited exactly once, and the drone cannot be launched and retrieved in the same location. The MHD variant has the same constraints as the BASE variant, plus the following ones. The drone has a finite endurance equal to $e = 20$, it cannot land and wait at the rendezvous location, the time to launch and retrieve the drone requires a time equal to $s_L = s_R = 1$, and 20% of customers are drone-incompatible (in particular, a customer $j \in N$ is drone-incompatible if $j - 1$ is a multiple of 5). The launch time s_L of the operations that start from the depot is not taken into account. Following Blufstein et al. (2024), for instances with $n \geq 49$ customers, nodes'

coordinates, e, s_L and s_R are multiplied by 100. The results of PFA were obtained with a time limit of one hour, and to the best of our knowledge, all best-known solutions for this set of instances belong to PFA.

		BASE						MHD					
$\|N\|$	μ	Ψ	opt	min	avg	avg_opt	time	Ψ	opt	min	avg	avg_opt	time
	1	25	25	0.00	0.00	0.00	5.03	25	25	0.00	0.00	0.00	5.09
9	2	25	25	0.00	0.00	0.00	4.99	25	25	0.00	0.00	0.00	5.08
	3	25	25	0.00	0.00	0.00	4.99	25	25	0.00	0.00	0.00	5.02
	1	25	25	0.00	0.00	0.00	11.09	25	25	0.00	0.00	0.00	14.01
19	2	25	25	0.00	0.01	0.01	10.47	25	25	0.00	0.00	0.00	12.12
	3	25	25	0.00	0.03	0.03	10.38	25	25	0.00	0.01	0.01	10.59
	1	25	25	0.00	0.01	0.01	21.72	25	25	0.00	0.00	0.00	29.98
29	2	25	25	0.00	0.03	0.03	19.20	25	25	0.00	0.02	0.02	22.93
	3	25	23	0.11	0.37	0.37	18.74	25	25	0.00	0.08	0.08	19.17
	1	25	25	0.00	0.04	0.04	34.98	25	25	0.00	0.00	0.00	49.20
39	2	25	24	0.02	0.23	0.23	28.45	25	25	0.00	0.04	0.04	35.76
	3	25	20	0.18	0.52	0.52	26.79	25	23	0.07	0.24	0.24	29.30
	1	25	22	0.01	0.07	0.07	49.52	25	25	0.00	0.01	0.01	77.04
49	2	25	24	0.01	0.23	0.23	34.64	25	21	0.02	0.08	0.08	47.43
	3	25	15	0.09	0.55	0.55	33.17	25	17	0.10	0.45	0.45	37.71
	1	25	21	0.02	0.17	0.17	61.80	25	25	0.00	0.05	0.05	103.10
59	2	25	19	0.04	0.38	0.38	44.88	25	21	0.03	0.22	0.22	61.82
	3	25	15	0.10	0.59	0.59	42.91	22	11	0.15	0.38	0.39	50.45
	1	25	23	0.00	0.18	0.18	78.17	23	23	0.00	0.09	0.08	126.97
69	2	25	13	0.07	0.47	0.47	59.90	23	14	0.08	0.50	0.52	81.68
	3	23	10	0.13	0.72	0.72	57.64	16	4	0.09	0.41	0.49	67.70
	1	23	19	0.01	0.28	0.27	97.02	20	17	0.02	0.20	0.21	153.56
79	2	22	8	0.11	0.49	0.50	74.74	18	4	0.19	0.63	0.72	102.51
	3	23	2	0.34	1.02	0.98	71.46	17	0	0.23	0.72	0.72	84.68
	1	13	11	0.01	0.45	0.53	123.66	11	9	0.00	0.20	0.16	189.22
89	2	19	3	0.18	0.83	0.82	93.74	8	0	0.17	0.59	0.87	131.94
	3	17	1	0.56	1.30	1.32	92.82	6	0	0.17	0.64	1.00	107.59
	1	1	1	0.00	0.49	0.11	152.16	1	0	0.00	0.44	0.27	230.49
99	2	10	2	0.09	0.90	0.83	118.65	3	0	0.04	0.80	1.26	163.29
	3	8	0	0.16	1.30	1.52	114.10	1	0	0.10	0.68	0.72	132.40
	1	212	197	0.01	0.17	0.12		205	199	0.00	0.10	0.05	
All	2	226	168	0.05	0.36	0.30		202	160	0.05	0.29	0.22	
	3	221	136	0.17	0.64	0.57		187	130	0.09	0.36	0.29	
All		659	501	0.07	0.39	0.33		594	489	0.05	0.25	0.18	

Table 2.9: Computational Results of GA on FSTSP instances.

Columns Ψ and opt indicate respectively the number of instances solved to optimality by PFA, and how many of these instances are solved to optimality by GA, by at least one

run. Columns min and avg report respectively the aggregate minimum and average percentage gap of GA with respect to the best-known solutions. Column avg_opt reports the aggregate average percentage gap of GA with respect only to the optimal solutions found by PFA. Column time reports the average computing time of GA in seconds.

For the BASE configuration, GA obtains an aggregate minimum and average gap with respect to the best-known solutions respectively of 0.07% and 0.39%. GA obtains an aggregate average gap of 0.33% with respect to the 659 proven optimal solutions found by PFA, and for 501 of them, GA finds an optimal solution (0.76%). When the drone endurance is limited (MHD) GA obtains an aggregate minimum and average gap with respect to the best-known solutions respectively of 0.05% and 0.25%. For the MHD variant, GA obtains an aggregate average gap of 0.18% with respect to the 594 proven optimal solutions found by PFA, and for 489 of them, GA finds an optimal solution (0.82%). The average computing time on these instances ranges from a few seconds on the smallest instances with 9 customers up to about 4 minutes on the largest instances with 99 customers. If we analyze the gap with respect to the different drone flying speed levels $\mu = 1, 2, 3$, we observe an aggregate average gap respectively equal to 0.17%, 0.36%, and 0.64% for the BASE variant, and a gap equal to 0.10%, 0.29% and 0.36% for the MHD variant. Both for the BASE and the MHD variant, the increase of μ (i.e., the increase of the drone speed) is associated with an increase in the gap. It is worth noting that for $\mu = 1$, GA was able to find an optimal solution for all but 21 (6 if we consider only the MHD variant) of the 417 instances tested. Moreover, GA was also able to find optimal solutions for some of the largest instances with 99 customers.

2.7.7 Impact of Key Components of GA

In this section, we evaluate the impact of two key components of the proposed heuristic method GA: the repair phase and the approximated split procedure. To evaluate the effectiveness of these components – described in Section 2.6 – we compare the results obtained by GA on Dataset A, against two variants, namely, $\text{GA}_{\text{repair}}$ and GA_{split}.

In the variant $\text{GA}_{\text{repair}}$, instead of using the *split* procedure to reinsert the missing customers into the offspring chromosome, a more classical and simpler distance-based reinsertion is adopted. In particular, the missing customers are reinserted one by one, in random order, in one of the empty positions that minimizes the sum of distances to the predecessor and successor nodes. Conversely, the variant GA_{split} removes the limit on the truck leg length within the split procedure, equal to $A^b = 12$.

Columns avg and time report the average gap and the computing time for each method, respectively. To ensure a fair comparison, $\text{GA}_{\text{repair}}$ has not been evaluated with an iteration-limit but, rather, with a time-limit equal to the average computing time ob-

tained by GA to execute $A^{epochs} = 2000$ iterations, depending on the instance size $|N|$ and the maximum drone speed v^D_{max}.

For $\text{GA}_{\text{repair}}$, the column time is replaced by the column iterations, which reports the number of iterations executed by $\text{GA}_{\text{repair}}$. The variant $\text{GA}_{\text{repair}}$ executes 10 to 100 times more iterations than GA. However, it achieves gaps similar to those of GA only on the ten-customer instances; thereafter, the gap grows substantially, reaching a gap of 17.10% on the 100-customer instances with $v^D_{max} = 10$ m/s. Overall, the variant GA_{split} achieves results comparable to those of GA, but at the cost of higher computing times. Some small gap improvements can be observed on the 100-customer instances with $v^D_{max} = \{10, 40\}$ m/s; however, the computing time on these instances is two to three times higher than that of GA.

		GA		$\text{GA}_{\text{repair}}$		GA_{split}	
$\|N\|$	v^D_{max}	avg	time	avg	iterations	avg	time
10	10	0.00	6.94	0.00	21782.10	0.00	6.96
	20	0.00	6.40	0.02	72918.90	0.00	6.41
	30	0.00	6.41	0.02	127013.10	0.00	6.42
	40	0.00	6.42	0.06	204590.00	0.00	6.44
25	10	0.00	43.55	1.87	22784.90	0.00	65.99
	20	0.01	27.30	1.78	49843.80	0.01	37.11
	30	0.07	27.43	2.27	88764.90	0.07	37.21
	40	0.00	28.24	2.21	154214.20	0.00	38.42
50	10	0.08	157.14	6.66	22657.10	0.05	294.49
	20	0.15	91.05	5.94	49304.90	0.14	145.86
	30	0.22	90.56	6.77	85126.00	0.22	144.62
	40	0.29	91.83	5.98	142370.20	0.29	145.03
100	10	0.37	735.30	17.10	22584.90	0.30	2005.46
	20	0.45	411.99	15.42	50736.70	0.45	934.84
	30	0.46	400.82	15.52	83285.70	0.46	903.85
	40	0.48	412.08	16.40	138589.10	0.40	906.59
All	10	0.11		6.41		0.09	
	20	0.15		5.79		0.15	
	30	0.19		6.14		0.19	
	40	0.19		6.16		0.17	
All		0.16		6.13		0.15	

Table 2.10: Comparison between GA and two variants for different maximum drone speeds on dataset A.

2.8 Conclusions

Most drone-aided routing problems assume a constant drone energy consumption model, that is, however, an assumption that does not hold in practice. The goal of this work was to study and propose effective solution methods for a more realistic variant of the Flying Sidekick Traveling Salesman Problem in which the drone power consumption is modeled as a more realistic non-linear function with respect to the speed and parcel weight.

We proposed an exact approach based on a compact MILP formulation and a genetic algorithm. Both solutions are based on the idea of computing in advance the minimum and maximum feasible drone leg travel times to avoid handling drone speeds as decision variables. To do so, we showed that the set of feasible drone leg travel times of a generic drone leg is convex if the energy-per-meter function is convex with respect to the speed, and devised a procedure to compute the minimum and the maximum of this set.

We point out that the underlying assumption of our proposed methods, i.e., that the energy-per-meter function that models the drone power consumption is convex with respect to the speed, is not an isolated case, but instead, a common characteristic of other popular drone energy models reported in literature, see for example Stolaroff et al. (2018) and Zhang et al. (2021).

The resulting compact MILP formulation allows to easily solve to optimality FSTSP-VDS instances with ten customers, but struggles with larger instances. For this purpose, we devised a genetic algorithm and evaluated it both on FSTSP-VDS and FSTSP (in which the drone speed is constant) instances. For the FSTSP-VDS, the proposed heuristic obtains an average makespan improvement of about 7% with respect to the state-of-the-art. Our heuristic also finds the optimal solutions of all the instances with ten customers solved to optimality by the MILP formulation. Regarding the FSTSP, the proposed heuristic obtains an average gap of 0.26% among the 1253 optimally solved instances ranging from 9 to 99 customers, and finds an optimal solution for 990 of them, including some of the largest instances with 99 customers.

From a managerial perspective, we investigated the impact of fixing the drone speed on the overall makespan. To get reliable insights, we only solved the ten instances with 10 customers solvable by the MILP formulation. The results show that even a reasonable drone speed like 10 m/s produces a massive increase of 19.60% in the makespan. At 20 m/s, the gap decreases to 5.07%, at 30 m/s and 40 m/s becomes respectively 23.00% and 32.66%. Choosing the maximum range speed for each drone flight resulted in a gap of 7.79%, greater than always flying at 20 m/s. These results suggest that choosing over-conservative speeds or high speeds but too "expensive" or too stringent selection policies, easily leads to a large increase in the makespan, and highlights the importance

of the drone speeds selection. This becomes even more prominent if we observe that these experiments have been performed on small instances with ten customers, thus, we can expect even larger gaps on larger real-world instances.

Computing the minimum and maximum drone leg times for all drone legs can become an issue when dealing with instances with more than 100 customers. For these instances, however, only heuristic approaches seem practicable, and therefore one could think of heuristic approaches also for computing these quantities, and of re-optimization procedures to call at runtime for promising drone legs only.

Chapter 3

Capacitated Vehicle Routing Problem

Is it possible to concurrently optimize different parts of a solution of a Capacitated Vehicle Routing Problem such that the recombination of the optimized partial solutions is still a feasible solution? The answer is clearly affirmative if the partial solutions to be optimized consist of a disjoint subset of routes, as it happens in the route-based decomposition scheme. In the other cases, the answer is unknown. This work addresses this question and unveils a counterintuitive property of the Capacitated Vehicle Routing Problem. A new decomposition scheme based on this property is proposed, named sequence-based decomposition.

3.1 Introduction

Historically, Vehicle Routing Problems (VRPs) were approached with exact methods, such as branch-and-bound, branch-and-cut and branch-and-price techniques. While these methods guarantee optimal solutions, their applicability is often limited to small or medium-sized instances due to their computational complexity. The advent of meta-heuristics, including genetic algorithms, simulated annealing, and tabu search, marked a significant shift by offering approximate solutions for larger problems, balancing solution quality with computational efficiency. However, algorithms and methods that work well for small to medium-sized instances may struggle or fail to scale effectively to larger problems. Large-scale VRP instances present unique challenges and opportunities in the field of combinatorial optimization. As the scale of these instances increases, the complexity of solving them grows exponentially due to the larger number of customers, vehicles, and potential routes.

Decomposition methods are crucial for addressing large-scale VRP instances. Rooted in the *divide-and-conquer* principle, decomposition methods consist of splitting the original (large-scale) problem into several smaller sub-problems, which can be solved far more

efficiently than the original one. After obtaining the solutions to these sub-problems, they are combined to form a solution for the original problem. We direct the reader to Santini et al. (2023) for an exhaustive analysis of decomposition strategies for vehicle routing problems.

To the best of our knowledge, the only known decomposition scheme that can be applied to a solution of a Capacitated Vehicle Routing Problem (CVRP) is the so-called *route-based decomposition* scheme[1]. The route-based decomposition scheme consists of partitioning the set of routes in the solution to generate independent sub-problems, solving the sub-problems, and then merging the corresponding sub-solutions. Since the resulting sub-solutions are independent of each other by definition, the merging operation is simply the union of the sub-solutions obtained by solving each sub-problem.

The main limitation of the route-based decomposition is clearly that the number of sub-problems that can be generated is limited by the number of routes in solution, and this number can be small even on large-scale instances if the vehicle capacity is large (see, for example, some large-scale instances proposed by Arnold et al. (2019)). Furthermore, when the number of routes in the solution is relatively small with respect to the number of sub-problems to be generated, it may happen that just few, if not just one, routes are assigned to a sub-problem, thus strongly limiting the possibility to obtain better solutions. In fact, in order to achieve an effective optimization, sub-problems should be able to work on a significant number of routes such that the inter-route optimization is not excessively restricted.

In this work, we present a novel decomposition scheme for the CVRP, named *sequence-based decomposition*, that generalizes the route-based decomposition by allowing different sub-problems to concurrently optimize different parts of the same route. In this case, the potential number of sub-problems that can be generated is no longer limited by the number of existing routes in the solution.

3.2 Problem Definition

The CVRP can be formally described as follows. A complete undirected graph $\mathcal{G} = (V, E)$ is given. The node set V is defined as $V = \{0\} \cup N$, where 0 represents the depot and $N = \{1, 2, \dots, n\}$ a set of $n \in \mathbb{N}^+$ customers to serve, each with a known positive demand $q_i, i \in [1, n]$. With each pair of nodes $(i, j) \in E$ is associated a non-negative travel cost $c_{ij} = c_{ji} \geq 0$. A fleet of identical vehicles of capacity Q located at the depot has to fulfill customer demands. A feasible vehicle route $R = (0, v_1, v_2, ..., v_r, 0)$ with $r \in \mathbb{N}^+$, is a simple

[1] A path-based decomposition scheme is mentioned in Santini et al. (2023). However, such a technique is not a decomposition scheme, but rather, a fixing procedure, since only one sub-problem is generated.

tour in $\mathcal{G}$ starting from the depot, visiting nodes $v_1, v_2, ..., v_r$ and returning back to the depot such that the total demand of the visited customers does not exceed the vehicle capacity Q. A feasible CVRP solution $\pi = \{R_1, R_2, ..., R_{|\pi|}\}$, is a set of feasible routes such that each customer is served exactly once. The CVRP consists of finding a feasible solution with minimum cost.

Observe that the edges selected by a CVRP solution form a multiset (i.e., a set with multiplicity), since an edge can be selected twice in case of single-customer routes. However, for the sake of conciseness, we could omit the prefix "multi" in the following.

3.3 Sequence-based Decomposition

The proposed *sequence-based* decomposition scheme creates $M \in \mathbb{N}^+$ complementary sub-problems $P_i, i = 1, 2, ..., M$, from a given feasible CVRP solution π. Let E_π be the multiset of edges selected by solution π, and $\{E_1, E_2, ..., E_M\}$ a partition of E_π, then, each sub-problem P_i is defined by fixing the edges $E_\pi \setminus E_i$. Since the edges $E_i \subseteq E_\pi$ are not fixed only in sub-problem P_i, we say that the edges E_i are "assigned" to sub-problem P_i. The partition $\{E_1, E_2..., E_M\}$ must be defined such that, for each route, at most one contiguous sequence of edges selected by π is assigned to the same sub-problem. The solution resulting from the decomposition π^* is defined as the union of the free edges selected by the sub-problems' final solutions $\pi_i^*, i = 1, 2, ..., M$, where by free edge we mean an edge not fixed.

Figure 3.1 shows a CVRP solution π in which the solution's edges E_π are partitioned into three multisubsets E_1, E_2 and E_3. Observe that for each route $R \in \pi$, the *residual capacity*, defined as $Q - \sum_{v \in R} q_v$, is greater than 0. Figure 3.2 shows an example of an infeasible solution π^* obtained from decomposing the solution π into three sub-problems defined by E_1, E_2 and E_3. The solution π^* in Figure 3.2 is indeed infeasible because it contains a route exceeding the vehicle capacity and subtour.

However, we will show in the following sections that both types of infeasibilities come from the residual capacity in the routes of the initial solution π, and that, surprisingly, a feasible solution is always returned if the residual capacity is removed from the initial solution. Therefore, before generating the sub-problems, an augmentation phase is applied to remove the residual capacity possibly present in the initial solution's routes.

Section 3.3 describes the four phases of the proposed sequence-based decomposition scheme: (i) partitioning, (ii) augmentation, (iii) sub-problems generation, and (iv) merging.

Section 3.4 shows that the resulting solution of the proposed sequence-based decompo-

sition scheme is guaranteed to be feasible, i.e., (i) cannot have routes that exceed the vehicle capacity and (ii) cannot have subtours. In all the following figures, superscripts inside nodes denote the nodes' demands.

Note that the solution and the corresponding partitioning shown in Figure 3.1 will be used as reference throughout this work.

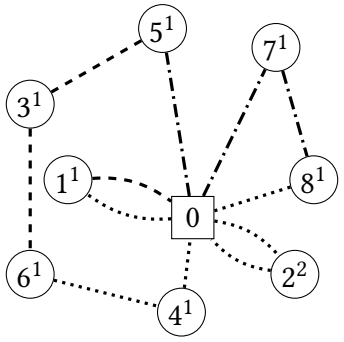


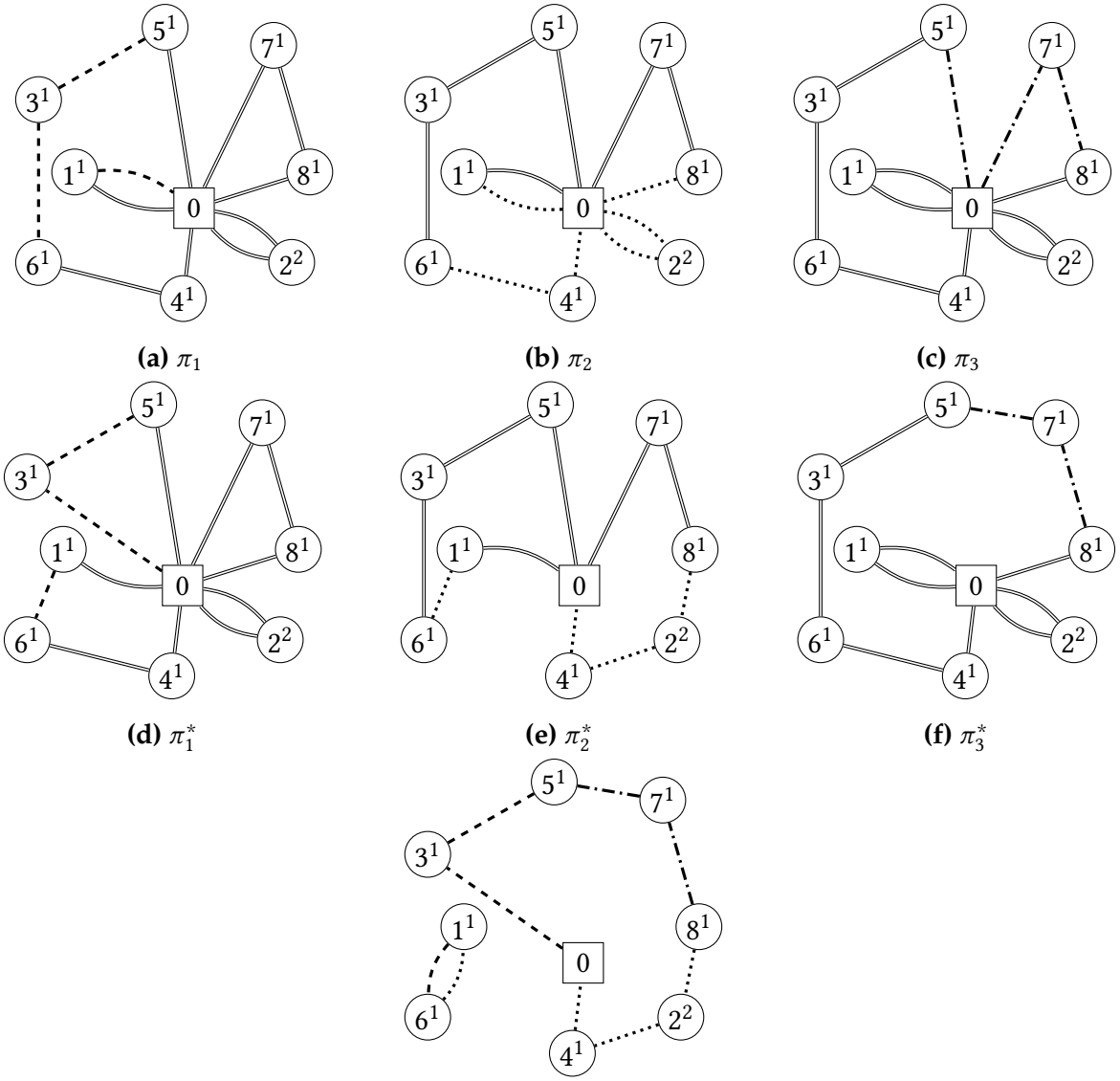
Figure 3.1: Example of a CVRP solution π for an instance with $n = 8$ customers and a vehicle capacity $Q = 6$. Superscripts denote customers' demands. The solution's edge multiset E_π is partitioned into three multisubsets $E_1 = \{(0,1),(3,5),(3,6)\}, E_2 = \{(0,1),(0,2),(0,2),(0,4),(0,8),(4,6)\}$ and $E_3 = \{(0,5),(0,7),(7,8)\}$. The partitioning is also depicted in figure by using different line styles, respectively, dashes, dots, and dashes alternated with dots. Observe that all routes have some residual capacity.

3.3.1 Partitioning Phase

Let π be a feasible solution and E_π be the set of edges selected by solution π. The partitioning phase defines a partition $\{E_1, E_2, ..., E_M\}$ of E_π such that, for each route $R \in \pi$, no more than one contiguous sequence of edges is assigned to $E_i, i \in [1, M]$. Sequences of only one edge are allowed. Figure 3.1 shows an example of a feasible CVRP solution whose edges are partitioned into three subsets. We say that $c \in N$ is a *conjunction customer* if its two incident edges belong to different partition sets, and we refer to $N' \subseteq N$ as the set of conjunction customers defined in the partitioning phase. For example, the set of conjunction customers defined by the partition in Figure 3.1 is $N' = \{1, 5, 6, 8\}$.

3.3.2 Augmentation Phase

Let λ_R be the set of contiguous sequences of edges of route $R \in \pi$ defined in the partitioning phase. Then, a new solution $\overline{\pi}$, hereafter called *augmented solution* of π, is defined such that each sequence of edges $S = ((v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)) \in \lambda_R, k \in \mathbb{N}^+, v_0, v_1, ..., v_k \in V$, is replaced by the edges $(v_0, v_S), (v_S, v_k)$, where v_S is a new node, hereafter called *augmented node*, which replaces $k - 1$ nodes $v_1, v_2, ..., v_{k-1}$. Observe that



(a) π_1 **(b)** π_2 **(c)** π_3

(d) π_1^* **(e)** π_2^* **(f)** π_3^*

(g) π^*

Figure 3.2: Example of infeasible decomposition. The solution in Figure 3.1 is decomposed into three sub-problems $P_i, i = 1, 2, 3$, each one defined by fixing the edges $E_\pi \setminus E_i$, where E_i denotes the edges assigned to sub-problem i as defined in Figure 3.1. Solutions π_i and π_i^* denote respectively the initial and final solutions for sub-problem P_i. The resulting solution π^* is obtained by merging the free edges of solutions π_1^*, π_2^* and π_3^*. Observe that π^* is infeasible since it contains a subtour and a route that exceeds the vehicle capacity $Q = 6$.

if $k = 1$, no node is replaced by v_S. The augmented node v_S is defined as follows. Its demand is non-negative and not smaller than the cumulative demand of the replaced customers, $q_{v_S} \geq \sum_{i=1}^{k-1} q_{v_i}$. The traveling cost to go from any node $i \in V$ to node v_S is $c_{iv_S} = c_{iv_1}$, and the traveling cost to go from node v_S to any node $i \in V$ is $c_{v_S i} = c_{v_{k-1} i}$.

Moreover, the augmented nodes' demands must be defined such that there is no residual capacity in the augmented solution's routes, i.e., $\sum_{v \in R} q_v = Q, \ \forall \ R \in \overline{\pi}$. We refer to Y as

the set of augmented nodes defined during the augmentation phase. Observe that the augmented solution $\overline{\pi}$ contains only augmented nodes and conjunction customers. An example of augmented solution is shown in Figure 3.3.

3.3.3 Sub-problems Generation Phase

Starting from the augmented solution $\overline{\pi}$, each sub-problem $P_i, i \in [1, M]$, is defined by a solution π_i in which each augmented node $v \in Y$ assigned to P_i is expanded into the original sequences of edges replaced by v during the augmentation phase. Furthermore, the edges incident to all remaining augmented nodes are fixed. Figure 3.4 shows an example of three sub-problems generated from the augmented solution of Figure 3.3. Note that the partial solutions of π_2^* and π_3^* shown in Figure 3.2 are not feasible for the augmented sub-problems in Figure 3.4 because they would violate the vehicle capacity in the corresponding sub-problems.

3.3.4 Merging Phase

Let π_i^* be the final solution obtained from the optimization of sub-problem $P_i, i \in [1, M]$. Let us denote the sequences of free edges selected by π_i^*, as the *partial solution* of π_i^*. Then, a new solution π^* is obtained by concatenating the partial solutions of all sub-problems $\pi_i^*, i \in [1, M]$. Since, by definition, each customer that is not a conjunction customer $v \in N \setminus N'$ appears exactly in one sub-problem, and each conjunction customer $c \in N'$ is a terminal node of exactly two partial solutions, we have the guarantee that each customer in solution π^* will have exactly two edges incident to it. Figure 3.6 shows the resulting solution obtained from merging the three solutions in Figure 3.5.

Here, we recap the notation that will be used in throughout this work. The current solution is denoted with π, and the augmented solution with $\overline{\pi}$. For each subproblem P_i, π_i represents its initial solution, while π_i^* denotes the final solution obtained after optimization. The merged solution resulting from combining all subproblem solutions is denoted by π^*.

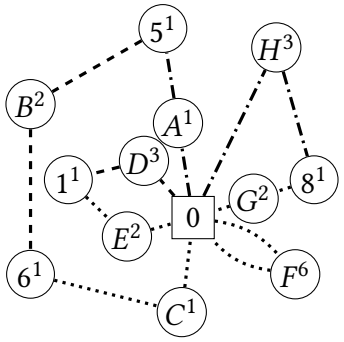
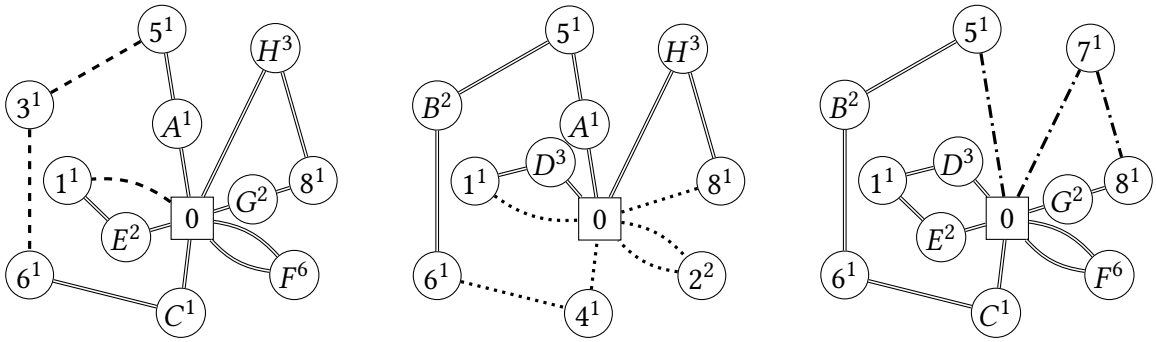
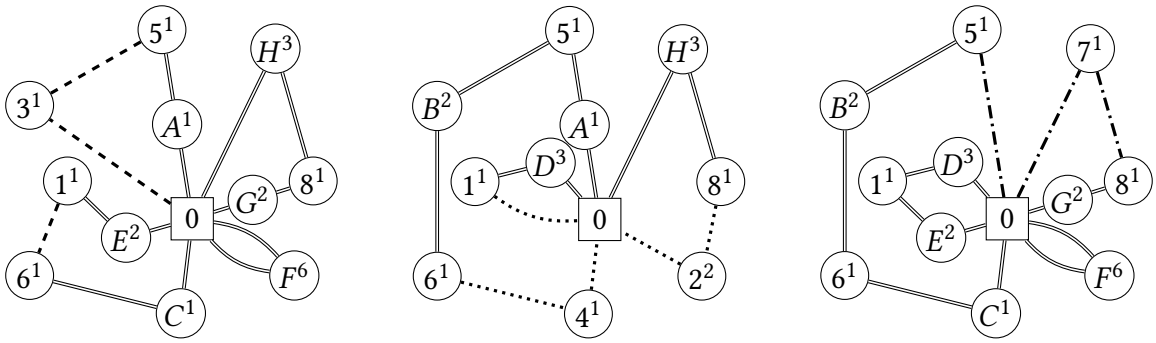


Figure 3.3: An augmented solution $\overline{\pi}$ of the solution π of Figure 3.1. Nodes $A, B, ..., H \in Y$ are augmented nodes. Nodes $1, 5, 6, 8 \in N'$ are conjunction customers. Observe that the augmented nodes' demands are defined such that there is no residual capacity. Other augmented solutions can be obtained by changing the demands of the augmented nodes as long as there is no residual capacity and the demand of nodes B, C, H is at least 1.



(a) π_1 **(b)** π_2 **(c)** π_3

Figure 3.4: Sub-problems' initial solutions π_1, π_2, π_3.



(a) π_1^* **(b)** π_2^* **(c)** π_3^*

Figure 3.5: Sub-problems' final solutions $\pi_1^*, \pi_2^*, \pi_3^*$. Observe that the partial solutions of π_2^* and π_3^* shown in Figure 3.2 are not feasible for these augmented sub-problems because they would violate the vehicle capacity in the corresponding sub-problems.

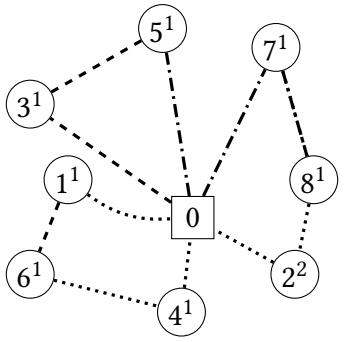


Figure 3.6: Feasible resulting solution π^* obtained by merging the free edges of solutions $\pi_1^*, \pi_2^*, \pi_3^*$.

3.4 Proof of Feasibility

In this section, we prove that the sequence-based decomposition described in Section 3.3 is a valid decomposition scheme for the CVRP, i.e., the resulting solutions obtained from such decomposition scheme (i) cannot have routes exceeding the vehicle capacity and (ii) cannot have subtours.

We show in the following that both (i) and (ii) are corollaries of a common lemma, Lemma 3.4.1. Let us first introduce the following definitions.

Definition 1. *Let S be a sequence of nodes. We define $L(S) = \sum_{v \in S} q_v$, as the cumulative demand of the nodes in S.*

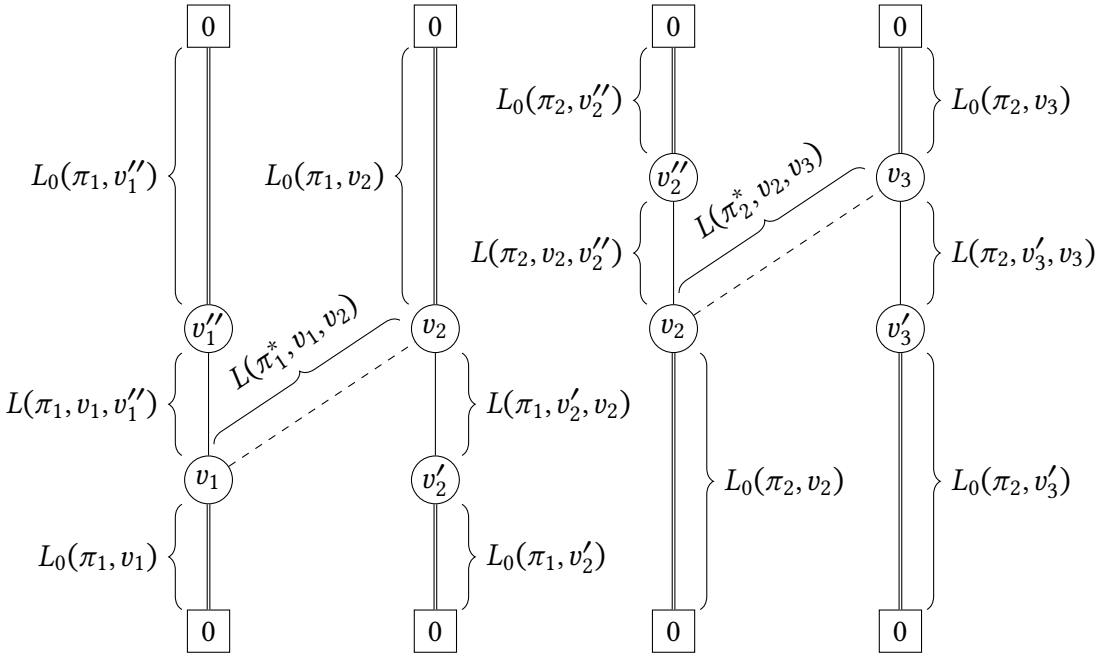
Definition 2. *Let π_i be a solution of sub-problem $P_i, i \in [1, M]$, and $v_1, v_2 \in N'$ two conjunction customers that belong to the same route. We define $L(\pi_i, v_1, v_2)$ as the cumulative demand of the customers in between v_1 and v_2 in solution π_i.*

Definition 3. *Let π_i be a solution of sub-problem $P_i, i \in [1, M]$, and let $v \in N'$ be a conjunction customer that belongs to a route such that one of the two edges incident to v is not fixed. We define $L_0(\pi_i, v)$ as the cumulative demand of the (augmented) nodes in between the depot and node v visited by the unique path with only fixed edges.*

For example, taking as reference the solutions in Figure 3.4, we have that $L(\pi_1, 5, 6) = q_3 = 1$, $L_0(\pi_1, 5) = q_A = 1$ and $L_0(\pi_3, 5) = q_C + q_6 + q_B = 4$. Observe also that $L_0(\pi_1, 5) + q_5 + L_0(\pi_3, 5) = 6 = Q$. It is then straightforward to state the following property for the conjunction customers.

Property 1. *Let $v \in N'$ be a conjunction customer and $R^i \in \pi_i, R^j \in \pi_j$, $i, j \in [1, M], i \neq j$, be the two routes containing node v in which one of the two edges incident to v is not fixed. Without loss of generality, let $R^i = (0, a_1, ..., a_m, v, ..., 0)$, and $R^j = (0, ..., v, a_{m+1}, ..., a_{m+k}, 0)$, with*

$m, k \geq 1$, *and* $a_1, ..., a_{m+k} \in Y \cup N'$. *Then, observe that* $\overline{R} = (0, a_1, ..., a_m, v, a_{m+1}, ..., a_{m+k}, 0) \in \overline{\pi}$ *is an augmented route defined during the augmentation phase, hence, with no residual capacity. Therefore,* $L_0(\pi_i, v) + q_v + L_0(\pi_j, v) = L(\overline{R}) = Q$.



(a) P_1 **(b)** P_2

Figure 3.7: Illustrative example of the functions L and L_0 for two sub-problems P_1 and P_2. Double-line edges denote fixed paths. Single-line edges denote the partial solutions of the initial solutions π_1 and π_2, respectively, for sub-problems P_1 and P_2. Dashed-line edges denote the partial solutions of the final solutions π_1^* and π_2^* that connect customer v_1 with customer v_2 in sub-problem P_1, and customer v_2 with customer v_3 in sub-problem P_2.

Figure 3.7 illustrates the meaning of the functions L and L_0. Observe that, by definition, π_1^* and π_2^* are feasible respectively for sub-problem P_1 and P_2, i.e., $L_0(\pi_1, v_1) + q_{v_1} + L(\pi_1^*, v_1, v_2) + q_{v_2} + L_0(\pi_1, v_2) \leq Q$ and $L_0(\pi_2, v_2) + q_{v_2} + L(\pi_2^*, v_2, v_3) + q_{v_3} + L_0(\pi_2, v_3) \leq Q$. In addition, Property 1 on the conjunction customer v_2 says that $L_0(\pi_1, v_2) + q_{v_2} + L_0(\pi_2, v_2) = Q$. Therefore, we have that $L_0(\pi_1, v_1) + q_{v_1} + L(\pi_1^*, v_1, v_2) \leq L_0(\pi_2, v_2)$ and $L_0(\pi_2, v_3) + q_{v_3} + L(\pi_2^*, v_2, v_3) \leq L_0(\pi_1, v_2)$. By combining the above relations, we obtain an upper bound on the cumulative demand of the path obtained by concatenating the path that goes from v_1 to v_2 in solution π_1^* with the path that goes from v_2 to v_3 in solution π_2^*, i.e.,: $q_{v_1} + L(\pi_1^*, v_1, v_2) + q_{v_2} + L(\pi_2^*, v_2, v_3) + q_{v_3} \leq L_0(\pi_2, v_2) - L_0(\pi_1, v_1) + q_{v_2} + L_0(\pi_1, v_2) - L_0(\pi_2, v_3) = Q - L_0(\pi_1, v_1) - L_0(\pi_2, v_3)$.

The above inequality provides an upper bound on the cumulative demand of such a path that, given the initial sub-problems' solutions, only depends on the endpoint customers. The following lemma generalizes such upper bound when more than two

paths are concatenated.

Lemma 3.4.1. *Let S be an elementary path resulting from the concatenation of the partial solutions that starts from node $v_1 \in N'$, ends at node $v_{k+1} \in N'$, $k \in \mathbb{N}_{>1}$, passing through the conjunction customers $v_2, ..., v_k \in N'$ in this order. Let $p[j], j \in [1, k]$, be the index of the sub-problem whose partial solution connect v_j with v_{j+1}. Then, we have that: $L(S) = \sum_{j=1}^{k} \left(q_{v_j} + L(\pi^*_{p[j]}, v_j, v_{j+1})\right) + q_{v_k+1} \leq Q - L_0(\pi_{p[1]}, v_1) - L_0(\pi_{p[k]}, v_{k+1})$.*

Proof. Without loss of generality, let $p[k+1]$ be the index of the sub-problem whose partial solution connects v_{k+1} with node $v \in N' \cup \{0\}, v \neq v_k$. Then, since $L_0(\pi_{p[j]}, v_{j+1}) + q_{v_{j+1}} + L_0(\pi_{p[j+1]}, v_{j+1}) = Q$, and $L_0(\pi_{p[j]}, v_j) + q_{v_j} + L(\pi^*_{p[j]}, v_j, v_{j+1}) + q_{v_{j+1}} + L_0(\pi_{p[j]}, v_{j+1}) \leq Q \; \forall \; j \in [1, k]$, because there is no residual capacity in $\overline{\pi}$, it follows that:

$$L_0(\pi_{p[j]}, v_j) + q_{v_j} + L(\pi^*_{p[j]}, v_j, v_{j+1}) \leq L_0(\pi_{p[j+1]}, v_{j+1}) \quad \forall \; j \in [1, k] \tag{3.1}$$

Then, we have that:

$$\begin{aligned}
L(S) &= \sum_{j=1}^{k} \left(q_{v_j} + L(\pi^*_{p[j]}, v_j, v_{j+1})\right) + q_{v_{k+1}} && (3.2)\\
&\leq \sum_{j=1}^{k} L_0(\pi_{p[j+1]}, v_{j+1}) - L_0(\pi_{p[j]}, v_j) + q_{v_{k+1}} && (3.3)\\
&= L_0(\pi_{p[k+1]}, v_{k+1}) - L_0(\pi_{p[1]}, v_1) + q_{v_{k+1}} && (3.4)\\
&= Q - L_0(\pi_{p[k]}, v_{k+1}) - q_{v_{k+1}} - L_0(\pi_{p[1]}, v_1) + q_{v_{k+1}} && (3.5)\\
&= Q - L_0(\pi_{p[k]}, v_{k+1}) - L_0(\pi_{p[1]}, v_1) && (3.6)
\end{aligned}$$

In which we used inequality (3.1) to get (3.3) and Property 1 to get (3.6). □

Figure 3.7 illustrates the case in which $k = 2$ paths are concatenated. In this case, Lemma 3.4.1 states that $q_{v_1} + L(\pi^*_{p[1]}, v_1, v_2) + q_{v_2} + L(\pi^*_{p[2]}, v_2, v_3) + q_{v_3} \leq Q - L_0(\pi_{p[1]}, v_1) - L_0(\pi_{p[2]}, v_3)$.

Corollary 3.4.2. *The solution π^* obtained by the sequence-based decomposition cannot have routes that exceed the vehicle capacity.*

Proof. Let R be a route that starts from the depot, goes through the conjunction customers $v_1, v_2, ..., v_k \in N'$ in this order, and then returns back to the depot. Let $p[j], j \in [1, k-1]$, be the index of the sub-problem whose partial solution connects v_j with v_{j+1}. Let A and B be respectively the cumulative demand of the partial solution that connects the depot with v_1, and the depot with v_k, excluding respectively the demands of v_1 and v_k. Since there is no residual capacity in the augmented routes, $A \leq L_0(\pi_{p[1]}, v_1)$ and $B \leq L_0(\pi_{p[k-1]}, v_k)$. Then, we have that:

$$L(R) = A + \sum_{j=1}^{k-1} \left(q_{v_j} + L(\pi^*_{p[j]}, v_j, v_{j+1})\right) + q_{v_k} + B \tag{3.7}$$

$$\leq L_0(\pi_{p[1]}, v_1) + Q - L_0(\pi_{p[1]}, v_1) - L_0(\pi_{p[k-1]}, v_k) + L_0(\pi_{p[k-1]}, v_k) \tag{3.8}$$

$$= Q \tag{3.9}$$

Where we used Lemma 3.4.1 to upper bound $\sum_{j=1}^{k-1} \left(q_{v_j} + L(\pi^*_{p[j]}, v_j, v_{j+1})\right) + q_{v_k}$. □

Corollary 3.4.3. *The solution π^* obtained by the sequence-based decomposition cannot have subtours.*

Proof. Suppose by contradiction that there exists a subtour S that starts from the conjunction customer $v_1 \in N'$, goes through the conjunction customers $v_2, v_3, ..., v_k \in N'$, and then returns back to v_1. Since customers have a positive demand by definition, the cumulative demand of the customers visited by the subtour S, $L(S)$, must be greater than 0. Let $p[j], j \in [1, k-1]$, be the index of the sub-problem whose partial solution connects v_j with v_{j+1}, and let $p[k]$ be the index of the sub-problem whose partial solution connects v_k with v_1. Then, we have that:

$$L(S) = \sum_{j=1}^{k-1} \left(q_{v_j} + L(\pi^*_{p[j]}, v_j, v_{j+1})\right) + q_{v_k} + L(\pi^*_{p[k]}, v_k, v_1) \tag{3.10}$$

$$\leq Q - L_0(\pi_{p[1]}, v_1) - L_0(\pi_{p[k-1]}, v_k) + Q - L_0(\pi_{p[k]}, v_k) - L_0(\pi_{p[k]}, v_1) - q_{v_1} - q_{v_k} \tag{3.11}$$

$$= 0 \tag{3.12}$$

Where we upper bounded $\sum_{j=1}^{k-1} \left(q_j + L(\pi^*_{p[j]}, v_j, v_{j+1})\right) + q_k$ and $L(\pi^*_{p[k]}, v_k, v_1)$ using Lemma 3.4.1 and used the fact that $L_0(\pi_{p[1]}, v_1) + q_{v_1} + L_0(\pi_{p[k]}, v_1) = L_0(\pi_{p[k-1]}, v_k) + q_{v_k} + L_0(\pi_{p[k]}, v_k) = Q$ (Property 1). Since this result contradicts the hypothesis that $L(S) > 0$, a subtour cannot occur. □

Remark 1. *If customers with null demands are allowed, subtours with a cumulative demand equal to 0 can occur. However, in this case, we can always transform the subtour into a feasible route by adding a visit to the depot between two consecutively visited customers.*

Remark 2. *Let $z(x)$ be the objective function of solution x. Then, $z(\pi^*) - z(\pi) = \sum_{i=1}^{M} z(\pi^*_i) - z(\pi_i)$.*

3.5 Computational Experiments

In this section, we evaluate a basic implementation of the proposed sequence-based decomposition scheme on the Belgium dataset introduced by Arnold et al. (2019), employing two different partitioning methods. An equivalent route-based decomposition

is used as a reference to establish whether the sequence-based decomposition can be useful in practice.

The Belgium dataset, hereafter referred to as dataset $\mathbb{B}$, consists of 10 very large-scale instances ranging from 3000 to 30000 customers, based on real parcel distributions in Belgium. Table 3.1 shows the main characteristics of each instance, namely, the name, the number of customers, the vehicle capacity, the minimum number of routes required to serve all customers defined as $\lceil \sum_{v \in N} q_v / Q \rceil$, and the depot location, either central (C) or eccentric (E). Each customer demand is set equal to one, two, or three, respectively, with probability 0.5, 0.3, and 0.2. Half of the instances have a low vehicle capacity, a high number of routes, and a centrally located depot, and are denoted as $\mathbb{B}1$ in the following. In contrast, the other half of the instances, denoted as $\mathbb{B}2$, have a high vehicle capacity, a small number of routes, and an eccentric depot.

In Section 3.5.1, we describe the implementation details of the evaluated methods. In Sections 3.5.2 and 3.5.3, we discuss, respectively, the quality of the resulting solutions and the achieved speedups. Section 3.5.4 presents an analysis of the overall performance of the evaluated methods, while Section 3.5.5 reports additional relevant statistics.

Instance	$\lvert N \rvert$	Q	routes	depot
Leuven1	3000	25	203	C
Antwerp1	6000	30	343	C
Ghent1	10000	35	485	C
Brussels1	15000	50	512	C
Flanders1	20000	50	683	C
Leuven2	4000	150	46	E
Antwerp2	7000	100	120	E
Ghent2	11000	170	110	E
Brussels2	16000	150	182	E
Flanders2	30000	200	256	E

Table 3.1: Features dataset $\mathbb{B} = \mathbb{B}1 \cup \mathbb{B}2$.

3.5.1 Implementation Details

The proposed sequence-based decomposition has been implemented using the master–worker paradigm. The *master* process is responsible for generating the initial solution and the sub-problems to assign to the workers, hereafter referred to as *solvers*, along with the corresponding initial solutions inherited by the master. Each solver optimizes the received sub-problem for It_{sub} iterations, and returns the best-found solution to the

master, which collects all solutions and recombines them to form a new feasible solution. This process is iterated until a prescribed number of iterations, It_{tot}, is reached.

Sub-problems are generated as follows. Each route is divided into two sequences of edges: one from the depot to the farthest customer, and the other one from the farthest customer back to the depot. A clustering procedure is then applied to obtain a partition of the sequences. Two clustering methods have been tested: the *barycenter clustering* described in Santini et al. (2023), and an alternative *iterative clustering*. Specifically, the barycenter clustering represents each sequence by its geometric barycenter and then applies the k-means algorithm (Lloyd 1982) to cluster these points (i.e., the sequences). In our preliminary computational experiments, we observed that barycenter clustering may produce highly unbalanced clusters. To address this issue, we proposed an iterative clustering approach, that, differently to the barycenter clustering, assign one sequence at a time to each cluster in an iterative way. At each iteration, the chosen sequence is the one corresponding to the closest customer with respect to the current centroid not yet assigned.

For each route, each unit of residual capacity is distributed among the augmented nodes randomly, with probabilities proportional to the number of customers replaced by each augmented node.

To ensure a fair comparison, the route-based decomposition is identical to the proposed sequence-based decomposition, except that routes are not split before clustering.

Each solver optimizes its assigned sub-problem with the FILO2 heuristic (Accorsi and Vigo 2024) for $It_{sub} = 2500$ (coreopt) iterations. The total number of iterations is set equal to $It_{tot} = 10^6$, corresponding to the "long" version of FILO2. Each instance is evaluated ten times with ten different random seeds by each method. The best known solution values (BKS) for the $\mathbb{B}$ instances were taken from CVRPLIB (2025) at the time of writing. The gaps are computed as $100 \cdot (z - BKS)/BKS$ where z is the final solution value obtained by the algorithms.

The algorithm was implemented in C++ and compiled using g++ 11.4. The experiments were performed on a 64-bit GNU/Linux Ubuntu 22.04 workstation with an Intel Xeon Gold 6254 CPU, running at 3.1 GHz, and with 384 GB of RAM.

All proposed methods are tested using 4, 8, 12 and 16 threads, and are compared to the "long" version of the sequential algorithm FILO2 executed for $It_{tot} = 10^6$ coreopt iterations, available at `https://github.com/acco93/filo2`. The results of FILO2 presented in this work were obtained by running the algorithm on the same machine described above.

3.5.2 Solutions' Quality

In this section, we compare the average gaps obtained by the route-based decomposition with those obtained by the sequence-based decomposition. Tables 3.2 and 3.3 report the average gaps obtained using respectively, the barycenter clustering and the iterative clustering as partitioning methods. Column 1 reports the average gaps of the sequential algorithm FILO2. Column n > 1 reports the average gaps of the [route | sequence]-based decomposition scheme using n threads, i.e., n-1 solvers. For each instance, the smallest gap among all parallel configurations is presented in bold.

Overall, the smallest average gap on dataset $\mathbb{B}$, as well as on sub-datasets $\mathbb{B}\mathbb{1}$ and $\mathbb{B}\mathbb{2}$, was achieved by the sequence-based decomposition combined with the barycenter clustering and 8 threads, yielding a gap of 0.424% on dataset $\mathbb{B}$, compared to 0.363% of FILO2. Observe that a similar average gap of 0.428% was obtained by the sequence-based decomposition combined with the iterative clustering and 12 threads. When the barycenter clustering is used, the sequence-based decomposition achieves the best gap in 7 out of 10 instances. When the iterative clustering is used, the sequence-based decomposition achieves the best gap in all 10 instances.

It is particularly insightful to analyze the instances with the smallest and largest number of routes – Leuven2 and Flanders1, respectively. Among all instances, Flanders1 is the only one where a parallel configuration was able to obtain an average gap better than the sequential algorithm FILO2 (0.305% versus 0.308%). This improvement was achieved using the sequence-based decomposition combined with the barycenter clustering and 16 threads. Conversely, Leuven2 yielded by far the worst gap, 1.308%, whereas FILO2 achieves an average gap of 0.300%. Specifically, this result was achieved using the route-based decomposition combined with the iterative clustering and 16 threads. Observe that when using 16 threads on Leuven2, the resulting sub-problems contain only 3.06 routes on average, making it very challenging to optimize effectively. A similar gap of 1.287% was also achieved by the route-based decomposition combined with the barycenter clustering and 16 threads. It is noteworthy to observe that these gaps are significantly reduced by the sequence-based decomposition to 0.754% and 1.158%, respectively.

Unexpectedly, the configurations using 4 threads exhibit a substantially larger average gap compared to their 8-thread counterparts.

Instance		route-based				sequence-based			
	1	4	8	12	16	4	8	12	16
Leuven1	0.195	0.353	**0.267**	0.296	0.328	0.329	0.283	0.303	0.286
Antwerp1	0.225	0.359	0.312	0.288	0.299	0.346	0.297	**0.262**	0.270
Ghent1	0.269	0.354	0.336	0.343	0.327	0.379	0.329	0.335	**0.313**
Brussels1	0.379	0.497	0.443	0.431	0.439	0.516	0.436	**0.411**	0.434
Flanders1	0.308	0.363	0.326	0.323	0.314	0.368	0.331	0.316	**0.305**
$\mathbb{B}1$	0.275	0.385	0.337	0.336	0.341	0.388	0.335	0.325	**0.322**
Leuven2	0.300	**0.411**	0.534	0.699	1.287	0.469	0.463	0.818	1.158
Antwerp2	0.387	0.543	0.499	0.499	0.539	0.533	**0.436**	0.525	0.566
Ghent2	0.395	0.482	0.462	0.573	0.775	0.463	**0.446**	0.549	0.754
Brussels2	0.535	0.714	0.588	0.641	0.722	0.664	**0.575**	0.612	0.701
Flanders2	0.634	0.757	**0.635**	0.691	0.716	0.714	0.649	0.663	0.659
$\mathbb{B}2$	0.450	0.581	0.544	0.621	0.808	0.569	**0.514**	0.633	0.768
$\mathbb{B}$	0.363	0.483	0.440	0.478	0.575	0.478	**0.424**	0.479	0.545

Table 3.2: Average gaps on the Belgium dataset using the barycenter clustering. Column 1 indicates the average gaps of the sequential algorithm FILO2.

Instance		route-based				sequence-based			
	1	4	8	12	16	4	8	12	16
Leuven1	0.195	0.396	0.299	0.288	0.320	0.351	**0.241**	0.282	0.318
Antwerp1	0.225	0.365	0.312	0.296	0.291	0.378	0.316	**0.284**	0.293
Ghent1	0.269	0.409	0.352	0.330	0.342	0.420	0.368	0.336	**0.317**
Brussels1	0.379	0.559	0.475	0.437	0.433	0.560	0.477	0.446	**0.422**
Flanders1	0.308	0.399	0.376	0.339	0.343	0.399	0.360	0.335	**0.333**
$\mathbb{B}1$	0.275	0.426	0.363	0.338	0.346	0.422	0.352	**0.337**	**0.337**
Leuven2	0.300	0.460	0.443	0.803	1.308	0.410	**0.398**	0.556	0.754
Antwerp2	0.387	0.579	0.455	0.464	0.501	0.587	0.455	**0.403**	0.446
Ghent2	0.395	0.518	0.446	0.450	0.559	0.546	0.417	**0.402**	0.442
Brussels2	0.535	0.807	0.601	0.586	0.616	0.764	0.587	0.585	**0.578**
Flanders2	0.634	0.832	0.658	0.666	0.702	0.835	0.689	**0.654**	0.656
$\mathbb{B}2$	0.450	0.639	0.521	0.594	0.737	0.628	**0.509**	0.520	0.575
$\mathbb{B}$	0.363	0.533	0.442	0.466	0.542	0.525	0.431	**0.428**	0.456

Table 3.3: Average gaps on the Belgium dataset using the iterative clustering. Column 1 indicates the average gaps of the sequential algorithm FILO2.

3.5.3 Computing Times Speedup

In this section, we compare the average speedups obtained by the route-based decomposition with those obtained by the sequence-based decomposition. Tables 3.4 and 3.5 report the average speedups obtained using the barycenter and iterative clustering respectively. Column 1 reports the average computing time of the sequential algorithm FILO2. Column n > 1 reports the average speedup with respect to FILO2 of the [route | sequence]-based decomposition scheme using n threads, i.e., n-1 solvers. For each instance, the largest speedup among all parallel configurations is presented in bold.

For both partitioning methods and across all instances, the highest speedups are achieved with 16 threads, with a maximum of 25.6 observed for the Leuven2 instance using the route-based decomposition combined with the iterative clustering and 16 threads. The smallest reported speedup is 2.4, observed for the Leuven1 and Flanders1 instances, both achieved using the sequence-based decomposition combined with the barycenter clustering and 4 threads.

On average, when the barycenter clustering is used, the sequence-based decomposition achieves higher speedups than the route-based decomposition with 8, 12, and 16 threads, but not with 4 threads. In particular, for the $\mathbb{B}\mathbb{1}$ instances, its speedup is roughly twice as large with 12 and 16 threads. The average speedup on the $\mathbb{B}$ instances using 16 threads is respectively 8.5 and 11.0 for the route-based and sequence-based decomposition. Notably, for the Antwerp2 instance, the route-based decomposition attains a maximum speedup of only 3.5, compared to 7.7 for the sequence-based decomposition.

In contrast, when the iterative clustering is used, the route-based decomposition consistently yields slightly higher speedups than the sequence-based decomposition across all thread configurations, achieving an average speedup of 17.5 on the $\mathbb{B}$ instances using 16 threads, compared to 15.5 for the sequence-based.

Instance		route-based				sequence-based			
	1	4	8	12	16	4	8	12	16
Leuven1	729	2.6	4.0	4.4	4.8	2.4	5.0	6.7	**8.5**
Antwerp1	810	3.1	2.6	3.0	6.3	2.8	6.0	9.4	**11.5**
Ghent1	872	2.8	3.5	4.0	5.8	2.5	5.6	7.9	**12.2**
Brussels1	882	2.9	3.3	3.0	4.4	2.6	5.8	9.4	**11.8**
Flanders1	1245	2.6	5.1	5.9	7.6	2.4	5.2	7.8	**9.8**
$\mathbb{B}1$	907	2.8	3.7	4.0	5.8	2.5	5.5	8.2	**10.7**
Leuven2	1021	3.1	7.1	11.6	**14.5**	2.8	6.2	9.5	11.1
Antwerp2	919	3.0	2.9	3.5	3.4	2.9	4.6	5.7	**7.7**
Ghent2	1075	2.9	6.8	11.1	**14.7**	2.7	6.1	9.9	13.7
Brussels2	1036	3.0	6.8	10.6	12.0	2.7	6.1	9.5	**12.9**
Flanders2	1479	2.7	5.8	9.2	**11.9**	2.5	5.3	8.2	10.9
$\mathbb{B}2$	1106	2.9	5.9	9.2	**11.3**	2.7	5.7	8.6	**11.3**
$\mathbb{B}$	1007	2.9	4.8	6.6	8.5	2.6	5.6	8.4	**11.0**

Table 3.4: Average speedups on the Belgium dataset using the barycenter clustering. Column 1 indicates the average times, in seconds, of the reference sequential algorithm FILO2.

Instance		route-based				sequence-based			
	1	4	8	12	16	4	8	12	16
Leuven1	729	3.5	8.0	11.6	**14.9**	3.0	7.2	10.8	13.9
Antwerp1	810	3.4	8.2	12.7	**16.0**	2.9	7.1	11.3	15.3
Ghent1	872	3.3	8.1	12.7	**15.8**	2.8	6.9	11.3	15.5
Brussels1	882	3.3	8.1	13.0	**17.7**	2.8	6.9	11.2	15.6
Flanders1	1245	2.9	6.9	11.0	**14.9**	2.5	6.0	9.5	12.9
$\mathbb{B}1$	907	3.3	7.9	12.2	**15.9**	2.8	6.8	10.8	14.7
Leuven2	1021	4.2	10.6	17.4	**25.6**	3.5	8.3	13.3	18.3
Antwerp2	919	3.5	8.4	12.7	15.2	3.1	7.4	11.9	**16.0**
Ghent2	1075	3.7	9.0	14.8	**20.9**	3.2	7.7	12.4	17.4
Brussels2	1036	3.6	8.4	13.4	**18.4**	3.1	7.4	11.8	16.3
Flanders2	1479	3.3	7.3	11.3	**15.3**	2.9	6.4	9.8	13.3
$\mathbb{B}2$	1106	3.7	8.8	13.9	**19.1**	3.1	7.4	11.9	16.3
$\mathbb{B}$	1007	3.5	8.3	13.1	**17.5**	3.0	7.1	11.3	15.5

Table 3.5: Average speedups on the Belgium dataset using the iterative clustering. Column 1 indicates the average times, in seconds, of the reference sequential algorithm FILO2.

3.5.4 Overall Performance

In this section, we summarize the overall performance of the proposed methods. Figure 3.8 compares all evaluated parallel configurations in terms of their average computing times (in logarithmic scale) and average gaps with respect to the best-known solutions.

The sequential algorithm FILO2 is represented by a triangle. Route-based (sequence-based) configurations are denoted with a square (circle) and a label Rn (Sn), where n indicates the number of threads used. Black points denote the use of the barycenter clustering, whereas white points denote the use of the iterative clustering. For both the barycenter clustering and iterative clustering configurations, the Pareto frontier representing the non-dominated configurations is displayed. First of all, it can be observed that the 4-thread configurations behave as outliers, as they are almost entirely dominated by all configurations with 8 and 12 threads. The reason for this behavior is not yet clear and requires further investigation, since the opposite behavior is expected. Second, the configurations based on the iterative clustering appear to be significantly better than the barycenter clustering counterparts, particularly when many threads are used.

All configurations on the two Pareto frontiers are sequence-based except for one (R16 with the iterative clustering), which nonetheless exhibits a gap of 0.542%, which is significantly higher than the corresponding sequence-based configuration, which has similar computing times but a gap of 0.456%.

Overall, a deterioration in solution quality can be observed as the number of threads increases (excluding outliers). However, this quality deterioration is less pronounced for the sequence-based decomposition, especially when combined with the iterative clustering.

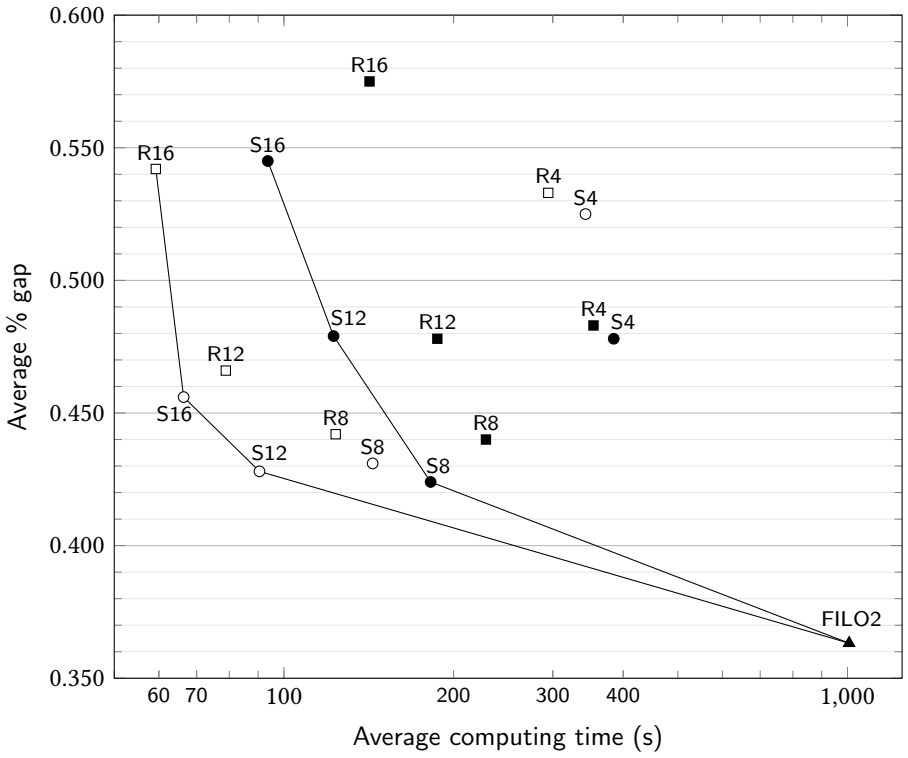


Figure 3.8: Performance plot of the evaluated methods. R stands for route-based, S for sequence-based. Black points denote the use of the barycenter clustering. White points denote the use of the iterative clustering.

3.5.5 Relevant Statistics

Table 3.6 reports some relevant statistics for each tested configuration, where R stands for route-based and S for sequence-based. Columns total and % idle report, respectively, the average total computing time (in seconds) and the average percentage of solvers' idle time. It can be observed that the percentage of idle time increases as the number of threads grows. For instance, in the case of the route-based decomposition combined with the barycenter clustering and 16 threads, 15 solvers remain idle for 62% of the total time, leading to a cumulative loss of more than 1300 seconds of computing time. The use of the iterative clustering significantly reduces these idle times to 24%, although they remain non-negligible.

Columns comm, partition, and init report the overall computing time overhead, expressed in seconds, required by the inter-process communication, by the solutions' partitioning, and by the sub-problems' initialization. The communication overhead is negligible and less than 0.2 seconds. The partitioning overhead is negligible for the barycenter clustering, less so for the iterative clustering, requiring up to 10 seconds when combined with the sequence-based decomposition and 4 threads. The sub-problems' initialization overhead is the most computationally demanding and appears to depend primarily on

the number of threads. These times are influenced by the solution method employed within the solvers. In our case, FILO2 requires some pre-processing in order to be very efficient in the later stages. Changing algorithm could alleviate this overhead. Note that both partition and init exhibit a linear decay as the number of threads increases. These two trends can be explained by the fact that the total number of produced partitions is given by $It_{tot}/(threads \cdot It_{sub})$ and that sub-problems are initialized in parallel.

Columns complete and partial indicate the average number of complete and partial routes in the sub-problems. Column % RSD denotes the average relative standard deviation of the generated sub-problems' dimensions. The larger this number is, the more unbalanced the sub-problems are. Observe that the iterative clustering configurations have much lower values than the barycenter clustering ones, leading consequently to much lower idle times.

		threads	time		overhead			sub-problems		
			total	% idle	comm	partition	init	complete	partial	% RSD
barycenter clustering	R	4	354.29	13.59	0.12	0.09	47.85	98.06	0.00	53.23
		8	228.30	44.01	0.07	0.05	24.92	42.03	0.00	73.63
		12	187.20	58.49	0.05	0.03	16.98	26.74	0.00	78.38
		16	141.89	62.13	0.05	0.03	13.14	19.61	0.00	80.28
	S	4	384.88	11.96	0.12	0.19	52.92	96.71	2.71	47.64
		8	182.14	26.46	0.06	0.13	27.37	40.90	2.29	59.73
		12	122.50	35.45	0.05	0.10	18.32	25.81	1.94	64.02
		16	93.63	40.91	0.04	0.09	13.70	18.81	1.71	65.69
iterative clustering	R	4	294.51	5.77	0.11	4.92	52.01	98.06	0.00	1.33
		8	123.56	12.89	0.06	2.18	27.78	42.02	0.00	2.51
		12	78.89	18.68	0.04	1.40	18.54	26.74	0.00	3.18
		16	59.26	24.30	0.04	1.03	13.75	19.61	0.00	3.70
	S	4	342.93	5.54	0.12	9.88	57.39	95.36	5.39	4.10
		8	143.71	12.09	0.06	4.46	29.63	39.57	4.90	8.69
		12	90.50	16.41	0.05	2.90	19.55	24.64	4.21	12.29
		16	66.39	20.07	0.04	2.16	14.42	17.73	3.76	15.12

Table 3.6: Statistics related to the computing times, overhead, and sub-problems.

3.6 Conclusions and Future Work

In this work, we presented a new decomposition scheme for the CVRP, named *sequence-based* decomposition, that generalizes the *route-based* decomposition by allowing different sub-problems to concurrently optimize distinct portions of the same route.

Computational experiments showed that even a basic sequence-based decomposition

helps mitigate the deterioration in solution quality typically observed with the route-based decomposition as the number of threads increases.

The proposed sequence-based decomposition, built upon the state-of-the-art sequential algorithm FILO2, was evaluated on ten very large instances, achieving slightly higher average gaps but with roughly linear speedups. For one instance, it was possible to achieve a slightly better average gap than FILO2 (0.305% instead of 0.308%) with an average speedup of 9.8 (127 seconds instead of 1245 seconds).

The solutions' quality is strongly influenced by the number of routes, and instances with fewer routes are generally more challenging to optimize. We now outline several strategies to address this bottleneck that will be explored in the future, some of which are unique to the sequence-based decomposition.

In our implementation, each route is divided into two sequences; however, this restricts the number of partial solutions available in the sub-problems. Increasing the number of sequences during the partitioning phase could improve the solutions' quality for the instances with fewer routes.

We have shown that, to guarantee the feasibility of the resulting solution, the augmented routes must have no residual capacity. A speculative approach, however, is to allow some residual capacity in order to create less constrained sub-problems. The merged solution obtained in this way may be infeasible. Rather than discarding all sub-solutions, one can iteratively merge each sub-solution with the current feasible solution, rejecting only those that lead to infeasibility.

When solvers terminate earlier, they remain idle. Instead, they could run additional iterations until the last solver completed. This should improve the gap without affecting the computing times. We tried using a time limit for the sub-problems, but without achieving satisfactory results.

Observe that the master process remains idle most of the time. A possibility is to allow the master to optimize the current solution in parallel with the solvers, and then select the most improved solution between the one proposed by the master and the one proposed by the solvers. Also for this approach, the gap should improve without affecting the computing times.

Finally, in this work, we used FILO2 as sequential algorithm to generate the initial solution and as optimizer inside solvers. Alternative solution methods could yield different results and will be investigated in future work.

Chapter 4

Routing and Wavelength Assignment Problem

[1]

In this work, we consider the problem of allocating wavelengths of a given optical network in order to serve a set of connection requests. In order to provide a robust allocation, communication must be ensured even in case of failure of some link, which is obtained by defining, for each connection request, two disjoint paths in the network. Each connection uses its *working path* by default, and is rerouted to its unique *recovery path* in case the working path is unavailable. As allocating dedicated resources to each connection may be highly inefficient, we adopt a shared protection policy, in which the same wavelength and path can be shared among different connections, provided that no collision arises. We discuss alternative relaxations for the problem based on Integer Linear Programming formulations, and introduce a metaheuristic algorithm based on the iterated local search paradigm. Computational experiments on realistic data show that our algorithms are able to produce feasible solutions with tight optimality gaps.

4.1 Introduction

Optical fiber communication is a method of transmitting information by sending pulses of light through an optical fiber. This technology offers several advantages, including flexibility, relatively low noise, and long transmission range. Typically, transmission is operated according to a wavelength division multiplexing scheme, which allows the transmission of multiple signals onto the same fiber by using different wavelengths of laser light.

[1] This chapter is based on Cavaliere et al. (2025)

Optical networks are obtained by combining fibers so as to allow connection between points that are geographically far from each other. Given the complexity of modern networks, which may include different optical components, as well as the considerable investments for establishing a connection between relevant points of the network, an optimal design of the physical topology of the network is of paramount importance. The problem of defining the links and the capacity to be installed on each link of a given network to obtain a certain level of service involves decisions at a strategic level and is known in the literature as the *network design problem*.

At a tactical level, once the network has been set up, the network operator receives a set of connection demands, each between a pair of points in the network. The *Routing and Wavelength Assignment* problem (RWA) asks to determine a connection path for each such demand by optimizing the use of the available resources. Typically, for a given demand, the same wavelength has to be used along all fiber links of the path unless converting components are installed.

The design of survivable networks is a crucial issue for fiber transmissions, where the large amount of data that fiber links can transmit results in networks that are typically very sparse compared with those using traditional bandwidth-limited technologies. This issue is addressed in the design phase by introducing some redundancy in the network. Similarly, survivability is crucial when the routing phase is considered. Indeed, multiplexing in modern transmission systems involves a large number of wavelengths, each operating at a very high frequency. As a result, the loss of connection on a fiber, due to a failure or to a major reconfiguration of the routing, even if lasting a few seconds, could result in a huge amount of data to be undeliverable. This motivates the request for (i) the design of survivable networks, and (ii) a robust allocation of the link capacities to the demands, so as to allow data transmission among all origin-destination pairs even in case some unpredictable events occur, e.g., a failure on some links of the network. In this paper we address request (ii), namely the robust allocation of network resources to a given set of origin-destination pairs.

Typically, failures involve a fiber break and robustness is enforced with respect to a certain model of uncertainty, defined as a set of stochastic scenarios; each scenario corresponds to the failure of some fiber links, possibly affecting the performance of the network. Given the small probability of failure for a single link, resource allocation of survivable optical networks typically requires to consider scenarios in which at most one link fails. A natural policy for allocating capacities of the network in a robust way is the so-called *dedicated path protection* scheme. In this scheme, one is required to define, for each connection demand, two edge-disjoint paths. Each path connects the associated origin-destination pair and uses a wavelength that is uniquely reserved for that demand. The first path, denoted as *working path* hereafter, is used when there is

no fault, whereas the second one, denoted as *recovery path*, is used in case of failure of any fiber link of the working path. When this happens, switching a connection from the working to the recovery path is quickly enforced by means of fiber switches. However, since a wavelength is uniquely reserved for each demand and path, only a fraction of the resources that are installed on the network are actually used simultaneously, and the resulting scheme may be highly inefficient.

A more efficient scheme is the so-called *global re-routing*, which only requires that a feasible path exists for each commodity under any failure scenario. However, this scheme implies re-routing of commodities that are not directly affected by the failure, which is highly undesirable as it introduces complex control issues and delays on the entire network.

In this paper, we consider a third scheme, known as *shared protection* policy, in which wavelengths on fibers can be shared between demands, provided no interference arises. This allows to reduce the negative effects of global re-routing while preserving an efficient usage of the resources.

4.1.1 Literature and Related Problems

Network design has been extensively studied in the scientific literature, with particular emphasis on polyhedral approaches (see, e.g., Grötschel et al. (1992), Magnanti et al. (1993, 1995) and Atamtürk (2002)). In its basic version, this problem calls for the minimum cost for installing capacities on the edges of a graph to serve a given set of demands. Capacities can typically be installed in integer multiples of a base unit, and there is no cost related to their actual usage. Later, survivable network design problems have been studied. When introducing link failures, classical approaches to network survivability are based on the use of redundancy without explicitly exploiting sharing opportunities. In this setting, the most common policy is the one requiring $K + 1$ disjoint paths for each commodity, which preserves connectivity in case of failure of up to K links (see Kerivin and Mahjoub (2005)). Flow formulations for the resulting problem are discussed in Balakrishnan et al. (2009), and valid inequalities used to strengthen the formulations are introduced. A computational comparison of alternative mathematical formulations, including models with an exponential number of variables, is presented in Gudapati et al. (2025). Another field of research focuses on protection schemes for survivable network design. Given the large amount of research on this topic, the reader is referred to Agarwal and Venkateshan (2014), where the most relevant contributions are reviewed, and alternative Integer Linear Programming (ILP) formulations are presented and computationally tested on randomly generated instances.

The routing and wavelength assignment problem we consider is defined *after* the

network has been designed and calls for an efficient use of the installed capacity under hard constraints that limit the flow on each edge. Typically, the objective function is the minimization of the number of channels used over the links, or the maximization of the number of connections that can be satisfied (see Krishnaswamy and Sivarajan (2001*a*)), or the minimization of the congestion (see Krishnaswamy and Sivarajan (2001*b*)), or the minimization of the multiplexing costs. RWA has been studied in Chlamtac et al. (1992), where complexity results and a heuristic algorithm are given. A version of the problem is considered in Ozdaglar and Bertsekas (2003), where the cost associated with an edge is given by a convex monotonically increasing function of the total flow on the edge. A survey on ILP formulations for RWA problems is given in Jaumard et al. (2007).

Fumagalli et al. (1999) consider a survivable RWA where protection is enforced by defining a suitable set of rings (of links), and the objective function minimizes the total wavelength mileage. Different strategies and schemes for network survivability are discussed in Zhou and Subramaniam (2000). ILP approaches to survivable RWA are developed in Ramamurthy and Mukherjee (1999) and Sahasrabuddhe and Mukherjee (2002). In Doshi et al. (1999), a variant of the problem is considered where, for each commodity, the working path is given and one has to determine the best backup path. Finally, Zang et al. (2003) address a survivable RWA problem in which each failure scenario may include more than one edge and involves all fibers that are buried in the same duct.

The rest of the paper is organized as follows. In Section 4.2 we formally introduce the considered problem, detail the shared protection policy and introduce an ILP formulation of the problem. In Section 4.3 we introduce relaxations allowing to derive lower bounds on the optimal value. Section 4.4 presents a heuristic algorithm based on the iterated local search paradigm. The performances of the proposed relaxations and algorithms are computationally evaluated in Section 4.5 on a benchmark of instances composed of both realistic instances and testbed from the literature. Finally, conclusions are drawn in Section 4.6.

4.2 Problem Description

The Routing and Wavelength Assignment that we consider can be formally modeled as follows. We are given an undirected multigraph $G = (V, E)$ where V is a set of nodes and E is a set of edges, possibly including parallel edges to represent alternative independent links between pairs of connected nodes. Each edge $e \in E$ corresponds to an optical fiber connection with capacity u, expressed as the number of different available wavelengths. In the remainder of this paper we will denote available wavelengths as *channels* and let $C = \{1, \dots, u\}$ be the corresponding set. Each channel used on an edge e induces a

positive cost ℓ_e.

We are also given a set $\mathcal{K}$ of origin-destination pairs, referred to as *commodities* hereafter, each corresponding to a connection demand. We assume that all commodities require the same bandwidth, and that this figure equals one channel. Although communication is bidirectional, for each commodity $k \in \mathcal{K}$, we arbitrarily refer to one of its endpoints as the source, s^k, and to the other one as the destination, t^k.

For each commodity $k \in \mathcal{K}$, connection is enforced along a *working path* W_k, i.e., a set of consecutive edges from the source to the destination node. In this paper, we assume that no wavelength conversion is allowed at the nodes. As a consequence of the wavelength continuity constraint, one has to also associate each commodity with a unique wavelength, i.e., the same channel has to be used along all edges of its path. The problem requires to route all the commodities by minimizing the cost induced by the used channels on the edges.

Robustness of the network asks to ensure connectivity also in case of disruptions. In particular, we require protection with respect to the *single failure of any edge* of the network. This is the standard requirement in the industry, where high-speed recovery algorithms assume a single-link failure model (see, e.g., Mouftah and Ho (2003), Somani (2006)). Therefore, each edge of the graph is associated with a possible *failure scenario* of the network, and connectivity has to be ensured under any failure scenario. To this aim, besides the working path, each commodity $k \in \mathcal{K}$ is associated with a *recovery path* R_k. Each commodity is always routed along its working path, unless any edge of that path fails; in this case, the commodity is routed along its recovery path. To ensure connectivity under all failure scenarios, working and recovery paths must be edge-disjoint, i.e., $W_k \cap R_k = \emptyset$. Channels on the two paths of the same commodity may be different.

Shared Protection scheme

In this paper, we consider the shared protection scheme, which allows for channel sharing on some edges, under the explicit requirement that these resources are *exclusively* reserved for a single commodity in any given scenario. To ensure this property when no failure occurs, no channel sharing is allowed among edges in the working paths. Instead, a set $\mathcal{K}' \subseteq \mathcal{K}$ of commodities can share a channel on any edge of their recovery paths provided that no failure scenario exists affecting more than one of their working paths. Given that failure scenarios are associated with the break of a single edge, sharing channels on edges of the recovery paths for commodities in $\mathcal{K}'$ is allowed if $\cap_{k \in \mathcal{K}'} W_k = \emptyset$; indeed, if commodities are associated with edge-disjoint working paths, they do not require re-routing on their recovery paths simultaneously. This configuration is depicted in Figure 4.1, where three commodities with the same source and destination are shown.

As working paths W_1, W_2 and W_3 are edge-disjoint, the recovery paths R_1, R_2 and R_3 are allowed to share the same channel on all edges.

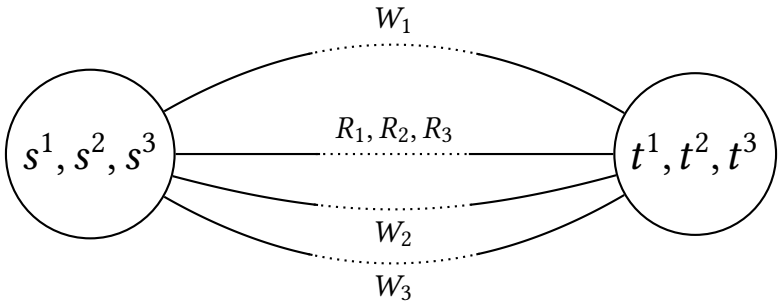


Figure 4.1: Channel sharing between recovery paths of commodities having edge-disjoint working paths.

A further sharing possibility is obtained by allowing the reuse of channels on a working path that are freed up due to a disruption. According to this policy, channel sharing is allowed between a working path of a commodity k_1 and the recovery path of another commodity k_2 in case the latter path is used only when the former fails. This is the case when a failure in W_{k_2} corresponds to a failure in W_{k_1} as well, i.e., if $W_{k_2} \subset W_{k_1}$. This configuration is depicted in Figure 4.2, where the working path of the first commodity is a superset of the working path of the second one, which uses a different channel (marked with an asterisk) for the common edges. Thus, channels on the edges of the working path of the first commodity can be shared with the recovery path of the second one.

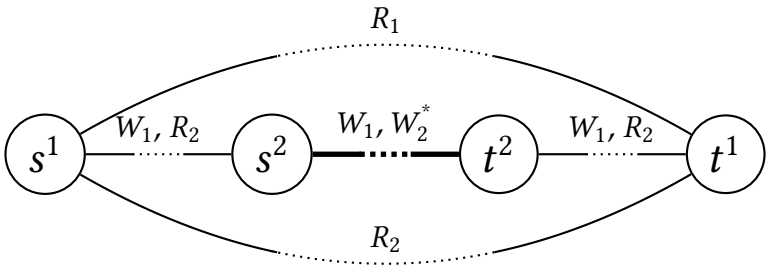


Figure 4.2: Channel sharing between the working path W_1 and the recovery path R_2. The asterisk above W_2 denotes that the channel of W_2 is different from the one of W_1. The thicker edge between s^2 and t^2 denotes that two different channels are used. Observe that this configuration does not cause any conflict since R_2 is used only if a failure in $W_2 \subset W_1$ occurs.

The impact of sharing channels on recovery paths in terms of reduced number of allocated channels is assessed by Buysse et al. (2009) to be remarkable and up to 20% for small-scale case studies.

4.2.1 Assumptions

We now introduce some assumptions, possibly discussing how to simplify the input graph in case the assumptions are not satisfied.

The first assumption is a necessary condition for the existence of a feasible solution.

Assumption 1. *Graph $\mathcal{G}$ contains, for each commodity, at least two disjoint paths from the source to the destination.*

Assumption 2. *Each vertex with degree two in graph $\mathcal{G}$ is either the source or the destination of some commodity.*

If this assumption is violated, one can define an equivalent instance in which each chain of vertices is shrunk into a single edge by removing the intermediate vertices. The cost of the resulting edge is the sum of the costs of the shrunk edges.

Assumption 3. *Graph $\mathcal{G}$ is connected and includes no cut with cardinality smaller than 2.*

If the assumption is violated, there exists a partition $(S, V \setminus S)$ of vertices of $\mathcal{G}$ such that at most one edge crosses the partition. By Assumption 1, each commodity has both the source and the destination in the same side of the partition. Thus, the graph can be decomposed into two subgraphs induced by vertex sets S and $V \setminus S$, respectively, which correspond to two independent sub-problems. This also allows to remove any vertex whose degree is smaller than 2, as it can neither be source or destination of a commodity, nor be part of a feasible path.

4.2.2 Mathematical Formulation

In this section, we introduce an ILP formulation for the problem at hand. The formulation uses the following binary decision variables

$$x_{ec}^{k} = \begin{cases} 1 \text{ if commodity } k \text{ uses channel } c \text{ and edge } e \text{ in the working path} \\ 0 \text{ otherwise} \end{cases} \quad (k \in \mathcal{K}; c \in C; e \in E)$$

$$z_{ec}^{kf} = \begin{cases} 1 \text{ if commodity } k \text{ uses channel } c \text{ and edge } e \text{ when edge } f \text{ fails} \\ 0 \text{ otherwise} \end{cases} \quad (k \in \mathcal{K}; c \in C; e, f \in E)$$

$$w_{ec} = \begin{cases} 1 \text{ if channel } c \text{ on edge } e \text{ is used} \\ 0 \text{ otherwise} \end{cases} \quad (c \in C; e \in E)$$

For each commodity $k \in \mathcal{K}$, we denote by $\boldsymbol{x}^k$ and $\boldsymbol{y}^k$ the set of associated x and y variables, respectively. As these variables represent the use of one among the $u = |C|$ available channels on a certain edge, we can encode an $s_k - t_k$ path by means of an

$|E| \times |C|$ binary vector, having one entry for each channel and edge. A path satisfying the continuity constraint has non-zero values for entries associated with one specific channel only. Let us denote by $\mathcal{P}^k \subset \{0,1\}^{|E|\times|C|}$ the set of all the incidence vectors encoding an $s_k - t_k$ path on a single channel $c \in C$. We can then model the problem as

$$\min \sum_{e\in E}\sum_{c\in C} \ell_e\, w_{ec} \tag{4.1a}$$

$$\boldsymbol{x}^k \in \mathcal{P}^k \qquad k \in \mathcal{K} \tag{4.1b}$$

$$\boldsymbol{y}^k \in \mathcal{P}^k \qquad k \in \mathcal{K} \tag{4.1c}$$

$$\sum_{c\in C}(x^k_{ec} + y^k_{ec}) \le 1 \qquad k \in K,\ e \in E \tag{4.1d}$$

$$z^{kf}_{ec} \ge x^k_{ec} - x^k_{fc} \qquad k \in \mathcal{K},\ e \in E,\ c \in C,\ f \in E,\ e \ne f \tag{4.1e}$$

$$z^{kf}_{ec} \ge y^k_{ec} + \sum_{c'\in C}(x^k_{fc'}) - 1 \qquad k \in \mathcal{K},\ e \in E,\ c \in C,\ f \in E,\ e \ne f \tag{4.1f}$$

$$w_{ec} \ge \sum_{k\in K} z^{kf}_{ec} \qquad c \in C,\ e \in E,\ f \in E,\ e \ne f \tag{4.1g}$$

$$x^k_{ec}, y^k_{ec} \in \{0,1\} \qquad k \in K,\ e \in E,\ c \in C \tag{4.1h}$$

$$z^{kf}_{ec} \in \{0,1\} \qquad k \in \mathcal{K},\ e \in E,\ c \in C,\ f \in E,\ e \ne f \tag{4.1i}$$

$$w_{ec} \in \{0,1\} \qquad e \in E,\ c \in C \tag{4.1j}$$

Inequalities (4.1d) impose that the two paths of each commodity are edge-disjoint. Constraints (4.1e) and (4.1f) express the logical link between the z variables and the working and recovery paths in case of failure. In detail, if an edge f fails, commodity k uses channel c on edge e in two cases: either W_k uses channel c on edge e while not including edge f (constraints (4.1e)), or W_k includes f (using an arbitrary channel) and R_k uses channel c on edge e (constraints (4.1f)). Finally, constraints (4.1g) define the channels that have to be used, whose total cost is minimized by the objective function.

In order to express conditions (4.1b), i.e., that for each commodity k, $\boldsymbol{x}^k$ variables define one $s_k - t_k$ path on a single channel $c \in C$, each undirected edge $e = (i,j) \in E$ can be replaced by a pair of directed arcs $A(e) = \{a, a'\}$. We denote by $\bar{\mathcal{G}} = (V, A)$ the resulting graph, where $A = \cup_{e\in E} A(e)$. Accordingly, we introduce, for each variable x^k_{ec}, two variables $\bar{x}^k_{ac}$ and $\bar{x}^k_{a'c}$ such that

$$x^k_{ec} = \bar{x}^k_{ac} + \bar{x}^k_{a'c} \tag{4.2}$$

With these additional variables, constraints (4.1b) can be explicitly imposed as

$$\sum_{a \in \delta^+(s^k)} \sum_{c \in C} \bar{x}^k_{ac} = \sum_{a \in \delta^-(t^k)} \sum_{c \in C} \bar{x}^k_{ac} = 1 \qquad k \in \mathcal{K} \tag{4.3}$$

$$\sum_{a \in \delta^-(v)} \bar{x}^k_{ac} = \sum_{a \in \delta^+(v)} \bar{x}^k_{ac} \qquad k \in \mathcal{K},\ v \in V \setminus \{s^k, t^k\}, c \in C \tag{4.4}$$

where $\delta^+(v)$ and $\delta^-(v)$ denote the forward and backward star of node v, respectively. These constraints impose that, for a given commodity k, the associated $\bar{x}^k$ variables define a unitary flow from s^k to t^k over a unique channel in C, while meeting flow conservation constraints for each channel and intermediate node.

Similarly, conditions (4.1c) are enforced by introducing variables $\bar{y}^k_{ac}$ and the corresponding constraints.

4.3 Relaxations

In this section, we introduce a hierarchy of relaxations for the problem, each producing a lower bound on the optimal solution value. We start from the weakest relaxation.

4.3.1 Unique Working, no Recovery Path

In this section, we consider relaxations that require to compute a unique working path for each commodity, but do not consider a recovery path. The resulting solution is potentially infeasible in all scenarios but one.

No Failure Channel Conversion

The first relaxation is obtained by

- disregarding failures; and
- dropping the wavelength continuity constraint.

In other words, in this relaxation, one is required to define unique working paths only and each path does not need to be assigned to a single channel (i.e., we assume that channels can be converted at each node). Besides disregarding channel allocation, this solution may result to be infeasible when the failure of any edge occurs.

As a result, one can replace the u distinct channels by a single channel with capacity u, and drop the channel index from the formulation. The resulting problem uses a decision variable w_e representing the capacity used on each edge $e \in E$, and is as follows

$$\text{(NFCC)} \qquad \min \sum_{e \in E} \ell_e \, w_e \tag{4.5a}$$

$$x^k \in \mathcal{P}^k \qquad k \in \mathcal{K} \tag{4.5b}$$

$$w_e \geq \sum_{k \in \mathcal{K}} x_e^k \qquad e \in E \tag{4.5c}$$

$$x_e^k \in \{0, 1\} \qquad k \in \mathcal{K},\ e \in E \tag{4.5d}$$

$$0 \leq w_e \leq u \qquad e \in E, \tag{4.5e}$$

where $\mathcal{P}^k$ keeps the same meaning although being defined according to a single channel.

As k is part of the input, problem (NFCC) is NP-hard by reduction from the undirected edge-disjoint paths problem (see Karp (1975)).

Rooted flow formulation Relaxation NFCC can be modeled by means of a computationally more effective ILP formulation. In this formulation, all commodities sharing the same source are aggregated into a *macro-commodity*. As the source and destination of each commodity are interchangeable, this choice affects the resulting set of macro-commodities. We model the problem of minimizing the number of macro-commodities as a vertex cover where all vertices have unitary cost. This problem is known to be NP-hard and can be formulated by the following ILP

$$\min \sum_{v \in V} \sigma_v \tag{4.6a}$$

$$\sigma_{s^k} + \sigma_{t^k} \geq 1 \qquad k \in \mathcal{K} \tag{4.6b}$$

$$\sigma_v \in \{0, 1\} \qquad v \in V. \tag{4.6c}$$

Here, each vertex $v \in V$ is associated with a variable σ_v indicating whether the vertex is the source of a macro-commodity, and constraints (4.6b) force one of the two endpoints of each commodity to be the source of a macro-commodity.

We denote by $\mathcal{M}$ the set of macro-commodities induced by the sources identified by solving model (4.6a)–(4.6c). If both endpoints of a commodity are selected, the commodity is associated to one of the two macro-commodities in an arbitrary way. For each macro-commodity $m \in \mathcal{M}$, we denote by s^m and q^m its source and the number of original commodities that are aggregated, respectively. In addition, for each node $v \in V \setminus \{s^m\}$, we let g_v^m be the number of aggregated commodities from s^m to v.

The rooted flow formulation uses integer variables x_a^m to describe the amount of flow (i.e., number of paths) on arc a for macro-commodity m and, accordingly, replaces constraints (4.5d) with $x_e^k \geq 0$. With this aggregation, constraints (4.5b) can be expressed as

$$\sum_{a \in \delta^+(s^m)} x_a^m = q^m \qquad m \in \mathcal{M} \tag{4.7a}$$

$$\sum_{a \in \delta^-(v)} x_a^m = \sum_{a \in \delta^+(v)} x_a^m + g_v^m \qquad m \in \mathcal{M},\ v \in V \setminus \{s^m\} \tag{4.7b}$$

These constraints impose that, for each macro-commodity m, exactly q^m aggregated paths start from the source s^m and exactly g_v^m of them terminate at each node $v \in V \setminus \{s^m\}$.

We conclude this section by observing that this aggregation has no straightforward extension in case channels are explicitly considered and/or when one has to simultaneously define disjoint working and recovery paths.

Worst Failure Channel Conversion

As in NFCC, we consider a relaxation in which unique working paths have to be defined only, and wavelength continuity constraints are removed. However, in this case, failures are partially taken into account by considering one scenario at a time. For each scenario, the definition of a feasible working path for each commodity provides a lower bound on the optimal solution value. Thus, a valid lower bound is obtained by considering the scenario which results in the largest solution cost, denoted as worst-case scenario. The resulting relaxation, denoted as WFCC in the following, can be solved by considering the failure of one edge $f \in E$ at a time, computing the solution cost of NFCC on graph $G_f = (V, E \setminus \{f\})$, and returning the largest value. Besides channel allocation, the corresponding solution guarantees a feasible routing in one failure scenario only.

In order to compute this relaxation, one has to solve formulation (4.5a)–(4.5d) up to $|E|$ times. In practice, the number of iterations can be reduced by observing that, given an optimal solution for a scenario f such that $w_e = 0$ for some edge e (i.e., edge e is available but is not used), one can skip the subsequent computation associated with the failure of edge e, since a feasible solution for that scenario is already available.

No Failure With Channels

We now consider a relaxation in which

- failures are disregarded; but
- the wavelength continuity constraint is maintained.

Thus, in this relaxation (denoted as NFWC) we have to explicitly consider the u distinct channels and therefore restore the channel index in the formulation. The resulting model is

$$\text{(NFWC)} \qquad \min \sum_{e \in E} \sum_{c \in C} \ell_e \, w_{ec} \tag{4.8a}$$

$$\boldsymbol{x}^k \in \mathcal{P}^k \qquad k \in \mathcal{K} \tag{4.8b}$$

$$w_{ec} \geq \sum_{k \in K} x^k_{ec} \qquad c \in \mathcal{C},\ e \in E \tag{4.8c}$$

$$x^k_{ec} \in \{0, 1\} \qquad k \in K,\ e \in E,\ c \in \mathcal{C} \tag{4.8d}$$

$$w_{ec} \in \{0, 1\} \qquad e \in E,\ c \in \mathcal{C} \tag{4.8e}$$

A solution to this model produces, for each commodity, a unique routing that is feasible when no failure occurs, but may result to be infeasible otherwise.

As for NFCC, we can define a tighter relaxation in which the worst-case failure scenario is considered. The resulting relaxation will be denoted as *Worst Failure With Channels* (WFWC).

4.3.2 Multiple Paths, Channel Conversion

All relaxations introduced above define, for each commodity, a unique path that can be infeasible under some scenarios. In this section, we introduce a relaxation, based on global re-routing, which does not produce a unique path; instead, given a commodity, a feasible (and possibly different) path is defined for each scenario. A similar relaxation has been used in Agarwal and Venkateshan (2014) for survivable network design for deriving lower bounds on the optimal solution value.

Besides the uniqueness of the paths, this relaxation drops the wavelength continuity constraint. In other words, the relaxation is obtained by

- allowing each commodity to have a different path for each scenario; and
- dropping the wavelength continuity constraint.

Thus, the relaxation reserves capacities on edges of the network that allow, under any failure scenario, to define a path for each commodity. However, identified paths change from scenario to scenario; therefore, there is no distinction between working and recovery paths, and only one path is identified per commodity and scenario.

Besides the w_e variables, the formulation uses binary variables x_e^{kf} indicating whether the path for commodity k uses edge e under scenario f (i.e., when f fails), and denote by $\boldsymbol{x}^{kf}$ the set of variables associated with a given commodity k and scenario f.

$$
\begin{aligned}
(\texttt{MPCC}) \qquad & \min \sum_{e \in E} \ell_e \, w_e && && (4.9a)\\
& x^{kf} \in \mathcal{P}^k && k \in \mathcal{K}, f \in E && (4.9b)\\
& w_e \geq \sum_{k \in \mathcal{K}} x_e^{kf} && e \in E, f \in E && (4.9c)\\
& x_f^{kf} = 0 && k \in \mathcal{K}, f \in E && (4.9d)\\
& x_e^{kf} \in \{0, 1\} && k \in \mathcal{K}, e \in E, f \in E && (4.9e)\\
& 0 \leq w_e \leq u && e \in E && (4.9f)
\end{aligned}
$$

This formulation is similar to (4.5), and therefore constraints (4.9b) could be expressed by means of the rooted flow formulation. The additional apex f refers to a specific scenario, identified by the failing edge. Constraints (4.9c) extend their counterparts and set the capacity to be reserved on each edge e as the maximum used for that edge under any scenario. Finally, constraints (4.9d) forbid the use of a failed edge.

Observe that relaxation `WFCC` is a relaxation of `MPCC`, as the former allows different capacity values to be defined for each failure scenario, while in the latter, capacities are unique for all scenarios.

Benders' Decomposition for `MPCC`

Formulation (4.9a)–(4.9f) involves a large number of variables and constraints, which may prevent its solution for large-size instances. In some cases, even solving the linear relaxation of the model may be computationally challenging for memory reasons.

Thus, we designed and implemented an alternative solution approach based on decomposition, allowing to project out the x variables from the formulation. Given a guess $\bar{w}_e$ on the capacity reserved on each edge e, which determines the current solution value, the formulation naturally decomposes into subproblems, each associated with a failure scenario identified by the failing edge f. This suggests to solve its continuous relaxation by means of a Benders' decomposition approach (Benders (1962)) in which

- a master problem determines the value of the w_e variables; and
- there is an LP subproblem for each edge failure scenario f, in which we verify if all commodities can be routed given the current value $\overline{w}_e$ of the master variables and the actual failure.

The sub-problem associated with a given scenario $\bar{f}$ is defined as follows

$$(SUB_{\bar{f}}) \qquad \min 0 \tag{4.10a}$$

$$\boldsymbol{x}^{k\bar{f}} \in \mathcal{P}^k \qquad k \in \mathcal{K} \tag{4.10b}$$

$$\sum_{k\in\mathcal{K}} x_e^{k\bar{f}} \leq \overline{w}_e \qquad e \in E \tag{4.10c}$$

$$x_{\bar{f}}^{k\bar{f}} = 0 \qquad k \in \mathcal{K} \tag{4.10d}$$

$$x_e^{k\bar{f}} \geq 0 \qquad k \in \mathcal{K},\ e \in E \tag{4.10e}$$

Being $\bar{w}_e$ input parameters, this sub-problem includes the subset of x variables associated with scenario $\bar{f}$ only. The upper bound on the value of the variables can be omitted as it is implicitly implied by constraints (4.10b), which are expressed by means of the rooted flow formulation (4.7).

If this sub-problem turns out to be infeasible, an extreme ray $(\bar{\mu}_m, \bar{\pi}_{mv}, \bar{\alpha}_e)$ of its dual is available as certificate of infeasibility. Here, $\bar{\mu}_m, \bar{\pi}_{mv}$ and $\bar{\alpha}_e$ are the dual variables associated with constraints (4.7a), (4.7b) and (4.10c), respectively. In this case, a feasibility cut can be derived

$$\sum_{m\in\mathcal{M}} q^m\,\bar{\mu}_m + \sum_{m\in\mathcal{M}}\sum_{v\in V\setminus\{s^m\}} g_v^m\,\bar{\pi}_{mv} - \sum_{e\in E}\bar{\alpha}_e\,w_e \leq 0. \tag{4.11}$$

By defining $\beta = \sum_{m\in\mathcal{M}} q^m\,\bar{\mu}_m + \sum_{m\in\mathcal{M}}\sum_{v\in V\setminus\{s^m\}} g_v^m\,\bar{\pi}_{mv}$, the generic inequality $\sum_{e\in E}\alpha_e\,w_e \geq \beta$ is added to the master, which is then re-optimized in an iterative fashion. Accordingly, at a generic iteration, the master problem reads as

$$(MAST) \qquad \min \sum_{e\in E} \ell_e\,w_e \tag{4.12a}$$

$$\sum_{e\in E} \alpha_{ie}\,w_e \geq \beta_i \qquad i \in \mathcal{I} \tag{4.12b}$$

$$0 \leq w_e \leq u_e \qquad e \in E, \tag{4.12c}$$

where $\mathcal{I}$ is the set of valid inequalities added so far. Iterations stop when all sub-problems are feasible.

Similar decomposition approaches have been presented in Bienstock and Mattia (2007) for solving capacity allocation problems in power grids subject to power line failures, and by Mattia (2012) for designing a multi-layer network with survivability requirements. Finally, Agarwal and Venkateshan (2014) consider a similar relaxation for survivable network design and solve it by Benders' decomposition. Note that an equivalent approach for solving the subproblem associated with a failure scenario can be obtained by exploiting the use of metric inequalities (see, e.g., Avella et al. (2007)).

Efficient Implementation of the Algorithm

A naive implementation of the Benders' decomposition scheme often shows a very slow convergence rate, resulting in an inefficient solution algorithm. We extensively tested the most effective techniques proposed in the literature for improving the performance of Benders' decomposition. We next describe the three tools that, according to our computational experience, turned out to have a significant computational impact in our application.

- Definition of the master problem: in our implementation, in addition to (4.12), the master problem includes variables and constraints that define one specific scenario. In particular, we embed in the master the scenario corresponding to the worst-case scenario according to `WFCC`.
- Scenario separation: at each iteration, sub-problems deemed feasible at the previous iteration are not re-evaluated as long as at least one scenario results infeasible. If no infeasibility is detected, all scenarios are re-evaluated, thus ensuring the correctness of the scheme. In our algorithm, we allow the generation and addition of many cuts at each iteration; this policy turned out to be computationally more efficient than re-optimizing the master as soon as an infeasible scenario is detected.
- Stabilization: at each iteration, sub-problems are evaluated with respect to a combination of the master solution and of a feasible solution, according to an in-out strategy (Fischetti and Salvagnin 2010). In detail, let $\overline{w}$ be the current master solution and $\hat{w}$ be a feasible solution of the master problem, obtained by solving each scenario independently and by considering, for each edge, a capacity equal to the maximum flow among all scenarios. For a given $\alpha \in [0, 1]$, we perform separation with respect to a convex combination of the two solutions, namely $\alpha\hat{w} + (1 - \alpha)\overline{w}$. In this way, we either define a new feasible solution improving over $\hat{w}$ or generate tighter cuts. Parameter α is dynamically adjusted based on the fraction of infeasible sub-problems at the last iteration: when less than 10% of the sub-problems are infeasible, the coefficient is halved to move closer to the master solution; conversely, when more than 50% of the sub-problems are infeasible, we double the coefficient (capped at 0.5), moving the convex combination closer toward the feasible solution. This dynamic approach balances cut strength and computational efficiency: early iterations benefit from aggressive cut tightening, whereas later iterations converge faster by directly focusing on the master solution. When the scheme is close to convergence, the number of infeasible sub-problems naturally decreases, resulting in smaller and smaller values for α, eventually close to zero at the final iterations.

4.4 Heuristic Algorithm

In this section, we introduce a metaheuristic algorithm, denoted as H, designed to compute high-quality feasible solutions for our problem. The proposed heuristic defines working and recovery paths for the commodities; during its execution, it may happen that these paths are defined for some commodities only (*served commodities*), while they are missing for other commodities (*unserved commodities*). Such an incomplete solution is called a *partial solution*. Algorithm H is based on the iterated local search paradigm, proposed by Lourenço et al. (2003), and relies on three main components, namely:

- a *constructive* procedure, used to build an initial solution and to allocate the unserved commodities of a given partial solution;
- an *improvement* procedure, used to possibly improve the current solution;
- a *shaking* procedure, used to perturb the current solution whenever it was not improved after a certain number of iterations.

In the following, we give a description of H and of its various components. The parameters used by the heuristic, denoted by the letter A plus a superscript, are described in Table 4.1.

An overview of H is given in Algorithm 3. The heuristic starts by invoking the constructive procedure, described in Section 4.4.1 on an empty solution (i.e., in which no commodity is served). After that, a randomized improvement procedure, described in Section 4.4.2, is applied to the current solution with the aim of (i) allocating the unserved commodities (if any) and (ii) reducing the resource utilization without decreasing the number of served commodities. If there is no improvement for A^{η} consecutive iterations, the shaking procedure, described in Section 4.4.3, is applied to perturb the best solution

Parameter	Meaning	Value
A^{time}	Time limit	1 hour
A^{η}	Max number of consecutive non-improving iterations before shaking	5000
A^{c}	Percentage of working channels evaluated during the improvement phase	20%
A^{ω}	Initial target damage for the shaking phase	1
A^{r}	Commodities tournament size	U[1,5]
A^{s}	Number of commodities evaluated during the shaking phase	10
A^{P}	Number of shortest paths evaluated during the shaking phase	10

Table 4.1: Parameters of algorithm H.

found. The algorithm alternates between the improvement and the shaking procedures until a prescribed time limit is reached, and the best solution found is returned.

In order to encourage diversification and escape from local minima, we introduce a controlled perturbation by updating the current and the incumbent solution also when an equivalent solution is found, i.e., when both solutions serve the same number of commodities and use the same amount of resources.

Algorithm 3: H

```
1 Compute the A^P shortest paths on G for each commodity k ∈ K
2 Apply the constructive heuristic and define the current solution
3 Initialize ω
4 while time < A^time do
5 |   Apply the improvement procedure to the current solution and possibly update
  |     both the current and the incumbent solution
6 |   If the incumbent is not improved in the last A^η iterations, then update ω and
  |     execute the shaking procedure to the incumbent, defining a new current
  |     solution
7 end
8 return the current solution
```

4.4.1 Constructive Procedure

The constructive heuristic takes as input a, possibly empty, partial solution and tries to allocate the unserved commodities. Commodities are considered one at a time, in a random order. For each commodity, we try to find an edge-disjoint pair of working and recovery paths. If such paths are found, an improved solution is defined in which the commodity is served.

Given a solution *Sol* and an unserved commodity k, we compute the disjoint paths in two steps: first, we compute a working path W_k, and then we determine a recovery path R_k that is edge-disjoint with respect to W_k. When computing the working path, we do not exploit sharing with already allocated commodities. Indeed, evaluating sharing would prevent the algorithm to know the precise cost of a candidate edge to be added to the working path before the working path is complete, thus, standard shortest-path algorithms could not be used. On the contrary, when computing the recovery path, all information about sharing is available as the working path is fixed. In any case, though recovery paths could exploit sharing with existing working paths in some cases (see Figure 4.2), we do not exploit this possibility. This is motivated by preliminary computational experiments showing that this opportunity may limit choices for the

following commodities, thus worsening the overall solution quality.

In order to compute the working path, the procedure randomly selects a subset Ω_w of candidate channels. For each candidate channel, the procedure computes the minimum-cost path for the current commodity, and selects the channel resulting with the minimum cost (breaking ties arbitrarily). The minimum-cost path on a channel is computed as a shortest path on graph $\mathcal{G}$ in which the lengths of the edges depend on the current solution. In particular, the length of each edge $e \in E$ is given by

$$g_e^w(Sol, c) = \begin{cases} \ell_e & \text{if } W_{c,e}^{Sol} \cup R_{c,e}^{Sol} = \emptyset \\ +\infty & \text{otherwise,} \end{cases} \tag{4.13}$$

where $W_{c,e}^{Sol}$ and $R_{c,e}^{Sol}$ denote the set of commodities served by solution Sol that use channel c of edge e as working and recovery path, respectively. In other words, each edge e has its original cost if available, and cannot be shared if already used by some other commodity.

Given the current solution, the minimum cost of a recovery path on channel c for a commodity k is computed as a shortest-path on graph $\mathcal{G}$ in which the length of each edge $e \in E$ is defined as follows

$$g_e^r(Sol, c, W_k) = \begin{cases} \ell_e & \text{if } W_{c,e}^{Sol} \cup R_{c,e}^{Sol} = \emptyset \\ +\infty & \text{if } \exists\, q \in W_{c,e}^{Sol} \\ +\infty & \text{if } \exists\, q \in R_{c,e}^{Sol} \,:\, W_q \cap W_k \neq \emptyset \\ 0 & \text{otherwise.} \end{cases} \tag{4.14}$$

These costs distinguish between cases in which an edge (i) is not used by the current solution; (ii) is used and cannot be shared; and (iii) is used and can be shared with no extra cost. As already mentioned, in case (ii) we do not exploit the possibility to share channels on edges that are used in the working path of a different commodity, though this would be possible in principle. For computing the recovery path for a commodity, the procedure considers all channels and selects the one associated with minimum cost.

We computationally observed that a full optimization in the selection of the working path might be too restrictive for subsequent decisions. Thus, in order to introduce randomness in the selection of the working path, we deliberately consider a random subset of channels only. Conversely, when designing the recovery path, we consider all channels in order to exploit all sharing opportunities. In both cases, all shortest paths are computed by using Dijkstra's algorithm (Dijkstra (1959)).

4.4.2 Improvement Procedure

The *improvement* procedure implements a randomized version of the iterated local search paradigm proposed by Lourenço et al. (2003). The main idea of the procedure consists of removing a single commodity from the current solution, and by re-executing the constructive procedure on the resulting partial solution. More precisely, the procedure selects the commodity to be removed among a randomly selected set $\mathcal{K}'$ of A^r candidate commodities. For each candidate commodity k', the procedure computes a measure of inefficiency $\Delta_{k'}$, defined as the difference between the current contribution of the commodity to the solution cost and the cost of the shortest path for the commodity in the original graph. We evaluate the contribution of a commodity to the solution cost as the cost reduction achieved when removing that commodity from the solution. The selected commodity is then removed from the solution, and the constructive procedure is executed on the current partial solution, to possibly allocate the commodity as well as other unserved commodities. In case the resulting solution is not worse than the current one, a replacement is implemented.

According to our preliminary computational experience, removing more than one commodity at a time during the improvement phase is not better than removing just one, as in the proposed algorithm.

4.4.3 Shaking Procedure

When the improvement procedure turns out to be ineffective, i.e., when it fails in improving the incumbent for A^η consecutive iterations, the algorithm needs to perturb the solution more intensively, so as to escape from local optima. To this end, we introduce a shaking procedure, inspired to the ruin phase of ruin-and-recreate algorithms (see Schrimpf et al. (2000)), which allows to remove more than one commodity at a time. The intensity of the shaking procedure is controlled by a parameter $\omega \in [1, |\mathcal{K}|]$, which represents the target number of commodities to be removed from the solution. If ω is too low, the algorithm may be unable to escape from the local optimum, whereas a large value for ω can spoil the quality of the solution. Our policy for updating the value of ω is to decrease it by one unit if the local optimum is worse than the best solution found, and to increase it by one unit, otherwise. We apply this perturbation to the incumbent solution, and use the resulting solution as new current solution at the end of the process.

As shaking is aimed at diversifying the search, the procedure is designed to possibly affect any commodity served in the solution. The procedure starts by choosing a commodity k among a randomly selected set of candidates, as in the improvement procedure; here, we also allow unserved commodities to be selected, in which case the

cost contribution is equal to $+\infty$. Then, it considers the A^P-shortest paths on the original graph for the selected commodity, and for each such path, it evaluates the possibility of assigning each channel to the working path, possibly removing other commodities from the solution. Finally, the recovery path for the selected commodity is defined.

The A^P (loopless) shortest paths for the selected commodity are computed at the beginning of the algorithm by means of the Yen (1971) algorithm. In this phase, parallel edges are ignored, i.e., they are considered as a unique edge. For designing the working path of commodity k, we consider each of the A^P shortest paths available and each channel: for each path and channel, we compute the associated *damage*, counting the number of commodities that are served in the current solution and that would conflict with the choice, i.e., using the same channel on some edges of the path. To exploit the possible presence of parallel edges in $\mathcal{G}$, the procedure randomly selects one of the edges connecting two consecutive nodes in the given path. The procedure selects the path and channel for which damage is closest to the target value ω. Once the working path (and its channel) is selected, the conflicting commodities are removed from the solution and the recovery path for commodity k is computed by solving a shortest path problem on graph $\mathcal{G}$ with edge costs defined as in (4.14). Finally, the constructive procedure is applied to the current partial solution to possibly increase the number of served commodities.

The core procedure is executed for a given number A^s of iterations; at the end, the best solution found is returned, and the corresponding damage is computed and a new target damage value is defined as $\omega = \omega - \omega_{best}$. The procedure is repeated if ω is positive, and is prematurely terminated if ω_{best} is 0.

4.5 Computational Experiments

In this section, we computationally test different approaches for the problem. We first evaluate the possibility to directly solve the ILP formulation by means of a general-purpose solver. The negative outcome of these experiments motivates the use of the lower bounding and heuristic procedures introduced in the previous sections. Thus, we first assess the tightness and computational effort of each relaxation, and then evaluate the performance of the proposed heuristic algorithm, also in comparison with a baseline constructive approach. Finally, we show that the combination of lower and upper bounding procedures allows to certify the quality of the produced solutions in terms of optimality gap.

All algorithms have been coded in C++20 and compiled with GCC 11.4.0. Experiments were run on a machine with AMD Ryzen 5 PRO 4650GE equipped with 16 GB of RAM

under Ubuntu 22.04.4 LTS operating system. All LPs and ILPs have been solved by using IBM ILOG CPLEX Optimization Studio 20.1.0.

4.5.1 Instances

We tested our algorithms on two benchmarks of instances.

The first set includes eight instances taken from the SNDlib (https://sndlib.put.poznan.pl/home.action), a library of data for Survivable fixed telecommunication Network Design aimed at providing data about realistic telecommunication networks. In particular, we selected 8 instances that were addressed in Agarwal and Venkateshan (2014) for a related resource allocation problem in which each commodity has a certain demand, though channels are not explicitly modeled. In order to define instances for our problem, we maintained the network structure and defined demands and capacities as follows. First, we grouped all commodities sharing the same source and destination, thus obtaining distinct commodities with non-unitary requests. Then, we normalized all requests with respect to the smallest one (rounding up the value, when needed). Finally, we solved a simplified problem to compute the smallest capacity to be installed on each edge so that two edge-disjoint paths can be defined for each commodity with no sharing and no explicit channel allocation. We remark here that the resulting capacity does not guarantee the existence of a feasible solution for the original problem, as channel allocation is not considered. At the same time, ignoring the sharing opportunity allows to define capacities that are not too tight in practice.

The second set of instances has been provided by Huawei and includes twenty realistic instances, referred to as 1–20 in the tables, representing prototypical configurations of existing networks.

Table 4.2 reports the main characteristics of the instances, i.e., the name, number of vertices, edges, commodities, the number of aggregated macro-commodities, and the value of the links' capacity. The considered benchmark shows a considerable variability in terms of size of the graph, density, commodity number and capacity.

In our first set of experiments, we tested the exact formulation (4.1). For each instance, the solver was run 5 times with 5 different random seeds to mitigate the effects of erraticity. For each run, the time limit was set to 5 hours and the solver was allowed to use 6 threads. All experiments prematurely terminated due to memory limits, but for two small instances from the SNDlib, though no feasible solution was computed in these cases either. Similar results were obtained by replacing constraints (4.1e) and (4.1f), that express a logical condition, by their quadratic counterpart and exploiting the capability of the solver to approach nonlinear quadratic models. Based on these negative outcomes, we conclude that directly approaching the exact formulation is not

Instance	$\|V\|$	$\|E\|$	$\|K\|$	$\|\overline{K}\|$	u
atlanta	15	22	1609	14	652
france	25	45	2087	24	443
germany50	50	88	1226	39	164
newyork	16	49	476	15	45
norway	27	51	1782	26	366
pdh	11	34	48	7	5
pioro40	40	89	1150	39	225
polska	12	18	100	11	37
1	20	55	321	13	80
2	114	295	550	31	80
3	83	342	2785	36	80
4	32	108	465	10	80
5	748	1283	1490	277	80
6	92	196	907	19	80
7	67	264	1001	29	80
8	80	330	1863	37	80
9	37	150	600	19	80
10	46	184	700	23	80
11	48	198	800	24	80
12	52	219	900	26	80
13	103	275	1913	40	80
14	57	142	820	21	80
15	178	363	1538	94	80
16	172	352	621	33	80
17	159	376	1954	70	80
18	55	116	711	27	80
19	377	546	827	194	80
20	377	546	2158	196	80

Table 4.2: Instance features.

a viable way for solving the problem. Unfortunately, also solving the linear relaxation of the formulation turned out to be very challenging in terms of memory requirements, allowing, in this case as well, to obtain the optimal value of the linear relaxation for the two smallest instances only. In addition, in both cases, the resulting dual bound was very weak. Therefore, in order to obtain feasible solutions for the problem and assess their quality, in the next two sections we examine the computational performances of the proposed lower and upper bounds.

4.5.2 Results on Lower Bounds

In this section, we present our computational results evaluating the quality of the lower bound obtained by solving the relaxations introduced in the previous sections. Table 4.3 reports the outcome of our experiments for relaxations based on a unique working and no recovery path (see Section 4.3.1). For NFCC, which is the baseline of our comparison, we report the value of the dual bound, whereas for all the remaining relaxations, we give the percentage improvement of the resulting dual bound with respect to the baseline. By denoting with LB and $\overline{L}$ the baseline and the value of the other lower bound under consideration, respectively, the percentage improvement is defined as $Imp_{LB} = 100\,(\overline{L} - LB)/LB$. In addition, for all methods we report the corresponding computing time. For all formulations (NFCC, WFCC, NFWC, and WFWC) we solved the linear relaxation of the model and used the dual simplex algorithm, which computationally turned out to be the most effective solution algorithm. When solving WFCC and WFWC, the linear programs to be solved at each iteration are always optimized from scratch. For both NFCC and WFCC we report two computing times, the first column (T_1) relative to a classical implementation of constraints (4.5b), and the second column (T_2) relative to a rooted flow formulation implementing constraints (4.7a) and (4.7b). Finally, in the table we mark with a "-" the cases where a formulation cannot be solved, either for time or memory limits.

Table 4.3 shows that relaxation NFCC can be solved for all instances in short computing time (column T_1), reaching 22.56 seconds for instance 5, which is the most challenging one. By considering the rooted flow formulation, the computing times are reduced by at least one order of magnitude. When considering channels (NFWC), the rooted flow formulation cannot be exploited, and due to memory limitation we are able to solve 12 out of 28 instances only. Concerning the value of the obtained lower bound, by explicitly modeling channels, we do not obtain any improvement. Solving WFCC requires a much larger computational effort than solving NFCC as this asks to solve an LP for each edge which may fail. Computing times reflect this larger effort, and can almost reach 4 hours for the most challenging instance. However, for WFCC the rooted flow formulation is extremely efficient, and computing times are reduced by three orders of

Instance	NFCC			NFWC		WFCC			WFWC	
	LB	T_1	T_2	Imp_{LB}	T	Imp_{LB}	T_1	T_2	Imp_{LB}	T
atlanta	3260	0.19	0.04	-	-	19.05	2.46	0.03	-	-
france	4944	0.59	0.02	-	-	5.24	16.79	0.04	-	-
germany50	3475	0.92	0.06	-	-	4.92	58.48	0.12	-	-
newyork	776	0.12	0.03	0.00	6.72	3.09	3.25	0.01	3.09	198.97
norway	5515	0.60	0.05	-	-	4.04	19.87	0.05	-	-
pdh	48	0.03	0.03	0.00	0.05	8.33	0.11	0.03	8.33	0.41
pioro40	3799	0.92	0.05	-	-	4.00	60.18	0.10	-	-
polska	213	0.03	0.02	0.00	0.34	11.27	0.11	0.03	11.27	3.83
1	882	0.09	0.03	0.00	8.72	7.48	1.94	0.03	7.48	234.12
2	3196	1.04	0.07	0.00	114.36	1.85	153.93	0.21	1.85	17310.45
3	11588	9.46	0.10	-	-	1.86	1183.68	0.50	-	-
4	1292	0.25	0.04	0.00	28.29	5.34	11.11	0.03	5.34	1342.59
5	4554	22.56	2.51	-	-	3.34	13642.27	34.77	-	-
6	2968	1.16	0.05	0.00	148.64	4.82	120.11	0.11	4.82	15672.48
7	4578	1.66	0.08	-	-	2.73	214.82	0.14	-	-
8	8502	5.40	0.12	-	-	2.07	800.56	0.43	-	-
9	1639	0.44	0.04	0.00	53.52	2.07	27.50	0.05	2.07	3429.22
10	2287	0.66	0.04	0.00	79.94	1.44	60.08	0.07	1.44	6800.21
11	2629	0.82	0.05	0.00	103.22	1.29	82.01	0.08	1.29	8691.93
12	3205	1.11	0.06	-	-	1.06	122.06	0.10	-	-
13	6269	4.23	0.09	-	-	2.23	384.66	0.17	-	-
14	2273	0.58	0.04	0.00	75.90	10.56	26.18	0.05	10.56	3331.13
15	5353	5.29	0.25	-	-	1.72	1163.04	1.44	-	-
16	3982	1.87	0.10	-	-	2.84	234.72	0.33	-	-
17	3710	6.60	0.17	-	-	3.07	924.15	0.60	-	-
18	1414	0.44	0.04	0.00	53.88	11.88	28.91	0.07	11.88	3314.03
19	3505	4.57	0.67	-	-	8.30	1639.30	7.19	-	-
20	5072	13.92	0.67	-	-	4.46	4544.08	7.35	-	-

Table 4.3: Results of the relaxations based on unique working and no recovery path.

magnitude for the most challenging instances, with the largest computing time being 34.77 seconds. The bound obtained is improved by 4.64% on average with respect to the case with no failure. Finally, by considering channels on top of WFCC, we obtain relaxation WFWC, for which we are only able to solve the same 12 instances we could solve for NFWC, with no improvement on the bound value with respect to WFCC. Summarizing, explicitly modeling channels results in formulations of very large size, which are often intractable due to memory limitations. Moreover, in all instances where NFWC and WFWC can be solved, they provide the same lower bound as formulations that allow channel conversion (NFCC and WFCC, respectively). Therefore, we disregard explicit channels when developing more sophisticated relaxations.

For what concerns formulation MPCC, we consider its linear relaxation and report both the results obtained by directly solving it (column "No-decomp.") and those obtained from the application of the Benders' decomposition approach (column "Benders") introduced in Section 4.3.2. According to our preliminary experiments, when directly tackling the formulation, the best option is to use the barrier algorithm. As we do not care about having a solution corresponding to a vertex of our formulation, we disable the crossover algorithm in order to reduce the computational effort. When solving the problem via Benders' decomposition, one can exploit multi-thread CPUs by (i) solving the master problem with multiple threads, and (ii) allocating a thread to each subproblem to be solved in parallel. Therefore, we test our implementation of Benders' decomposition with 1 and 6 threads, respectively. In order to have a fair comparison, the same settings are considered when solving MPCC without decomposition.

Table 4.4 reports the value of the lower bound associated with MPCC, its percentage improvement over the baseline given by NFCC and, for each approach and setting (number of threads), the associated computing time.

The linear relaxation of formulation MPCC can be solved directly by means of the barrier algorithm for all but 5 instances, where the solving process is terminated due to exceeded memory limits. For solved instances, the computing times range from fractions of seconds up-to more than 9 hours of computing time (instance 13). The computing time is almost halved when the process is executed on 6 cores, but instance 13 cannot be solved anymore. When Benders' decomposition is used, we never exceed the memory limits and are always able to compute a valid lower bound, although, for some instances, the computational effort can be significantly larger than without decomposition. The algorithm benefits from parallelization and the resulting computing time is around one third of that in the sequential case, on average. Furthermore, Table 4.4 shows that the lower bound produced by MPCC is considerably larger than the baseline, the percentage improvement being equal to 34.27% on average.

Instance	*value*	Imp_{LB}	No-decomp.		Benders	
			T_{1th}	T_{6th}	T_{1th}	T_{6th}
atlanta	5908	81.23	0.13	0.12	0.16	0.12
france	8145	64.75	1.92	0.96	1.19	0.43
germany50	4480	28.92	145.75	43.80	45.96	12.74
newyork	906	16.75	4.03	1.44	1.24	0.48
norway	7233	31.15	5.80	2.29	3.05	0.99
pdh	64	33.33	0.47	0.23	0.17	0.13
pioro40	4809	26.59	14.58	9.23	72.29	14.40
polska	307	44.13	0.06	0.06	0.10	0.06
1	1188	34.69	1.37	0.65	0.83	0.34
2	4095	28.13	146.04	73.10	471.24	121.71
3	13008	12.25	213.46	110.40	2340.73	485.43
4	1612	24.77	1.33	0.99	2.67	0.85
5	6803	49.39	-	-	>500k	132626.50
6	4033	35.88	36.06	22.16	64.95	13.84
7	5395	17.85	46.81	29.47	438.23	101.22
8	9663	13.66	181.05	92.68	1888.29	659.69
9	1918	17.02	8.36	5.30	32.19	9.56
10	2645	15.65	19.40	12.26	87.87	22.06
11	3013	14.61	21.68	13.41	127.72	29.88
12	3619	12.92	30.44	18.03	254.01	61.09
13	8040	28.25	33487.62	-	457.91	102.73
14	3430	50.90	8.63	6.16	6.16	2.58
15	6681	24.81	-	-	15800.40	2514.69
16	5328	33.80	473.73	223.67	977.95	198.64
17	5225	40.84	-	-	1616.82	519.66
18	2286	61.67	49.33	18.24	7.31	2.73
19	5792	65.25	-	-	7536.74	2239.30
20	7629	50.41	-	-	4396.79	1325.84

Table 4.4: Results of the continuous relaxation of `MPCC` and computing times with 1 thread and 6 threads.

4.5.3 Results on Upper Bounds

Table 4.5 presents the results of our heuristic algorithm. The table gives, for each instance, the optimality gap of the initial solution. For some instances, the initial solution is unable to serve all the requests; in these cases, we report in brackets the percentage of unserved commodities. We do not report the associated computing time, which was always below 1 second. In addition, we give the percentage improvement and the corresponding optimality gap of the best solution found after 1 hour and 5 hours, respectively. All optimality gaps are computed with respect to the same reference value, given by the best available lower bound, namely `MPCC` (reported in Table 4.4). By denoting with LB the reference lower bound and with UB the value of an upper bound, the optimality gap is computed as $opt_{gap} = 100\,(UB - LB)/LB$. In addition, by denoting with UB and $\overline{U}$ the value of the initial solution and the value of an improved solution, respectively, the percentage improvement is defined as $Imp_{UB} = 100\,(UB - \overline{U})/UB$.

The initial constructive heuristic is able to find a feasible solution in 19 instances out of 28, while in the remaining 9 cases, a certain number of requests remain unserved. Even disregarding unserved requests, its performance is not satisfactory, the associated average optimality gap being equal to 17.83%. By applying the proposed iterated local search heuristic, the quality of the solution is considerably improved both in terms of feasibility, which is achieved for all instances, and of solution value. After 1 hour, the average cost reduction is around 10% and the corresponding average optimality gap is reduced to 8.59%. When a longer time limit is used (5 hours), further non-negligible improvements are obtained: the average cost reduction with respect to the initial solution is 10.86% and the average optimality gap is 7.86%. The optimality gap is always below 10% but for 6 challenging and very large instances.

We now report an ablative study on the impact of the acceptance strategy mentioned at the end of Section 4.4 and of the shaking procedure described in Section 4.4.3. Three variants of `H` have been evaluated, obtained by removing updating in case of equivalent solutions (`NEQ`), by disabling the shaking phase (`NS`), and by disabling both features (`NSEQ`). Table 4.6 reports the percentage cost increase of the best solution found by each variant with respect to the solution cost when both features are activated. All algorithms are executed with a time limit equal to 1 hour and all of them are able to compute a feasible solution for all instances.

The table shows that all variants produce solutions that are worse than those computed by `H` for all instances but one (pdh), for which `NS` and `NEQ` find solutions of the same cost. When both features are deactivated (`NSEQ`) the average percentage cost increase of the solution is equal to 6.88%. by disabling one feature at a time, worsening is less consistent and reduces to 1.68 and to 1.28 for `NEQ` and `NS`, respectively. This shows that

Instance	Initial solution		H (after 1h)		H (after 5h)	
	opt_{gap}		opt_{gap}	Imp_{UB}	opt_{gap}	Imp_{UB}
atlanta	8.52	(0.44)	1.70	6.94	1.32	7.29
france	9.38	(0.14)	3.48	6.11	2.73	6.83
germany50	44.50	(0.24)	11.97	36.95	11.32	37.41
newyork	14.93		6.12	9.39	5.73	9.77
norway	14.65		6.32	8.89	5.97	9.23
pdh	23.81	(4.17)	3.03	21.43	3.03	21.43
pioro40	18.21		10.56	8.55	9.57	9.56
polska	17.92		5.54	13.10	5.25	13.37
1	14.35		4.73	10.09	4.35	10.45
2	19.72		10.61	10.19	10.16	10.64
3	13.44		9.25	4.61	8.53	5.36
4	14.80		5.84	9.51	5.51	9.83
5	23.59		15.16	9.93	13.60	11.56
6	19.79		9.74	11.14	9.00	11.85
7	14.61		7.96	7.22	7.52	7.66
8	12.66		9.05	3.97	8.21	4.84
9	14.95		7.48	8.07	6.76	8.78
10	14.74		8.39	6.93	7.87	7.45
11	15.05		8.50	7.16	7.66	8.01
12	14.24		8.75	6.02	8.05	6.73
13	21.66	(0.73)	12.55	10.42	11.66	11.32
14	19.86	(0.73)	9.21	11.73	8.46	12.45
15	18.57		10.47	9.06	9.63	9.90
16	19.62		9.19	11.48	8.34	12.30
17	19.27		10.92	9.36	9.80	10.49
18	10.88	(0.70)	5.42	5.77	4.67	6.51
19	22.78	(0.36)	13.71	10.52	12.65	11.60
20	22.68	(0.23)	14.75	9.30	12.80	11.33
Average	17.83		8.59	10.14	7.86	10.86

Table 4.5: Results of the upper bound.

Instance	NEQ	NS	NSEQ
atlanta	1.58	0.58	5.91
france	1.91	0.26	4.65
germany50	2.52	0.59	7.98
newyork	2.07	2.59	9.12
norway	2.16	0.61	7.78
pdh	0.00	0.00	13.64
pioro40	2.18	0.46	6.97
polska	1.85	4.00	11.08
1	1.76	1.68	8.82
2	1.46	0.74	5.94
3	0.73	0.34	3.16
4	0.82	0.93	5.90
5	1.82	1.45	4.56
6	2.10	1.30	7.79
7	1.77	0.53	5.46
8	0.63	0.08	2.80
9	1.88	1.59	6.32
10	1.04	1.39	5.68
11	1.25	1.12	5.31
12	0.38	0.58	4.26
13	4.10	0.40	8.55
14	3.12	2.44	9.82
15	2.31	0.63	6.42
16	1.98	1.50	6.97
17	1.24	2.59	6.72
18	1.12	2.52	8.03
19	1.18	2.26	6.05
20	2.20	2.64	6.92
Average	1.68	1.28	6.88

Table 4.6: Percentage cost increase when disabling diversification strategies.

the simple strategy of updating the incumbent in case of equivalent solutions has, on average, a greater impact than the shaking procedure on the solution quality. In any case, both features are useful for producing high-quality solutions and are partially complementary.

4.6 Conclusions

We considered the problem of assigning a given set of connection requests to a minimum cost set of wavelengths in an optical network. The assignment is designed in such a way to allow communication also in case of failure of an edge, and a shared protection policy is considered. For this problem, we introduced a mathematical formulation and different bounding procedures, including one based on a Benders' decomposition. In addition, we proposed a heuristic approach relying on the iterated local search paradigm and exploiting diversification strategies. Computational experiments were performed both on instances from the literature and on realistic instances from the industry, showing that our algorithms provide solutions whose optimality gap is around 8% on average.

Bibliography

Accorsi, L. and Vigo, D. (2024), 'Routing one million customers in a handful of minutes', *Computers and Operations Research* **164**, 106562.

Agarwal, Y. and Venkateshan, P. (2014), 'Survivable network design with shared-protection routing', *European Journal on Operational Research* **238**(3), 836–845.

Agatz, N., Bouman, P. and Schmidt, M. (2018), 'Optimization approaches for the traveling salesman problem with drone', *Transportation Science* **52**(4), 965–981.

Applegate, D. L., Bixby, R. E., Chvátal, V., Cook, W., Espinoza, D. G., Goycoolea, M. and Helsgaun, K. (2009), 'Certification of an optimal tsp tour through 85,900 cities', *Operations Research Letters* **37**(1), 11–15.

Arnold, F., Gendreau, M. and Sörensen, K. (2019), 'Efficiently solving very large-scale routing problems', *Computers and Operations Research* **107**, 32 – 42.
URL: *http://www.sciencedirect.com/science/article/pii/S0305054819300668*

Atamtürk, A. (2002), 'On capacitated network design cut-set polyhedra', *Mathematical Programming* **92**, 425–437.

Avella, P., Mattia, S. and Sassano, A. (2007), 'Metric inequalities and the network loading problem', *Discrete Optimization* **4**, 103–114.

Balakrishnan, A., Mirchandani, P. and Natarajan, H. P. (2009), 'Connectivity upgrade models for survivable network design', *Operations research* **57**(1), 170–186.

Banker, S. (2022), 'The race for last mile drones', https://www.forbes.com/sites/stevebanker/2022/08/16/the-race-for-last-mile-drones/. last accessed on June 6, 2025.

Benders, J. (1962), 'Partitioning procedures for solving mixed-variables programming problems', *Numerische Mathematik* **4**, 238–252.

Bienstock, D. and Mattia, S. (2007), 'Using mixed-integer programming to solve power grid blackout problems', *Discrete Optimization* **4**(1), 115–141.

Blufstein, M., Lera-Romero, G. and Soulignac, F. J. (2024), 'Decremental state-space relaxations for the basic traveling salesman problem with a drone', *INFORMS Journal on Computing* **36**(4), 1064–1083.

Boccia, M., Mancuso, A., Masone, A. and Sterle, C. (2023), 'A new MILP formulation for the flying sidekick traveling salesman problem', *Networks* **82**(3), 254–276.

Boccia, M., Masone, A., Sforza, A. and Sterle, C. (2021), 'A column-and-row generation approach

for the flying sidekick travelling salesman problem', *Transportation Research Part C: Emerging Technologies* **124**, 102913.

Bouman, P., Agatz, N. and Schmidt, M. (2018), 'Dynamic programming approaches for the traveling salesman problem with drone', *Networks* **72**(4), 528–542.

Buysse, J., Leenheer, M. D., Develder, C. and Dhoedt, B. (2009), Exploiting relocation to reduce network dimensions of resilient optical grids, *in* 'Proceedings of IEEE/VDE Workshop on Design of Reliable Communication Networks (DRCN)', pp. 100–106.

Campuzano, G., Lalla-Ruiz, E. and Mes, M. (2023), 'The drone-assisted variable speed asymmetric traveling salesman problem', *Computers and Industrial Engineering* **176**, 109003.

Cavaliere, F., Malaguti, E., Michelotto, F., Monaci, M., Pointurier, Y. and Vigo, D. (2025), 'Lower and upper bounds for wavelength allocation in optical networks', *Submitted to Networks* .

Chen, C., Demir, E., Hu, X. and Huang, H. (2025), 'Transforming last mile delivery with heterogeneous assistants: drones and delivery robots', *Journal of Heuristics* **31**, 8.

Chlamtac, I., Ganz, A. and Karmi, G. (1992), 'Lightpath communications: an approach to high bandwidth optical wan's', *IEEE Transactions on Communications* **40**(7), 1171–1182.

CVRPLIB (2025), 'Capacitated vehicle routing problem library'. visited on 2024-09-24.
URL: *http://vrp.galgos.inf.puc-rio.br/index.php/en/*

Dantzig, G. B. and Ramser, J. H. (1959), 'The truck dispatching problem', *Management Science* **6**(1), 80–91.

Dantzig, G., Fulkerson, R. and Johnson, S. (1954), 'Solution of a large-scale traveling-salesman problem', *Journal of the Operations Research Society of America* **2**(4), 393–410.

de Freitas, J. C. and Penna, P. H. V. (2020), 'A variable neighborhood search for flying sidekick traveling salesman problem', *International Transactions in Operational Research* **27**(1), 267–290.

Dell'Amico, M., Montemanni, R. and Novellani, S. (2021*a*), 'Algorithms based on branch and bound for the flying sidekick traveling salesman problem', *Omega* **104**, 102493.

Dell'Amico, M., Montemanni, R. and Novellani, S. (2021*b*), 'Modeling the flying sidekick traveling salesman problem with multiple drones', *Networks* **78**(3), 303–327.

Dell'Amico, M., Montemanni, R. and Novellani, S. (2022), 'Exact models for the flying sidekick traveling salesman problem', *International Transactions in Operational Research* **29**(3), 1360–1393.

Dijkstra, E. W. (1959), 'A note on two problems in connexion with graphs', *Numerische Mathematik* **1**, 269–271.

Doshi, B. T., Dravida, S., Harshavardhana, P., Hauser, O. and Wang, Y. (1999), 'Optical network design and restoration', *Bell Labs Technical Journal* **4**(1), 58–84.

El-Adle, A. M., Ghoniem, A. and Haouari, M. (2023), 'A variable neighborhood search for parcel delivery by vehicle with drone cycles', *Computers and Operations Research* **159**.

Es Yurek, E. and Ozmutlu, H. C. (2018), 'A decomposition-based iterative optimization algorithm for traveling salesman problem with drone', *Transportation Research Part C: Emerging Technologies* **91**, 249–262.

Fischetti, M. and Salvagnin, D. (2010), An in-out approach to disjunctive optimization, *in* A. Lodi, M. Milano and P. Toth, eds, 'Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems', Vol. 6140 of *Lecture Notes in Computer Science*, Springer Berlin Heidelberg, pp. 136–140.

Fogel, D. B. (1994), 'An introduction to simulated evolutionary optimization', *IEEE Transactions on Neural Networks* **5**(1), 3–14.

Fumagalli, A., Cerutti, I., Tacca, M., Masetti, F., Jagannathan, R. and Alagar, S. (1999), Survivable networks based on optimal routing and wdm self-healing rings, *in* 'IEEE INFOCOM '99. Conference on Computer Communications. Proceedings. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. The Future is Now (Cat. No.99CH36320)', Vol. 2, pp. 726–733 vol.2.

Grötschel, M., Monma, C. L. and Stoer, M. (1992), 'Facets for polyhedra arising in the design of communication networks with low-connectivity constraints', *SIAM Journal on Optimization* **2**(3), 474–504.

Gudapati, N. V., Malaguti, E., Monaci, M. and Paronuzzi, P. (2025), 'A computational study on integer programming formulations for hop-constrained survivable network design', *Discrete Applied Mathematics* **362**, 71–81.

Guth, A. H. (2001), 'Time since the beginning', *Astrophysical Ages and Time Scales* **245**, 3–17.

Ha, Q. M., Deville, Y., Pham, Q. D. and Hà, M. H. (2018), 'On the min-cost traveling salesman problem with drone', *Transportation Research Part C: Emerging Technologies* **86**, 597–621.

Ha, Q. M., Deville, Y., Pham, Q. D. and Hà, M. H. (2020), 'A hybrid genetic algorithm for the traveling salesman problem with drone', *Journal of Heuristics* **26**(2), 219–247.

Jaumard, B., Meyer, C. and Thiongane, B. (2007), 'Comparison of ILP formulations for the RWA problem', *Optical Switching and Networking* **4**(3–4), 157–172.

Johnson, S. G. (2007), 'The NLopt nonlinear-optimization package', https://github.com/stevengj/nlopt.

Karp, R. M. (1975), 'On the computational complexity of combinatorial problems', *Networks* **5**(1), 45–68.

Kerivin, H. and Mahjoub, A. R. (2005), 'Design of survivable networks: A survey', *Networks* **46**(1), 1–21.

Krishnaswamy, R. and Sivarajan, K. (2001*a*), 'Algorithms for routing and wavelength assignment based on solutions of LP-relaxation', *IEEE Communications Letters* **5**(10), 435–437.

Krishnaswamy, R. and Sivarajan, K. (2001*b*), 'Design of logical topologies: A linear formulation for wavelength routed optical networks with no wavelength changers', *IEEE/ACM Transactions on Networking* **9**(2), 184–198.

Liang, Y.-J. and Luo, Z.-X. (2022), 'A survey of truck–drone routing problem: Literature review and research prospects', *Journal of the Operations Research Society of China* **10**(2), 343–377.

Liu, Z., Sengupta, R. and Kurzhanskiy, A. (2017), A power consumption model for multi-rotor small unmanned aircraft systems, *in* '2017 International Conference on Unmanned Aircraft Systems (ICUAS)', pp. 310–315.

Lloyd, S. (1982), 'Least squares quantization in pcm', *IEEE Transactions on Information Theory* **28**(2), 129–137.

Lourenço, H. R., Martin, O. C. and Stützle, T. (2003), *Iterated Local Search*, Springer US, Boston, MA, pp. 320–353.

Macrina, G., Di Puglia Pugliese, L., Guerriero, F. and Laporte, G. (2020), 'Drone-aided routing: A literature review', *Transportation Research Part C: Emerging Technologies* **120**, 102762.

Magnanti, T. L., Mirchandani, P. and Vachani, R. (1993), 'The convex hull of two core capacitated network design problems', *Mathematical Programming* **60**, 233–250.

Magnanti, T. L., Mirchandani, P. and Vachani, R. (1995), 'Modeling and solving the two-facility capacitated network loading problem', *Operations Research* **43**, 142–157.

Mattia, S. (2012), 'Solving survivable two-layer network design problems by metric inequalities', *Computational Optimization and Applications* **51**(2), 809–834.

Michelotto, F. and Roberti, R. (2025), 'A MILP and a genetic algorithm for the flying sidekick TSP with variable drone speeds', *Under review at Computers and Operations Research* .

Mouftah, H. T. and Ho, P.-H. (2003), *Optical Networks: Architecture and Survivability*, Springer US.

Murray, C. and Chu, A. (2015), 'The flying sidekick traveling salesman problem: Optimization of drone-assisted parcel delivery', *Transportation Research Part C: Emerging Technologies* **54**, 86–109.

Ozdaglar, A. E. and Bertsekas, D. P. (2003), 'Routing and wavelength assignment in optical networks', *IEEE/ACM Transactions on Networking* **11**(2), 259–272.

Poikonen, S., Golden, B. and Wasil, E. A. (2019), 'A branch-and-bound approach to the traveling salesman problem with a drone', *INFORMS Journal on Computing* **31**(2), 335–346.

Raj, R. and Murray, C. (2020), 'The multiple flying sidekicks traveling salesman problem with variable drone speeds', *Transportation Research Part C: Emerging Technologies* **120**, 102813.

Ramamurthy, S. and Mukherjee, B. (1999), Survivable WDM mesh networks—Part I: Protection, *in* 'Proceedings of IEEE INFOCOM', pp. 744–751.

Roberti, R. and Ruthmair, M. (2021), 'Exact methods for the traveling salesman problem with drone', *Transportation Science* **55**(2), 315–335.

Sahasrabuddhe, L. and Mukherjee, B. (2002), 'Fault tolerance in IP-over-WDM networking: WDM protection versus IP restoration', *IEEE Journal on Selected Areas in Communications* **20**(1), 21–33.

Santini, A., Schneider, M., Vidal, T. and Vigo, D. (2023), 'Decomposition strategies for vehicle routing heuristics', *INFORMS Journal on Computing* **35**(3), 543 –559.
URL: *https://pubsonline.informs.org/doi/10.1287/ijoc.2023.1288*

Schermer, D., Moeini, M. and Wendt, O. (2020), 'A branch-and-cut approach and alternative formulations for the traveling salesman problem with drone', *Networks* **76**(2), 164–186.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H. and Dueck, G. (2000), 'Record breaking optimization results using the ruin and recreate principle', *Journal of Computational Physics* **159**(2), 139–171.

Somani, A. K. (2006), *Survivability and Traffic Grooming in WDM Optical Networks*, Cambridge University Press.

Stolaroff, J. K., Samaras, C., O'Neill, E. R., Lubers, A., Mitchell, A. S. and Ceperley, D. (2018), 'Energy use and life cycle greenhouse gas emissions of drones for commercial package delivery', *Nature Communications* **9**(409).

Tamke, F. and Buscher, U. (2023), 'The vehicle routing problem with drones and drone speed selection', *Computers and Operations Research* **152**, 106112.

Vidal, T., Crainic, T. G., Gendreau, M., Lahrichi, N. and Rei, W. (2012), 'A hybrid genetic algorithm for multidepot and periodic vehicle routing problems', *Operations Research* **60**(3), 611–624.

Vásquez, S. A., Angulo, G. and Klapp, M. A. (2021), 'An exact solution method for the tsp with drone based on decomposition', *Computers and Operations Research* **127**, 105127.

Yen, J. Y. (1971), 'Finding the *k* shortest loopless paths in a network', *Management Science* **17**(11), 712–716.

Zang, H., Ou, C. and Mukherjee, B. (2003), 'Path-protection routing and wavelength assignment (RWA) in WDM mesh networks under duct-layer constraints', *IEEE/ACM Transactions on Networking* **11**(2), 248–258.

Zhang, J., Campbell, J. F., Sweeney II, D. C. and Hupman, A. C. (2021), 'Energy consumption models for delivery drones: A comparison and assessment', *Transportation Research Part D: Transport and Environment* **90**, 102668.

Zhou, D. and Subramaniam, S. (2000), 'Survivability in optical networks', *IEEE Network* **14**(6), 16–23.

Finanziato dall'Unione europea- Next Generation EU, Missione 4, Componente 2, Investimento 4.1 (DM 351/2022) CUP J33C22001720002.