UNIVERSITÀ DEGLI STUDI DEL MOLISE

DIPARTIMENTO DI BIOSCIENZE E TERRITORIO

Ph.D. IN BIOLOGIA E SCIENZE APPLICATE

CURRICULUM SCIENZE APPLICATE

CYCLE XXXV

SSD MAT/09

Paths and Tours on Graphs: variants and extensions in the context of autonomous vehicles.

Advisor:

Prof. Giuseppe Izzo

Co-Advisor:

Prof. Giovanni Capobianco

Ph.D Coordinator:

Prof. Filippo Santucci De Magistris

Candidate:

Dott. Davide Donato Russo

Ph.D. thesis reviewers

Prof. **Diego Cattaruzza**
École Centrale de Lille
diego.cattaruzza@centralelille.fr

Prof. **Anne Meyer**
Technische Universität Dortmund
anne2.meyer@tu-dortmund.de

Evaluation committee

Prof. **Claudia Archetti**
Department of Information Systems, Decision Sciences, and Statistics, ESSEC
claudia.archetti@essec.edu

Prof. **Raffaele Cerulli**
Dipartimento di Matematica, Università di Salerno
raffaele@unisa.it

Prof. **Vittorio Latorre**
Dipartimento di Bioscienze e Territorio, Università degli studi del Molise
vittorio.latorre@unimol.it

Prof. **Anna Sciomachen**
Dipartimento di Economia, Università di Genova
sciomach@economia.unige.it

Contents

Outline

Human beings have always tried to find the best possible path to move from one location to another. Often, the best path means the shortest one. Even if the transport system never stopped changing, the problem of finding a shortest path is still valid and widely discussed. Nowadays, we are looking at a rapid diffusion of autonomous vehicles, such as drones, robots, forklifts, and cars. In this context, the problem of finding, in a reasonable amount of time several paths or tours, is becoming extremely important in various fields. The Shortest Path Problem (SPP) and the Vehicle Routing Problem (VRP) are defined to formalize these scenarios. The SPP aims to find a minimum length path that connects a source node s and a destination node d. The SPP is known to be solvable in polynomial time using various algorithms [62]. Nevertheless, several variants of the problem, which have many practical applications in the real world, are NP-Hard [69]. Furthermore, another interesting application field concerns the definition of a tour of minimum length that allows, for example, a carrier to serve all its clients and then go back to its deposit. For this reason, problems such as the Vehicle Routing Problem (VRP)[70], Travelling Salesman Problem (TSP)[65] [74], Arc Routing Problem (ARP)[56], and so on, have growing interest in the community of researchers. All these problems, and many of their variants, are known to be NP-hard [69]. In this thesis, we propose an overview of path and tour problems in the context of autonomous vehicles. In particular, concerning the definition of paths of minimum length, we present a new variant of the Shortest Path Problem, namely the Concurrent Shortest Path Problem which considers the possibility of having a set of autonomous vehicles that tries to find a path of minimum length while avoiding collisions. Then we present an exact reduction technique for the k-Colour Shortest Path Problem that considers the possibility of having a multi-modal transport system. For both of them, we propose exact and heuristic algorithms. Furthermore, while focusing on the definition of tours for drones, we present a new variant of the TSP, that is the Close-Enough Generalized Routing Problem, that combines the properties of the Close-Enough Travelling Salesman Problem (CETSP) and the Close-Enough Arc Routing Problem (CEARP).

In the first part of this thesis, we present a variant of the SPP, namely the Concurrent Shortest Path Problem, to define multiple paths of minimum length connecting several pairs of source-destination nodes while avoiding collisions between paths. This variant of the SPP can be applied to various practical scenarios. One consists of handling the traffic of a smart city ([87], [54]), where everything is interconnected and handled by a central system. In a possible futuristic scenario, where 100% of vehicles are 'smart' (autonomous), some possible purposes of the central system could be the minimization of the time required by each vehicle to reach its destination, the reduction of traffic, the minimization of the fuel consumption, etc. Similar applications that are already been applied in the industry consist of defining paths of minimum length for robots and forklifts in a warehouse ([52], [90]) or the tour definition for a fleet of drones ([85], [39]). In this thesis, we propose a variant of the SPP called Concurrent Shortest Path Problem (CSPP) to define paths for a fleet of vehicles to avoid collision while minimizing the maximum completion time. After a formal definition of the problem, we propose three Mixed-Integer Linear Programming (MILP) formulations. The first two are based on a flow formulation. The first one discretizes the time dimension to avoid collisions, and the second one uses a novel transformation of the graph to reduce the complexity of the problem. Finally, the last one is inspired by the Bin Packing Problem (BPP). Furthermore, given the high complexity of the formulations, we propose two heuristic approaches to obtain feasible solutions in a reasonable amount of time.

Other variants of the SPP that fit various real-world applications are defined on edge-coloured graphs. In particular, in the second part of this thesis, we focus our attention on a specific variant of the SPP defined on an edge-labeled graph, called the k-Colour Shortest Path Problem (k-CSPP). Labeled problems are widely discussed in the literature. They find applications mostly in telecommunications networks, such as packet routing, and in multi-modal transport systems also for autonomous vehicles. The k-CSPP consists of finding a shortest path in a weighted edge-coloured graph, where the maximum number of different colours that can be used is fixed to be k. The first formal definition of the k-Colour Shortest Path Problem is presented in the paper of Ferone et al. [37]. They propose a mathematical model based on a flow formulation and a Branch and Bound Algorithm (B&B) to solve the problem. In their second work [35], Ferone et al. propose a novel Dynamic Programming algorithm (DP) that uses a path-labeling approach with an A^*-like technique as an exploration strategy. The authors also define a new set of instances with fewer colours to analyze the behavior of their approaches, varying the number of colours. As a contribution to the k-Colour Shortest Path Problem, we propose a heuristic approach (CCDA) able to identify optimal or near-optimal solutions regardless of the size of the instances; we propose an effective graph reduction strategy, able to remove from the graph more than 90% of nodes and edges; and an exact approach that combines the two approaches described before and guarantees optimal solutions in reasonable running times. We perform several tests to address the effectiveness of the proposed approaches and to analyze the characteristics of the benchmark instances.

The second main topic handled in this thesis concerns the definition of tours. A tour is a path that starts and ends in a depot node and connects it with all the targets of the graph. Nowadays, problems that solve and extend this scenario are extremely important in various contexts. Given the rapid growth of technologies, such as wireless connection systems, problems like the TSP, and the ARP are evolving to adapt to new scenarios. In fact, the Close-Enough Arc Routing Problem (CEARP) [31], and the Close-Enough Travelling Salesman Problem (CETSP) [27] are proposed to handle these new possibilities. In these generalizations, to cover a target is not necessary to achieve its exact position. Each target has a range of action, that is considered visited if traversed by the vehicle. In both these generalizations, the aim consists of finding a shortest tour that starts and ends at the depot node and intersects each neighborhood once. These problems have several practical applications in the context of Unmanned Aerial Vehicles for military and civil missions like supply delivery (food, munition, etc.), geographic region monitoring and military surveillance [12]. Moreover, even the robot monitoring wireless sensor networks can be modeled using this problem [93] [78]. As a contribution to these problems, in this thesis, we propose the Close-Enough Generalized Routing Problem (CEGRP). It combines the properties of the CETSP and the CEARP, defining a new generalization where a tour can pass through constrained and unconstrained areas. This means that a tour cannot be defined using only algorithms for the CETSP, as well as for the algorithms for the CEARP. Considering the CETSP is closely related to drones, this new variant refers to a drone flying freely in space. Unfortunately, this scenario is not always true. If we consider locations such as schools, hospitals, military or residential areas, flying over these areas is usually prohibited for several reasons such as safety, public order, privacy, etc. In the Close-Enough Generalized Routing Problem, we introduce the concept of a flight zone. We differentiate two zones: one where the drone can fly freely, called the Free Flight Zone (FFZ), and the other where the possibility of flight is limited to specific corridors (e.g. roads) or prohibited, called the Constrained Flight Zone (CFZ). Given a graph $G = (N, E, T, Z)$, where N is the set of nodes, E is the set of edges (e.g. roads), T is the set of targets to serve, and Z is the set of CFZ, we need to find a tour that starts and ends at the depot, and minimizes the total sum of edge lengths. We formally define the problem and examine it. We also propose a heuristic approach, namely, Convert and Conquer (C&C) that performs a set of conversion steps to solve the CEGRP by reducing it into the Generalized Travelling Salesman Problem (GTSP). Future developments include defining approaches that can also handle intermediate situations, with variable mixes of CFZ and FFZ, and examining its behavior concerning its boundary cases.

This dissertation reports in detail on all described until now. Following this outline, we review the literature and state of the art in Chapter 1. We discuss variants of the Shortest Path Problem, namely CSPP and k-CSPP in Chapters 2 and 3. Then we consider the second main scenario concerning the definition of tours, defining the CEGRP and its characteristics in Chapter 4. In conclusion, we summarize the whole work in Chapter 5, discussing some conclusions and a few future developments.

Chapter 1

Literature Review

In this chapter, we present a brief discussion on the state of the art about the definition of paths and tours on graphs. Particularly, we focus first on the Shortest Path Problem (SPP), its variants, and some of the main solving algorithms, with particular attention to the coloured variants of the SPP. Then, we focus on the definition of a tour of minimum length for problems defined as Close-Enough Problems.

The Shortest Path Problem: Since its first definition in 1956 [40] the SPPis still one of the most important and discussed topics in operational research. Also, the number of publications about it remains high over the years ([62], [41]) due to the capacity of this problem to model many real-world applications, as evidenced by its innumerable variants. Given a graph G, a shortest path from a source node s to a destination node d is a path that connects the two nodes minimizing a specified length function. The SPP is known to be solved in polynomial time using various algorithms (e. g. the Dijkstra's Algorithm [30]), as reported by [62]. However, many of its variants are known to be NP-hard [86].

One of these applications consists, for example, of handling the traffic of a smart city as described by Variaya et al. [87] and Katrakazas et al. [54]. The authors present a scenario where 100% of vehicles are 'smart' (autonomous), and everything is interconnected and handled by a central system. In this context becomes extremely important to define a management system able to organize the traffic by defining the paths and the tours for each vehicle. Some possible purposes of the central system could be the minimization of the time required by each vehicle to reach its destination, the reduction of traffic, and the optimization of fuel consumption. Further applications related to autonomous vehicles consist of defining paths of minimum length for robots and forklifts in a warehouse ([52], [90]) or the tour definition for a fleet of drones ([85], [39]).

From a general point of view, most of the research about autonomous vehicles concerns the Vehicle Routing Problem (VRP) ([42], [57]), even though this problem does not focus on the possibility that two vehicles could collide between them. To close this gap, Lui et al. [60] and Bichiou et al. [7] propose an optimization model to optimize the movement of autonomous vehicles through intersections. Hassan et al. [49] proposes a fully distributed algorithm where vehicles close to an intersection cooperate to define a schedule that allows all of them to safely pass the intersection. Katrakazas et al. [54] propose a survey about real-time motion planning for autonomous vehicles, where different methods are compared. These methods could be: finding a path, searching for the safest maneuver, and determining the most feasible trajectory. All of these techniques aim to provide a safe and collision-free path for each vehicle while considering the length of the path and the traffic. Other authors focus on the definition of path or tour for vehicles such as robots. For example, Schouwenaars et al. [75] propose a new approach to fuel-optimal path planning of multiple vehicles that uses a combination of linear and integer programming (LP and ILP). This approach considers a receding time horizon where the paths are composed of a sequence of locally optimal segments. Therefore, the authors show that receding horizon strategies can lead the system to unsafe conditions.

In the scenario of defining independent paths for multiple vehicles, the current state of the art is represented by the Disjoint Shortest Path Problem (DSPP) proposed by Eliam et al. [32] in 1998. The DSPP is commonly used in communication networks to guarantee the reliability of transmissions ([43], [44]). It is defined on a graph $G = (N, A)$ with N the set of nodes, A the set of arcs, and a set of pairs of different nodes $SD = \{(s, d) : s, d \in N, s \neq d\}$. The problem aims to find $|SD|$ different disjoint shortest paths between each pair (s, d). The DSPP can be defined on directed or undirected graphs. The paths can be vertex or edge-disjoint. The problem is NP-complete, as shown by Eliam et al. [32] and Karp et al. [53].

Eliam et al. [32] propose a polynomial algorithm for the Two Disjoint Shortest Path Problem (2DSPP) in undirected graphs. Robertson et al. [72] present a polynomially bounded algorithm to verify if there are $|SD|$ vertex-disjoint paths in a graph G. The same authors, in [73], describe a method to solve the DSPP that, for fixed $|SD| \geq 0$, has a computational cost of $O(|V|^3)$. Sidhu et al. [79] discuss defining multiple disjoint paths between pairs of nodes in a network to increase the effective bandwidth, reduce congestion, and reduce the probability of dropped packets. The authors propose a distributed distance-vector algorithm to find multiple disjoint paths to a destination. Torrieri et al. [84] propose three efficient algorithms to identify a set of disjoint paths. The first can obtain an optimal set of disjoint paths, while the other two can construct larger sets of non-optimum paths. Kobayashi et al. [55] consider two different objective functions for the problem, the former wants to minimize the total path length (Min-Sum); the latter wants to minimize the length of the longest path (Min-Max), considering two and three pairs of distinct paths. Bjorklund et al. [9] define a polynomial-time Monte-Carlo algorithm to find disjoint paths of the smallest total length.

Other variants consider both multiple or single source-destination pairs. The one proposed by Weiner et al. [89] consists of finding the largest number of pairs that can be connected by paths, using each edge within the network at most once. Furthermore, Festa et al. [38] present the Shortest Path Tour Problem in which a shortest path from a given source node to a given destination node must be found such that it crosses a sequence of nodes that are given in a fixed order. One more variant of the DSPP addressing this kind of scenario is the Capacitated Shortest Multi-paths Problem proposed by Bentz et al. in [4]. This problem, given an undirected graph and a set of vertex pairs, consists of finding a path for each pair that respects the capacity constraints. It means that the number of paths that pass through each edge must be at most the capacity of this edge.

Finally, one last problem that must be considered is the multi-agent pathfinding (MAPF) problem [81] [76]. It is a generalization of the single-agent pathfinding that consists of a graph and a number of agents. For each agent, a start state and a goal state are given. The problem aims to find paths for all agents from their start states to their goal states, under the constraint that agents cannot collide during their movements. Time is assumed to be discretized, and in every time step, each agent is situated in one of the graph vertices and can perform a single action within two possibilities: wait and move. A wait action means that the agent stays in its current vertex for another time step. A move action means that the agent moves from its current vertex to an adjacent vertex. Several types of conflicts between agents are defined in [81]; for example vertex conflicts, edge conflicts following constraints, and so on.

Coloured Problems: In this thesis, we consider also the possibility of using different kinds of vehicles. This concept is defined in literature as multi-modal transport system [19]. Some variants of the SPP that consider this characteristic are defined on an edge-coloured or edge-labeled graph. In the remainder of this thesis, the terms 'colours' and 'labels' will be used interchangeably.

These graphs, characterised by an enumerable characteristic associated with the edges that are usually called colour or label, allow one to create generalised versions of many problems defined on graphs. The Minimum Labelling Spanning Tree (MLST) Problem [67], [11] is a generalisation of the classic Minimum Spanning Tree Problem that, thanks to the use of labels, has been exploited to model problems concerning telecommunications networks. Cerulli et al. [23] propose various extensions of the problem and investigated several approaches, comparing their results and characteristics. Furthermore, Cerrone et al. [21] introduce and analyse the Strong Generalised Minimum Label Spanning Tree Problem, presenting an Integer Linear Programming (ILP) formulation and three heuristic approaches. Silva et al. [28] provide a tighter bound on the time complexity of the MLST and propose a new Mixed-Integer Programming based (MIP) meta-heuristic, namely multi-start local branching. Strictly related to the MLST Problem is the k-Labelled Spanning Forest Problem. Cerulli et al. [22] define this problem and propose an exact approach to solve it, and Consoli et al. [25] present a comparison between the main meta-heuristic approaches. The Rainbow Spanning Forest Problem (RSFP) consists of finding a spanning forest of graph G such that the number of components (trees) is minimum, and each connected component contains only edges with different colours. Carrabs et al. [15], [16] propose this problem, studied its complexity, and then design two approaches for it, one exact and one greedy.

Labeled graphs have been used not only for tree and forest problems but also for path identification problems. In the Orderly Coloured Longest Path Problem (OCLPP), the aim is to find the longest possible path such that a set of constraints on the sequence of the colours is respected. Carrabs et al. [17] performed several tests of all existing formulations, summarising their characteristics, in order to determine which of the formulations obtains better results and under what conditions. Yuan et al. [94] developed an ILP formulation and a heuristic algorithm to minimise the number of colours for a path; indirectly, the authors succeeded in minimising the number of overlapping colours to prevent a single failure from causing multiple failures.

For a node-coloured graph, the All Colours Shortest Path Problem (ACSP), studied by Bilge et al. [8], consists of finding a minimum-cost shortest path that uses all the different colours in an undirected weighted graph, where each node is assigned a colour. Carrabs et al. [18] propose a variant of the problem where the source is unknown, and for this variant, they define a mathematical formulation and propose a Variable Neighborhood Search meta-heuristic. Finally, a problem family close to the coloured problem family consists of resource-constrained problems. The Resource Constrained Shortest Path Problem (RCSPP) [36], [80] is based on the definition of an L-dimensional vector of resources R in addition to the graph G. In particular, each edge is linked to a resource attribute that needs to be addressed during path planning. A deep analysis of the RCSPP, its applications, and possible approaches is performed by Irnich et al. [51], Avella et al. [1], Beasley et al. [2] and Di Puglia Pugliese et al. [71]. To solve the RCSPP, Marinakis et al. [63] propose an effective hybrid algorithm that combines the Particle Swarm Optimization meta-heuristic and a Variable Neighborhood Search approach. Tilk et al. [83], exploiting the asymmetry in the number of forward and backward label extensions for the Shortest Path Problem with Resource Constraints, propose an effective dynamic halfway-point algorithm. Zhu et al. [95] present an implementation of a three-stage approach for the dynamic version of the RCSPP that has to be solved as a sub-problem in the Branch and Price algorithm, while Horváth et al. [50] investigate a Linear Programming (LP) based Branch and Bound method and introduce new cutting planes and separation procedures for the problem. Finally, Di Puglia Pugliese et al. [29] analyse the problem in uncertain data conditions and propose a robust formulation to obtain optimal solutions.

In this thesis, we focus on an NP-Hard variant of the SPP that finds application in the context of multi-modal transport system, namely k-Colour Shortest Path Problem (k-CSPP). It consists of finding a shortest path in a weighted edge-coloured graph, where the maximum number of different colours that can be used is fixed to be k. The k-CSPP is defined by Ferone et al. in [37]. They propose a mathematical model based on a flow formulation and a Branch and Bound Algorithm based on the idea that by relaxing the colour constraints, the problem can be solved using a shortest path algorithm. After the computation of a relaxed solution, if the number of colours used in the solution is higher than k, the solution is inadmissible; otherwise, the new solution can be evaluated in terms of the path length. It becomes the new incumbent solution if it is better than the previous best solution; otherwise, it is discarded. The problem of finding a path between a source node and a destination node with a fixed maximum number of colours, namely k-CSPP, is proven to be NP-hard by Broersma et al. [10], who reduce it from the 3-SAT (3-Satisfiability) problem. In particular, the authors highlight that any instances of the problem of finding a path with at most k colours can be related to an instance of k-CSPP, where each edge has a null cost. Ferone et al. in [35], propose a dynamic programming (DP) algorithm that uses a path-labeling approach with an A^*-like technique as an exploration strategy. The authors also define a new set of instances with fewer colours to analyse the behaviour of their approaches, varying the number of colours. Testing their approaches on two test sets highlight that the DP algorithm outperformed previous approaches in terms of the number of optimum solutions and the computational time. An interesting discussion highlighting the differences between the RCSPP and the k-CSPP can be found in [37].

Close-Enough Problems: The second main topic of this thesis concerns the definition of tours on graphs. A tour is a path that starts from and ends at a depot node and connects it with all the targets of the graph. If the targets to cover are nodes, the problem of defining a tour connecting all of these nodes is known as Travelling Salesman Problem (TSP) [70], [65], [74]. Otherwise, if the tour is defined to connect a set of arcs, the problem is known as Arc Routing Problem (ARP) [56]. All these problems, and many of their variants, are known to be NP-hard [69]. Given the fast growth of technologies, such as wireless connection systems [78], these problems are evolving to adapt to new scenarios. The CEARP [31], and the CETSP [27] are proposed to handle these new scenarios. In these generalizations, a target is equipped with a proximity sensor (e. g. an R-FID tag), so to cover a target is not necessary to achieve the exact position of a target; instead, a path must be close enough to the target t to cover it. Each target has a neighborhood of radius r and if the distance between an edge e and the center of the target t is lower than or equal to the radius, it is possible to affirm that the edge e covers the target t.

The CETSP is a variant of the TSP that consists of finding the shortest tour that starts and ends at the depot node and intersects each neighborhood once. The CETSP has several practical applications in the context of Unmanned Aerial Vehicles for military and civil missions like supply delivering (food, munition, etc.), geographic region monitoring and military surveillance [12]. Moreover, even the robot monitoring wireless sensor networks can be modelled using this problem [93] [78]. Behdani et al. [3], propose a mixed integer programming formulation based on a discretization schema for the problem. Their approach allows the definition of both upper and lower bounds for the problem. Gulczynski et al. [45] propose an interesting discussion about several heuristics proposed in the literature to face the

CETSP. Yang et al. [92] present an effective double-loop hybrid algorithm based on particle swarm optimization and genetic algorithm. Furthermore, Carrabs et al. present in their work, a heuristic approach to compute the lower and the upper bound for the problem [14] [12]. They also present an effective discretization schema for the target neighborhoods [13]. Finally, Wang et al. [88] present a Steiner Zone Variable Neighborhood Search heuristic (SZVNS) able, in a small amount of time, to obtain good solutions even on big instances.

The CEARP is a generalization of the Arc Routing Problem (ARP). In this problem, unlike the CETSP, a set of candidate arcs is provided. A subset of them, of minimum cost, has to be selected as a solution tour that intersects the neighborhood of all targets. An arc covers a target if the target is either on the arc or within a predetermined distance (radius) from the arc. The CEARP was presented by Drexl et al. [31]. The authors demonstrate that the CEARP is NP-Hard, and propose a Branch and Cut algorithm to optimally solve the problem.

Há et al. [47] introduce a mixed-integer programming formulation for the problem; the same authors present also a new ILP formulation in [48]. Furthermore, Cerrone et al. [20] propose a flow formulation for the problem and propose an effective technique to reduce the size of the input graph. Their experiments address the effectiveness of the reduction techniques. An extremely detailed overview of the ARP, its variants, and its future possibilities is proposed by Corberan et al. [26]. Bianchessi et al. [6] propose the Min-Max Close-Enough ARP; in this problem, a fleet of vehicles, while serving a set of customers, has to balance the length of different tours. The authors propose a Branch and Cut, a Branch and Price, and a heuristic algorithm. Furthermore, the same authors also propose an extension of the CEARP, namely the Profitable Close-Enough Arc Routing Problem (PCEARP) [5]. It associates a profit to each customer that is obtained when the target is served.

Finally, the GTSP is an extension of the TSP, defined by Laporte et al. [59], where, given a set of targets and a partition of them into groups, we want to find a minimum length tour that includes exactly one target from each group. This problem finds applications in numerous fields, including air-plane routing [46], mail delivery, material flow system design, vehicle routing [58], and warehouse order picking [68]. Carrabs et al. [12] define one possible formulation for the GTSP. Several authors propose applications of meta-heuristic approaches, such as the Ant Colony Optimization of Yang et al. [91] or the Particle swarm optimization algorithm of Shi et al. [77].

Chapter 2

The Concurrent Shortest Path Problem

2.1 Introduction

A smart city is an urban area where, thanks to innovative technologies, it is possible to optimize and improve infrastructures and services. Cities are evolving and becoming smarter. Both public and private urban mobility are changing. In particular, the advent of autonomous driving vehicles opens up new possibilities for optimizing and rationalizing urban traffic. This chapter focuses on the scenario of a smart city ([87], [54]), in which all the vehicles are autonomously driven, interconnected, and handled by a central system. In this scenario, we consider that a set of vehicles, in the same street network, and at the same time interval, want to reach their destination. Other applications consist of defining paths of minimum length for robots and forklifts in a warehouse ([52], [90]) or the route definition for multiple drones ([85], [39]).

The simplest way to model this scenario is the SPP. The case with a single vehicle can be solved in polynomial time using well-known algorithms for the SPP ([30]). Therefore, in this new problem, we address the scenario in which we have more vehicles that try to reach their destinations concurrently. Actually, this problem is formalised by the Disjoint Shortest Path Problem (DSPP) ([32], [79], [84]). The DSPP is commonly used in communication networks to guarantee the reliability of transmissions ([43], [44]). It is defined on a graph $G = (N, A)$ with N the set of nodes, A the set of arcs, and a set of pairs of different nodes $SD = \{(s, d) : s, d \in N, s \neq d\}$. The problem aims to find $|SD|$ different disjoint shortest paths between each pair (s, d). The DSPP can be defined on directed or undirected graphs. The paths can be vertex or edge-disjoint. The problem is NP-complete, as shown in [32] and [53].

The vertex disjoint variant of the DSPP has two main limitations. First, a node is usable in only one path; this is not realistic because a node of a street graph must be used by multiple vehicles. We address this problem by modifying the graph G

by considering the time dimension. Second, traffic handling is not considered at all in the DSPP. We address this problem by considering the capacity of the arcs. Capacity means how many vehicles can travel the street simultaneously by keeping the minimum safety distance between them. The Capacitated Shortest Multi-paths Problem ([4]) is a variant of the DSPP. Given an undirected graph and a set of vertex pairs, it consists of finding a path for each pair that respects capacity constraints on arcs. It means that the number of paths that pass through each arc must be at most its capacity. Our approach differs from the Capacitated Shortest Multi-paths Problem ([4]) because we consider the capacity of each arc also from a temporal point of view.

To effectively model this scenario, we introduce the Concurrent Shortest Path Problem (CSPP), a generalization of the DSPP that consists of defining multiple paths of minimum length connecting all source-destination pairs while avoiding collisions between paths. Other possible objectives for the problem could be the minimisation of the time required by each vehicle to reach its destination, the traffic reduction, or the fuel consumption minimisation. We propose three mathematical formulations for the problem. The first is based on a bin packing formulation, the second is based on a flow formulation that considers the time dimension, and the third is again a flow formulation with the introduction of a graph transformation to remove the time dimension and the introduction of new constraints to handle the conflicts between arcs. Furthermore, to solve large-size instances, we propose two heuristic approaches. Computational results over a benchmark of randomly generated instances show that the proposed heuristics can produce good solutions in a reasonable amount of time.

Finally, one last observation is required to clarify the differences between the CSPP and the Multi-Agent Path-finding (MAPF) Problem [81] [76]. This problem aims to find paths for all agents from their start states to their goal states, under the constraint that agents cannot collide during their movements. Therefore, this problem differs from the CSPP because the MAPF considers two possible actions for each agent, wait and move. Instead, in the CSPP, the wait action is not allowed, and once started, a vehicle must keep moving until it reaches its destination node. Furthermore, a vehicle in a given time instant, for the CSPP can be in movement in a given arc, while in the MAPF it will surely be in a vertex of the graph. Surely, the two problems are close in terms of characteristics, so we can assume that some transformation in the structure of the instances and the algorithms can allow us to compare the state-of-the-art approaches proposed for the two problems. This is an interesting possibility for future research.

The rest of this chapter is organized as follows. Section 2.2 presents a formal definition of the problem. Section 2.3 proposes three mixed integer linear programming models (MILP models). Section 2.3 also describes an effective graph transformation used to remove the time dimension in the problem definition. Section 2.4 contains the description of two heuristic approaches to handle large-sized instances. Section 2.5 provides an analysis of the experiments performed to assess the quality of the proposed algorithms. Finally, Section 2.6 presents some conclusions and possibilities for future research.

2.2 Problem Definition

The CSPP considers a set of vehicles that, starting from a source node, want to reach their destination node. The problem aims to minimize the maximum completion time (Min-Max).

We now describe the mathematical formulation for the CSPP. The formulation is based on a directed graph $G = (N, A)$, where $N = \{1, 2, ..., n\}$ is the set of nodes, and $A = \{(i, j) : i, j \in N, i \neq j, \exists!(i, j) \forall i, j \in N\}$ is the set of arcs between nodes. We note that the graph is not complete and so $A \subseteq N \text{x} N$.

For each arc (i, j) we are given:

- the cost $c_{ij} \in \mathbb{Z}^+$, interpreted as the travel time of the arc;
- the capacity of the arc $g_{ij} \geq 0$, interpreted as the maximum number of vehicles that can be on the specific arc at the same time.

Figure 2.1 shows an example of the CSPP. Since vehicles cannot pass each other, the queue on an arc is considered a FIFO queue: the first vehicle entering an arc is also the first vehicle leaving it.

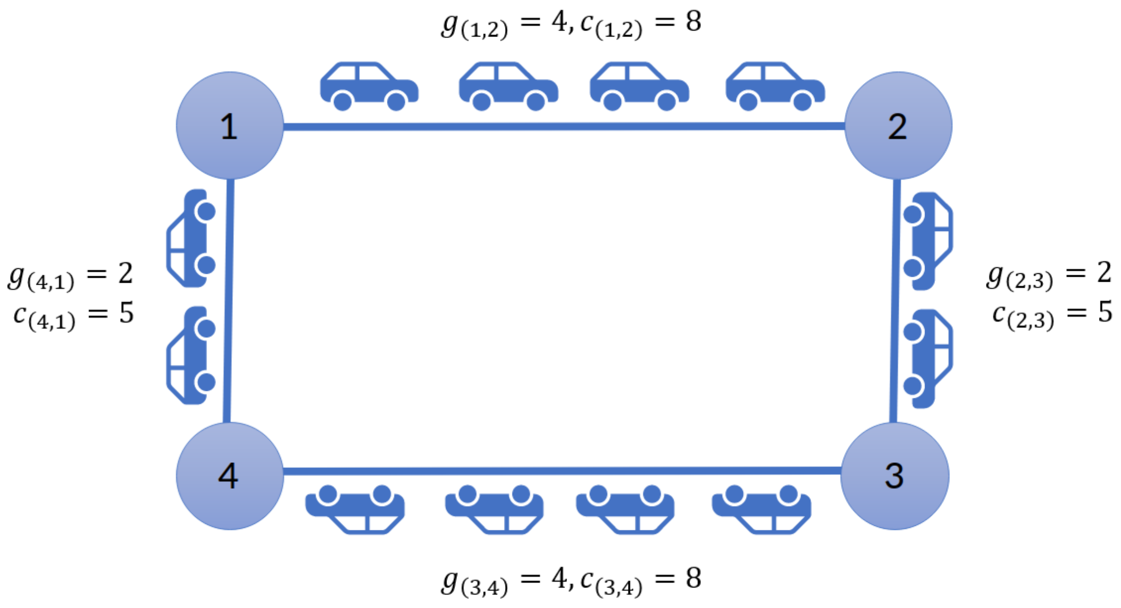


Figure 2.1: This figure shows an example of the capacity queue. On arc $(1, 2)$ the capacity is equal to 4, for this reason at most 4 vehicles can be on that arc at the same time.

Let $T = \{0, \alpha, 2\alpha, ..., T_{\max}\}$ be the set of time instants, indexed by t ($T_{\max}$ should be as small as possible to reduce the complexity of the problem). We impose that the smallest time instant considered to discretize the time horizon is α. More information about the α can be found in Section 2.2.1.

We define the set $SD \subseteq N \times N$ as the set of distinct source-destination pairs, also referred to as vehicles. For each source-destination pair (s, d), we are given a release time $t_{sd} \geq 0$, which is defined as the minimum starting time for the vehicle (s, d).

The objective of the problem consists of routing all SD-pairs so that the time instant at which all pairs have reached their destination is minimized, two distinct vehicles do not cross the same node at the same time (collisions), and the vehicles do not exceed the capacity of the arcs.

2.2.1 α value

One parameter that should be analysed is the length of the minimum time instant defined as α. The usage of time in the problem definition gives us the possibility to know the exact position of each vehicle at any time. In practice, this is not possible due to technological limitations, so we define a constant value α that is used to discretize the time horizon and take into account various aspects of real settings (i.e. the safety distance between vehicles).

The safety distance is the minimum distance that must be between two consecutive vehicles in motion, and it allows vehicles to stop without provoking collisions. The safety distance is commonly considered equal to the reaction space (given by the space needed by the driver to get the perception of danger minus the start braking space) plus the braking space (which is given by the start braking space minus the space required to the vehicle to stop).

According to [24] and [61], the ideal safety distance between vehicles with a speed of $25km/h$ is almost $13m$. In our work, considering the proposed scenario with 100% of vehicles completely autonomous, and considering that these kind of vehicles have proximity sensors and high-quality cameras to observe the environment around them and predict collisions, we can assume that the reaction space could be considered almost equal to the ideal reaction time for a human, and so $0.5s$ ([24]). This causes a reduction of the safety distance; the obtained one, considered in our work is $6m$. Now, given that, with a speed of $20km/h$ considered as reference, a vehicle covers almost $5.5m$ in one second, and given that, the minimum time instant should be considered big enough to guarantee the safety distance between two vehicles, in our work we define $\alpha = 1$, in the way to guarantee $6m$ between each pair of moving vehicles.

2.3 Mathematical Formulations

In this section, we present the three formulations for the CSPP. In Section 2.3.1, we describe a Bin-Packing Formulation (BPF), which makes use of separation constraints proposed for Bin-Packing Problems. Section 2.3.3 presents a Time Flow

Formulation (TFF) based on the classic flow formulation for routing problems. In Section 2.3.4, we present the graph transformation procedure, which removes the time dimension from the temporal graph by introducing a set of conflicts. The related formulation is a flow formulation with a new set of constraints to handle the conflicts between arcs.

2.3.1 Bin-Packing Formulation (BPF)

We propose in this section a first formulation inspired by Bin-Packing formulations. The Bin-Packing Problem ([64]), consists of defining, among a finite set of bins of limited capacity, where a set of items must be packed. The problem aims to minimise the number of used bins while assigning all items to the containers. This problem has several applications, such as filling up containers, loading trucks with weight capacity constraints, creating file backups, and so on.

In the BPF we consider also the constant value namely $\alpha \geq 0$, which represents the minimum time distance between vehicles crossing the same arcs. It is related to the safety distance that vehicles must have, between them, while driving. Further information about the α value can be found in Section 2.2.1.

The BPF formulation uses the following sets of binary and real variables:

$$y_{ij}^{sd} = \begin{cases} 1, & \text{if arc } (i,j) \text{ is used by vehicle } (s,d) \\ 0, & \text{otherwise} \end{cases}$$

$$x_{i}^{sd} = \begin{cases} 1, & \text{if node } i \text{ is crossed by vehicle } (s,d) \\ 0, & \text{otherwise} \end{cases}$$

$$B_{ij}^{sd} \geq 0, \text{beginning time of vehicle } (s,d) \text{ on arc } (i,j)$$

$$E_{ij}^{sd} \geq 0, \text{ending time of vehicle } (s,d) \text{ on arc } (i,j)$$

$$v_{i}^{sd,s'd'} = \begin{cases} 1, & \text{if vehicles } (s,d) \text{ and } (s',d') \text{ both cross node } i \\ & \text{and } (s,d) \text{ comes before } (s',d') \\ 0, & \text{otherwise} \end{cases}$$

A mathematical model for the CSPP, based on a multi-commodity flow formulation and on bin-packing constraints following ideas taken from [34], is the following:

$$\min C_{\max} \tag{2.1}$$

$$\text{s.t. } C_{\max} \geq E_{id}^{sd} \qquad \forall (s,d) \in SD, \forall i \in N : (i,d) \in A \tag{2.2}$$

$$\sum_{j \in N} y_{sj}^{sd} = 1 \qquad \forall (s,d) \in SD \tag{2.3}$$

$$\sum_{j \in N} y_{jd}^{sd} = 1 \qquad \forall (s,d) \in SD \tag{2.4}$$

$$\sum_{i \in N} y_{ik}^{sd} = \sum_{j \in N} y_{kj}^{sd} \qquad \forall (s,d) \in SD, \forall k \in N : k \neq s,d \tag{2.5}$$

$$x_s^{sd} = 1 \qquad \forall (s,d) \in SD \tag{2.6}$$

$$x_d^{sd} = 1 \qquad \forall (s,d) \in SD \tag{2.7}$$

$$x_k^{sd} = \sum_{j \in N} y_{kj}^{sd} \qquad \forall (s,d) \in SD, \forall k \in N : k \neq d \tag{2.8}$$

$$x_k^{sd} = \sum_{j \in N} y_{jk}^{sd} \qquad \forall (s,d) \in SD, \forall k \in N : k \neq s \tag{2.9}$$

$$v_i^{sd,s'd'} + v_i^{s'd',sd} \geq x_i^{sd} + x_i^{s'd'} - 1 \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \forall i \in N \tag{2.10}$$

$$v_i^{sd,s'd'} + v_i^{s'd',sd} \leq 1 \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \forall i \in N \tag{2.11}$$

$$v_i^{sd,s'd'} \leq x_i^{sd} \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \forall i \in N \tag{2.12}$$

$$v_i^{sd,s'd'} \leq x_i^{s'd'} \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \forall i \in N \tag{2.13}$$

$$B_{ij}^{sd} \leq M y_{ij}^{sd} \qquad \forall (s,d) \in SD, \forall (i,j) \in A \tag{2.14}$$

$$E_{ij}^{sd} \leq M y_{ij}^{sd} \qquad \forall (s,d) \in SD, \forall (i,j) \in A \tag{2.15}$$

$$t_{sd} \leq \sum_{i \in N:(s,i)\in A} B^{sd}_{si} \qquad \forall (s,d) \in SD \tag{2.16}$$

$$B^{sd}_{ij} + c_{ij} \leq E^{sd}_{ij} + M(1 - y^{sd}_{ij}) \qquad \forall (s,d) \in SD, \forall (i,j) \in A \tag{2.17}$$

$$B^{sd}_{jk} \leq E^{sd}_{ij} + M(2 - y^{sd}_{jk} - y^{sd}_{ij}) \qquad \forall (s,d) \in SD, \forall i,j,k \in N : (i,j) \in A, (j,k) \in A \tag{2.18}$$

$$B^{sd}_{jk} \geq E^{sd}_{ij} - M(2 - y^{sd}_{jk} - y^{sd}_{ij}) \qquad \forall (s,d) \in SD, \forall i,j,k \in N : (i,j) \in A, (j,k) \in A \tag{2.19}$$

$$\sum_{k \in N:(j,k)\in A} B^{sd}_{jk} + \alpha \leq \sum_{i \in N:(j,i)\in A} B^{s'd'}_{ji} + M(1 - v^{sd,s'd'}_{j}) \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \quad \forall j \in N : j \neq d, j \neq d' \tag{2.20}$$

$$\sum_{k \in N:(k,j)\in A} E^{sd}_{kj} + \alpha \leq \sum_{i \in N:(i,j)\in A} E^{s'd'}_{ij} + M(1 - v^{sd,s'd'}_{j}) \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \quad \forall j \in N : j \neq s, j \neq s' \tag{2.21}$$

$$\sum_{j \in N:(j,i)\in A} E^{sd}_{ji} + \alpha \leq \sum_{k \in N:(i,k)\in A} B^{s'd'}_{ik} + M(1 - v^{sd,s'd'}_{i}) \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \quad \forall i \in N : i \neq s, i \neq d' \tag{2.22}$$

$$\sum_{k \in N:(i,k)\in A} B^{sd}_{ik} + \alpha \leq \sum_{j \in N,(j,i)\in A} E^{s'd'}_{ji} + M(1 - v^{sd,s'd'}_{i}) \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \quad \forall i \in N : i \neq d, i \neq si \tag{2.23}$$

$$v^{sd,s'd'}_{i} + v^{sd,s'd'}_{j} + v^{s'd',sd}_{i} + v^{s'd',sd}_{j} \geq 2(y^{sd}_{ij} + y^{s'd'}_{ij} - 1), \qquad \forall (s,d),(s',d') \in SD : (s,d) \neq (s',d'), \forall (i,j) \in A \tag{2.24}$$

$$v^{sd,s'd'}_{i} + v^{s'd',sd}_{j} - 1 \leq M(2 - y^{sd}_{ij} - y^{s'd'}_{ij}), \qquad \forall (s,d),(s',d') \in SD, \forall (i,j) \in A \tag{2.25}$$

$$v^{sd,s'd'}_{j} + v^{s'd',sd}_{i} - 1 \leq M(2 - y^{sd}_{ij} - y^{s'd'}_{ij}), \qquad \forall (s,d),(s',d') \in SD, \forall (i,j) \in A \tag{2.26}$$

The objective function (2.1) consists of minimizing the maximum end time between all the vehicles. This value is represented by the real variable $C_{\max}$, which, on constraints (2.2) is set to be greater than or equal to all variables E_{id}^{sd}. Constraints (2.3) and (2.4) ensure that exactly one arc has to go out from the source node s, and that exactly one arc has to enter the destination node d for each vehicle in SD. Constraints (2.5) guarantee the flow conservation. Constraints (2.6) and (2.7) assure that the source and the destination nodes are used for each $(s, d) \in SD$. Constraints (2.8) and (2.9) guarantee that if node $i \in N$ is used by a vehicle $(s, d) \in SD$, then there must be, respectively, an exiting and an entering arc. Note that these constraints are not defined for the destination node d and the source node s. Specifically, the combination of constraints (2.3), (2.4), (2.5), (2.6), (2.7), (2.8) and (2.9) guarantee that the source and the destination nodes for each vehicle are used. Furthermore, if they are used there must be an arc exiting the source node and entering the destination node. They assure that, for each node, except for source and destination nodes, the number of entering and exiting arcs is equal. We note that we need to use both node and arc variables because both of them are required to handle the beginning and the ending time of each arc and to avoid collisions.

Constraints (2.10) ensure that at most one of $v_i^{sd,s'd'}$ and $v_i^{s'd',sd}$ can take value 1, and constraints (2.11) ensure that at most one of (s, d) or (s', d') goes before the other on node $i \in N$. Constraints (2.12) and (2.13) guarantee that, given a node $i \in N$ and a pair of vehicles $(s, d), (s', d')$, variables $v_i^{sd,s'd'}$ are 0 when one of the two variables x_i^{sd} or $x_i^{s'd'}$ are 0. The combination of constraints (2.10), (2.11), (2.12) and (2.13) is used to set the order in which two vehicles can cross a common node. (s, d) goes before (s', d') or vice-versa.

Then, constraints (2.14) and (2.15) set B_{ij}^{sd} and E_{ij}^{sd} to 0 if (s, d) does not use the arc (i, j). The value of M is described in Section 2.5.1. Constraints (2.16) set the minimum starting time, namely the release time, for each vehicle. Constraints (2.17) assure that if a vehicle (s, d) uses an arc (i, j), the difference between the end and the beginning time on that arc is at least equal to the time length of that arc (c_{ij}). Constraints (2.18) and (2.19) guarantee that all vehicles cannot wait on nodes during their movement. Given a node $j \in N$, the ending time of the arc $(i, j) \in A$ entering j node must be equal to the starting time of the arc $(j, k) \in A$ exiting that node.

Constraints from (2.14) to (2.19) are used to assure the consistency between the beginning and the ending time of every single vehicle. Consistency means that: in each arc, if it is used, the ending time must be higher than the beginning time, the time required to cover an arc must be at least equal to its cost, and a vehicle cannot wait on nodes.

Constraints (2.20) impose that if one vehicle begins moving from a node j at time t, then any other vehicle should start at least α time instant after the first one. Constraints (2.21) impose that if one vehicle arrives on a node j at time t, then another vehicle should arrive on the same node at least α units of time after the first one. Constraints (2.22) assure that if a vehicle ends its movement on a node i at time

t, then another vehicle should start moving from that node, at least α units of time after. Constraints (2.23) assure that if a vehicle starts its movement from a node i at time t, then another vehicle should end moving in that node, at least α units of time after. Furthermore, these two constraints are not valid, respectively for the destination and the source node of each vehicle. We note that (2.22) do not consider ending time in the source node of (s, d) and starting time from the destination node of (s', d'); on the other side, (2.23), do not consider starting time in destination node of (s, d) and ending time in source node of (s', d'). Constraints from (2.20) to (2.23) are used to assure the consistency between the beginning and the ending time between all pairs of vehicles. Consistency in terms of different vehicles having different starting and ending times on nodes, the ending time of a vehicle on a node must be different from the starting time of another vehicle and vice-versa.

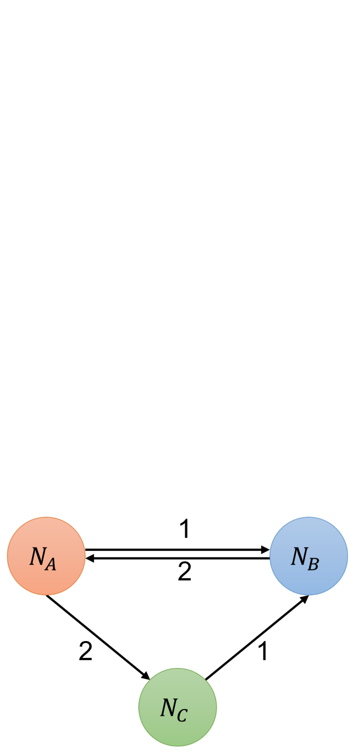
Finally, constraints from (2.24) to (2.26) are used to avoid the overcoming between vehicles. This is assured imposing that if two vehicles $(s, d) \in SD$ and $(s', d') \in SD$ both cross arc $(i, j) \in A$, then the order in which the vehicles cross node i must be the same order in which the vehicles cross node j. If both variables y_{ij}^{sd} and $y_{ij}^{s'd'}$ are equal to one, then, at least two variables between $v_i^{sd,s'd'}$, $v_j^{sd,s'd'}$, $v_i^{s'd',sd}$, and $v_j^{s'd',sd}$ must be equal to one. The case $v_i^{sd,s'd'} = v_i^{s'd',sd} = 1$ is prohibited by constraints (2.10). The cases $v_i^{sd,s'd'} = v_j^{s'd',sd} = 1$ or $v_j^{sd,s'd'} = v_i^{s'd',sd} = 1$ are prohibited by constraints (2.25) and (2.26); we note that these two cases are allowed only when both vehicles (s, d) or (s', d') do not use arc (i, j). Thus, the only two cases allowed are the one in which (s, d) enters and exits the arc (i, j) before (s', d') or vice-versa and so no overcomes are allowed.

We note that the validity of the capacity constraints of a generic arc $(i, j) \in A$ is assured by imposing the minimum distance, represented by the α value, between vehicles in constraints 2.20, 2.21, 2.22, and 2.23.

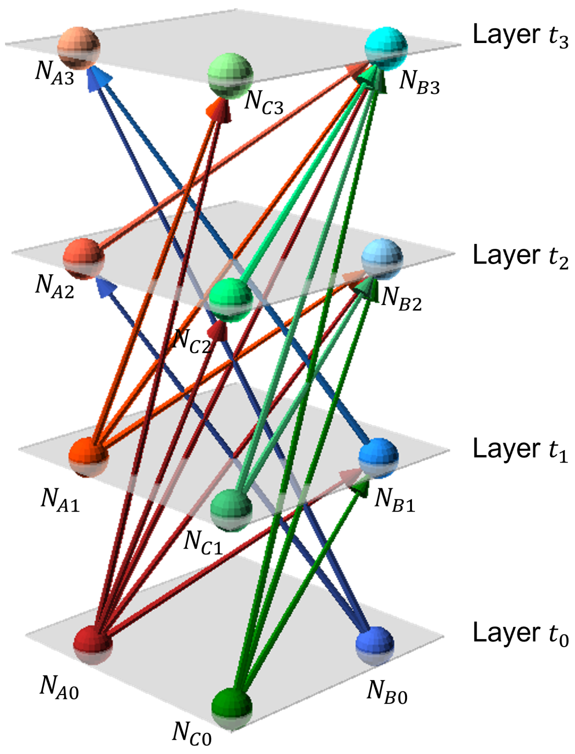
2.3.2 Temporal Graph Definition

In this section, we formalize the concept of the temporal graph as an extension of the previous graph $G = (N, A)$ where we consider nodes and arcs associated with time instants, namely temporal nodes and temporal arcs. More precisely, a temporal graph is a pair $G^T = (N^T, A^T)$, where $N^T = \{i^t = (i, t) : i \in N, t \in T\}$ is the set of temporal nodes and $A^T = \{(i^t, j^h) : i^t, j^h \in N^T, h \geq t + c_{ij}\}$ is the set of temporal arcs that connect pairs of temporal nodes. Each temporal node is represented by a node of the original graph $i \in N$ and a time instant $t \in T$. Besides, note that $(i^t, j^h) \in A^T$ if and only if it is possible to reach node $j \in N$ at time h starting from node $i \in N$ at time t.

In Figure 2.2, we can see an example of the original graph G and the correspondent temporal graph G^T.



(a) Original Graph G



(b) Temporal Graph G^T

Figure 2.2: This Figure shows the transformation of Graph (a) in a Temporal Graph (b); as can be seen, there are four layers, each one associated with a time instant (e. g. from t_0 to t_3); it is possible to see that each layer contains all the nodes of G, while there are no arcs between the temporal nodes of the same layer. Very important to highlight is that all the temporal arcs are directed and the direction is from layer t_i to layer t_j with $i < j$.

2.3.3 Time-Flow formulation (TFF)

The second formulation proposed makes use of the following decision variables, defined on $SD \times A^T$:

$$x_{i,t,j,h,sd} = \begin{cases} 1, & \text{if vehicle } (s,d) \in SD \text{ uses temporal arc } (i^t, j^h) \\ 0, & \text{otherwise} \end{cases}$$

Besides, we also use a positive variable $C_{\max}$ to denote the point in time at which the last SD-pair reaches its destination.

From these variables, model TFF consists of:

$$\min \quad C_{\max} \tag{2.27}$$

$$\text{s.t.} \quad C_{\max} \geq (h\, x_{i,t,j,h,sd}), \forall i^t, j^h \in N^T, (s,d) \in SD \tag{2.28}$$

$$\sum_{(s^t,i^h)\in A^T : t \geq t_{sd}} x_{s,t,i,h,sd} = 1, \forall (s,d) \in SD \tag{2.29}$$

$$\sum_{(i^t,d^h)\in A^T : t\geq t_{sd}} x_{i,t,d,h,sd} = 1, \forall (s,d)\in SD \tag{2.30}$$

$$\sum_{j^h:(i^t,j^h)\in A^T} x_{i,t,j,h,sd} = \sum_{k^h:(k^h,i^t)\in A^T} x_{k,h,i,t,sd}, \tag{2.31}$$
$$\forall i^t \in N^T : i \neq s, i\neq d, (s,d)\in SD$$

$$\sum_{(s'd')\in SD}\sum_{k^h:(k^h,s^t)\in A^T} x_{k,h,s,t,s'd'} \leq M(1 - \sum_{j^h:(s^t,j^h)\in A^T} x_{s,t,j,h,sd}), \tag{2.32}$$
$$\forall (s,d)\in SD$$

$$\sum_{(s'd')\in SD}\sum_{j^h:(d^t,j^h)\in A^T} x_{d,t,j,h,s'd'} \leq M(1 - \sum_{j^h:(j^h,d^t)\in A^T} x_{j,h,d,t,sd}), \tag{2.33}$$
$$\forall (s,d)\in SD$$

$$\sum_{(s',d')\in SD}\sum_{tt<t}\sum_{hh>h} x_{i,tt,j,hh,s'd'} \leq M(1 - \sum_{(s,d)\in SD} x_{i,t,j,h,sd}), \tag{2.34}$$
$$\forall (i^t, j^h)\in A^T$$

$$\sum_{(s,d)\in SD}\sum_{h,k\in T, k\leq t<h} x_{i,k,j,h,sd} \leq g_{ij}, \forall (i,j)\in A, t\in T \tag{2.35}$$

The objective function (2.27) of the problem consists of minimizing the value of the variable $C_{\max}$, the maximum end time among all the used arc variables, as defined by constraints (2.28). Constraints (2.29) and (2.30) impose that exactly one arc has to go out from the source and has to enter the destination for each SD-pair, regardless of the starting and the ending time, for each vehicle. The release time of each vehicle is assured by constraints (2.29) and (2.30) because the starting time of each arc exiting the source node and entering the destination node must be greater than the release time of each vehicle. Constraints (2.31) are the flow conservation constraints, which assure that the number of entering and exiting arcs for each vehicle is the same. We note that constraints (2.31) do not include the source and the destination nodes of each vehicle.

Furthermore, we need to assure that the source node for vehicle $(s,d)\in SD$ is not used by other vehicles at the same time as $(s,d)\in SD$. The same is for the destination node for vehicle $(s,d)\in SD$. This is because constraints (2.31) do not include the source and the destination nodes of each vehicle. To handle this scenario, constraints (2.32) impose that when a vehicle $(s,d)\in SD$ exits from its source node s at time t no other vehicles can enter the same node at the same time, therefore other vehicles $(s',d')\in SD$ can use the source node s of (s,d) in different time instant $t'\neq t$. As well for destination nodes, constraints (2.33) impose that when a vehicle $(s,d)\in SD$ enters its destination node d at time t no other vehicles can exit the same node at the same time, therefore other vehicles $(s',d')\in SD$ can use the destination node d of (s,d) in different time instant $t'\neq t$.

Constraints (2.34) are used to avoid overcomes between vehicles. This means that if vehicle $(s,d)\in SD$ uses temporal arc $(i^t,j^h)\in A^T$, then any other vehicle cannot use temporal arcs $(i^{tt}, j^{hh}) \in A^T : tt < t, hh > h$. Finally, constraints (2.35) assure that the total number of vehicles that can be in a queue in the same time instant $t \in T$ on the same arc $(i, j) \in A$ cannot exceed the capacity g_{ij} of the arc.

2.3.4 Graph Transformation with alternative Flow Formulation (AFF)

In this section, we present an alternative formulation based on a transformation of the graph applied on the temporal graph described in Section 2.3.2.

The new formulation proposed in this section performs a further transformation step on the temporal graph to reduce it into a planar graph to remove the time dimension. The transformation process is the following. Given a temporal graph $G^T = (N^T, A^T)$, we create a new graph $G' = (N', A')$ such that, for each temporal node $i^t \in N^T$ a new node $n \in N'$ is created. For each temporal arc $(i^t, j^h) \in A^T$, a new arc $(n, m) \in A'$ is created. The cost of the arc $(n, m) \in A'$ is $c_{nm} = h - t$. We note that $c_{nm} \geq c_{ij}$.

To handle the various incompatibility caused by the usage of temporal arcs at the same time by different vehicles, we define a set $\mathcal{H}$ of Compatibility Sets. It is a set of sets of arcs $(n, m) \in A'$ that cannot be used in the same solution by different vehicles. These sets are created to avoid overcoming between vehicles and exceeding the arc's capacity.

Given $\mathcal{H}$ a set of subsets of arcs, namely Compatibility Set, where each $H \in \mathcal{H}$ is a subset of A' with $|H| \geq 2$, we define a function

$$f_H : \mathcal{H} \to \mathbb{Z}^+ \tag{2.36}$$

that $\forall H \in \mathcal{H}$ assigns an integer value $f_H(H) \leq |H|$. This number represents the maximum number of elements of H that can be in an admissible solution.

In particular, $\forall (i, j) \in A, tt \in T$ a set $H \in \mathcal{H}$ is defined. H contains all the arcs $(n, m) \in A'$ that, in the graph transformation process, are created from the arc $(i^t, j^h) \in A^T$ where $t \leq tt \leq h$. Finally, the value of $f_H(H)$ is set equal to the capacity of the arc g_{ij} of $(i, j) \in A$. For example, consider as arc $(i, j) \in A$ the one between N_C and N_B in Figure 2.2a and consider $tt = 1$. Then, the corresponding H will contain the following temporal arcs of Figure 2.2b: $\{(N_{C0}, N_{B1}), (N_{C0}, N_{B2}), (N_{C0}, N_{B3}), (N_{C1}, N_{B2}), (N_{C1}, N_{B3})\}$. If for this example, we impose the capacity equal to its cost, namely 1, then $f_H(H) = 1$; so at most one arc, $(n, m) \in H$ can be in a possible solution for the CSPP. We note that the arc (N_{C2}, N_{B3}) does not belong to the set H because it starts after the time instant $tt = 1$.

Considering now, a vehicle $(s_1, d_1) \in SD$ using the temporal arc $(i^t, j^h) \in A^T$ and a vehicle $(s_2, d_2) \in SD$ following the temporal arc $(i^{tt}, j^{hh}) \in A^T$, where $tt > t$ and $hh < h$. In this situation, vehicle (s_2, d_2) overcomes vehicle (s_1, d_1). However, the street queue is a FIFO queue. Therefore, this maneuver must be denied. To avoid overcomes between vehicles, we use the Compatibility Set $\mathcal{H}$. In particular, we create a set H containing each pair of arcs $(n,m),(q,w) \in A'$ that are created from the temporal arc $(i^t, j^h), (i^{tt}, j^{hh}) \in A^T$ such that $tt > t$ and $hh < h$. The f_H value is $f_H(H) = 1$ in a way to avoid both the arcs that allow an overcome to be used in a feasible solution. We note that in the example proposed in Figure 2.2, considering as arc $(i,j) \in A$ the one between C and B, a new set H is created. It will be: $H = \{(N_{C1}, N_{B2}), (N_{C0}, N_{B3})\}$, and the value of $f_H(H) = 1$.

Finally, vehicles have an ideal starting time t_{sd} that could be postponed; this means that we can have multiple source nodes and, consequently, multiple destination nodes for each vehicle. They are all the temporal nodes $s^t, d^h \in N^T$ corresponding to the nodes $s, d \in N$, with $t \geq t_{sd}$ for the sources and $h \geq t_{sd} + \alpha$ for the destinations. We note that the time for the destination nodes is considered $h \geq t_{sd} + \alpha$ because the path from the source to the destination has a minimum cost of α in the case of a single arc of minimum cost. Section 2.2.1 contains more information about the α value.

To map this possibility in the graph G' we create a super-source node $\hat{s}$, a super-destination node $\hat{d}$, and a set of arcs that connect the super-source and the super-destination, respectively to the possible source nodes and destination nodes related to the temporal nodes s^t with $t \geq t_{sd}$ and d^h with $h \geq t_{sd} + \alpha$. The costs of the arcs from the super source to the various possible source nodes are defined to consider the delay of the vehicle to its starting time. For example, the cost of the arc between $\hat{s}$ and s^t where $t = t_{sd}$ will be 0, the cost of the arc between $\hat{s}$ and s^h where $h = t_{sd} + \alpha$ will be α. Instead, the costs of the arcs entering the super destination are considered equal to 0. This super-source will be the source, and the super-destination will be the destination of the (s,d) pairs on graph G'. Particularly, the vehicle set SD is transformed in $\hat{SD}$, which contains $\hat{s}$, instead of the source node s and $\hat{d}$ instead of the destination node d. Consequently the pair of super-source-super-destination is $(\hat{s}, \hat{d}) \in \hat{SD}$. We note that with this transformation no time information is provided.

Furthermore, a set H and a set H', that contain, respectively, all the new arcs leaving the super-source $\hat{s}$ and all the new arcs incoming in the super-destination $\hat{d}$, are created, and the value of the (2.36) is $f_H(H) = f_H(H') = 1$. Figure 2.3 shows an example of this process.

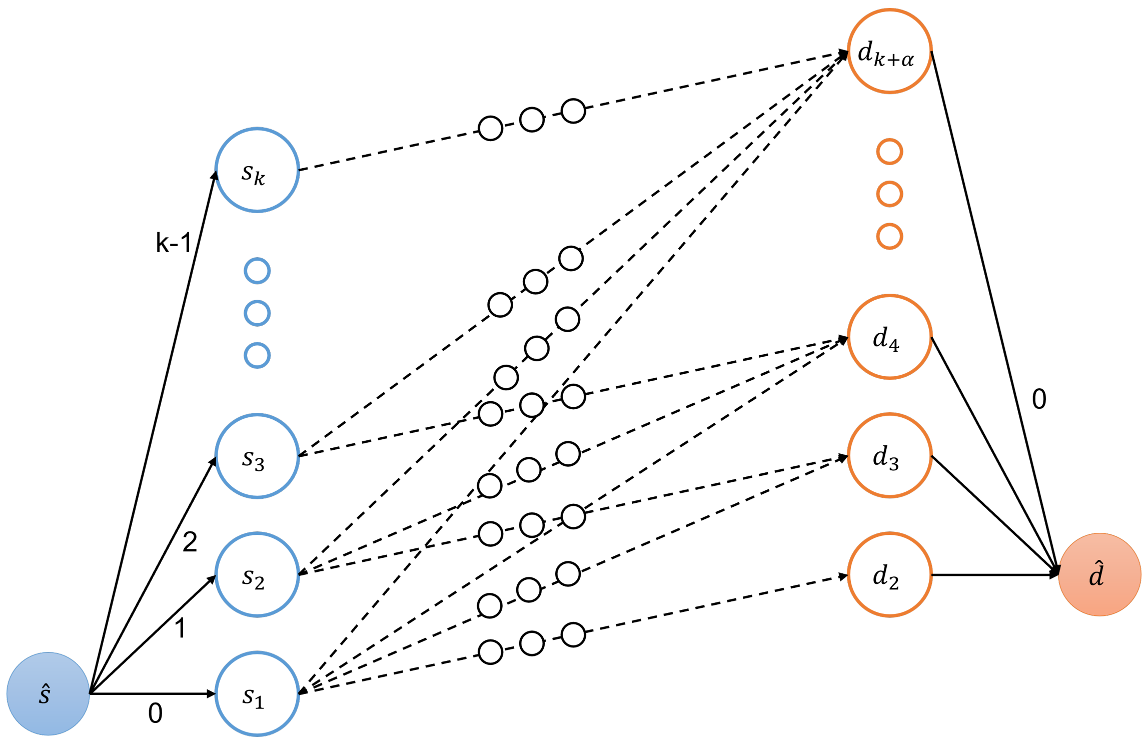


Figure 2.3: Multi-Source and Multi-Destination example

To better understand the process described in this section, we provide, in Appendix CSPP a complete example of the transformation process.

Once completed the conversion of the graph, we present a new formulation for the CSPP based on the converted graph. Given a graph $G' = (N', A')$, where N' is the set of nodes, and A' is the set of arcs. Given a set of pairs of nodes $\hat{SD}$ representing the pairs of the source-destination node to connect. Given $\mathcal{H}$ a set of a subset of arcs, namely Compatibility Set, and the function (2.36) that $\forall H \in H$ assigns an integer value $f_H(H) \leq |H|$.

Let

$$p_{i1}^{ik} = \{(i_1, ..., i_k) : (i_j, i_{j+1}) \in A', 1 \leq j \leq k-1\} \tag{2.37}$$

be a generic path, that connects i_1 and i_k; and let P be the set of all different paths between each pair of nodes. We refer to a path connecting an $(\hat{s}, \hat{d})$ pair with $p_{\hat{s}}^{\hat{d}}$.

Let

$$c : P \rightarrow \mathbb{Z}^+ \tag{2.38}$$

be a cost function, that assigns, to each path, the value given by the sum of the costs of the arcs

$$c(p_{i1}^{ik}) = \sum_{i_j, i_{j+1} \in p_{i1}^{ik}} c_{i_j i_{j+1}} \tag{2.39}$$

Finally, we define a function f that, for each pair $p_{i1}^{ik} \in P$ and $H \in \mathcal{H}$, assigns the value $f(p_{i1}^{ik}, H) = |H \bigcap \{(n,m) \in A' : (n,m) = (i_j, i_{j+1}) \in p_{i1}^{ik}\}|$. Practically, f returns the number of arcs in A' used in path p_{i1}^{ik} that belong to set H.

Formally, we can define the CSPP as the problem to find a set of paths $S \subseteq P$ such that:

$$\min_{S \subseteq P} \max_{p \in S} c(p) \tag{2.40}$$

is minimum, and:

$$\exists! \quad p = (\hat{s}, ..., \hat{d}) \in S \qquad \forall (\hat{s}, \hat{d}) \in \hat{SD} \tag{2.41}$$

$$p_1 \bigcap p_2 = \emptyset \qquad \forall p_1, p_2 \in S \tag{2.42}$$

$$\sum_{p \in S} f(p, H) \leq f_H(H) \qquad \forall H \in \mathcal{H} \tag{2.43}$$

The binary variables used in this formulation are:

$$x_{i,j}^{sd} = \begin{cases} 1, & \text{if the arc } (i,j) \in A' \text{ is used in the path } (\hat{s}, \hat{d}) \\ 0, & \text{otherwise} \end{cases}$$

The mathematical model for the CSPP, based on a multi-commodity flow formulation is the following:

$$\min \quad C_{\max} \tag{2.44}$$

$$\text{s.t.} \quad C_{\max} \geq \sum_{(i,j) \in A'} c_{ij} x_{i,j}^{sd}, \qquad \forall (\hat{s}, \hat{d}) \in \hat{SD} \tag{2.45}$$

$$\sum_{(\hat{s},j) \in A'} x_{\hat{s},j}^{sd} = 1, \qquad \forall (\hat{s}, \hat{d}) \in \hat{SD} \tag{2.46}$$

$$\sum_{(i,\hat{d}) \in A'} x_{i,\hat{d}}^{sd} = 1, \qquad \forall (\hat{s}, \hat{d}) \in \hat{SD} \tag{2.47}$$

$$\sum_{(\hat{s},\hat{d}) \in \hat{SD}} \sum_{(i,j) \in A'} x_{i,j}^{sd} \leq 1, \qquad \forall j \in N' \tag{2.48}$$

$$\sum_{(\hat{s},\hat{d}) \in \hat{SD}} \sum_{(i,j) \in A'} x_{i,j}^{sd} \leq 1, \qquad \forall i \in N' \tag{2.49}$$

$$\sum_{(i,j) \in A'} x_{i,j}^{sd} - \sum_{(k,i) \in A'} x_{k,i}^{sd} = 0, \qquad \forall (\hat{s}, \hat{d}) \in \hat{SD}, \forall i \in N' : i \neq \hat{s}, i \neq \hat{d} \tag{2.50}$$

$$\sum_{(\hat{s},\hat{d}) \in \hat{SD}} \sum_{(i,j) \in H} x_{i,j}^{sd} \leq f_H(H), \qquad \forall H \in \mathcal{H} \tag{2.51}$$

The objective function (2.44) consists of minimizing the longest path. It corresponds to minimizing the maximum completion time between all vehicles. Constraints (2.45) set the value of the max value greater than or equal to the longest path between all vehicles. Constraints (2.46) and (2.47) assure that exactly one arc has to go out from the source and has to enter the destination node for each element of $\hat{SD}$. Constraints (2.48) assure that, between all vehicles, at most one arc enters each node; while constraints (2.49) guarantee that at most one arc leaves each node considering all vehicles (these two families of constraints hold for the source and destination nodes too). Constraints (2.50) assure that for each node i used by vehicle $(\hat{s}, \hat{d}) \in \hat{SD}$ where $i \neq \hat{s}, i \neq \hat{d}$, the number of entering and exiting arcs, used by the same vehicle, are the same. Finally, constraints (2.51) assure that given a subset of arcs H, at most $f_H(H) \in \mathbb{Z}^+$ arcs can be used in a solution S.

2.4 Heuristic Approaches

The mathematical formulations proposed before can provide feasible solutions, in a reasonable amount of time, in large instances only in a few cases.

For this reason, we present two algorithms: the Delayed Shortest Path Algorithm (DSPA), and the Multi-Vehicle Shortest Path Algorithm (MVSPA). The DSPA assumes that the best possible choice for each vehicle is to follow its shortest path without considering time and collisions. Then, to avoid collisions between vehicles, the algorithm performs a collision avoidance step by modifying the arrival time of vehicles in the nodes where a collision happens. The MVSPA is a fast algorithm that builds a solution computing the shortest paths one by one starting from a specific computation order of pairs $(s, d) \in SD$. The MVSPA can be improved, in terms of solution quality by selecting an initial computation order of good quality. This algorithm also performs a failure avoidance step to avoid infeasible solutions.

2.4.1 Delayed Shortest Path Algorithm

The Delayed Shortest Path Algorithm (DSPA) is a constructive-based heuristic, which builds a possible solution assuming that the best possible choice for each vehicle (s, d) to achieve its destination is a shortest path $sp(s, d)$. This assumption is made *a priori*, regardless of the time instant and the possible collisions. Furthermore, when a path is computed it is added to the solution performing a collision-avoidance step. This step consists of verifying if the current path collides with any other path in the solution. A collision means that two paths, namely $p(s, d)$ and $p'(s', d')$, cross the node i^t. If a collision happens, the DSPA delays of α units of time the instant in which the new path p' crosses the node i^t. This results in changing the temporal node from i^t to $i^{t+\alpha}$ in path $p'(s', d')$. If this resolves the collision, the path is added to the solution, otherwise, this step is repeated by checking the collision with the following paths.

Algorithm 1 Delayed Shortest Path Algorithm

```
 1: Input: $G^T, SD$
 2: $S \leftarrow \emptyset$
 3: for all $(s,d) \in SD$ do
 4:     $p$ = Dijkstra($(s,d)$)
 5:     $qP \leftarrow S$
 6:     while $qP$ is not empty do
 7:         $p'$ = extractPath($qP$)
 8:         if $p$ collides with $p'$ then
 9:             identify the collision node $i^t$
10:             change $i^t$ with $i^{t+\alpha}$ in path $p$
11:             change nodes $j^h$ following $i^t$ with $h+\alpha$ until the end of the path $p$
12:         end if
13:         if $p$ overcomes $p'$ then
14:             identify the arc $(i^t, j^h) \in p$ of overcoming
15:             change $t$ or $h$ in path $p$ to avoid overcoming
16:         end if
17:         if $p$ has been updated then
18:             $qP \leftarrow S$
19:         end if
20:     end while
21:     if $p$ violates arc capacity then
22:         increase the starting and all the following times of vehicle $p$ by $\alpha$
23:         go to Step 5
24:     end if
25:     add $p$ in $S$
26: end for
```

In pseudo-code 1 is described the DSPA. The approach takes as input the temporal graph G^T and the set SD (Line (1)). It computes for each $(s,d) \in SD$ (Line (3)) a shortest path from node s to d (Line (4)). Then, from Line (5) up to Line (24), the collision avoidance step is performed. In particular, qP represents the set of other paths to compare with p to identify possible collisions. It is initialized using all the paths in the solution (Line (5)). The new path p is compared with all the paths already in the solution (Line (7)) by sequentially extracting the paths p' from the queue qP. In Line (8) is verified if p and p' collide in a temporal node i^t. If that happenes, the time in which p crosses node i is identified (Line (9)), and increased by α (Line (10)). For all the nodes j^h following i^t in the path p the time is increased by α to maintain the minimum cost of the other arcs equal.

Then, the algorithm verifies if the path p overcomes any other path p' already computed in Line (13). A path p overcomes a path p' if given an arc $(i,j) \in A$ such that p uses the temporal arc (i^t, j^h) and p' uses the temporal arc (i^{tt}, j^{hh}) is verified that $t > tt$ and $h < hh$ or $t < tt$ and $h > hh$. It means that both the vehicles use the same arc, but p enters the arc after p' but exits before or vice-versa. When an overcoming occurs, the algorithm identifies the arc (i^t, j^h) in Line (14). The times in which p crosses the arc (i^t, j^h) are changed to avoid the overcoming in Line (15). This is performed by modifying the value of t or h.

We note that the increasing applied to time indexes of path p influences the rest of the path until it ends. This means that if p crosses as first node i^t, and t is increased by α, all the time of the following nodes will be increased by α. This is performed to maintain consistency with the arc costs. If any update is applied to path p (Line (17)), the set of the paths to check is reset in Line (18). This is to ensure that avoiding a collision with a path p' will not cause a collision with path p''. If no collisions are found between all paths, the algorithm verifies if the new path p causes a violation of the capacity of the arc in Line (21). If yes, the starting time of the path p is increased by α in Line (22), and the process of checking if a collision occurs is repeated (Line (23)). Finally, if all the checks are satisfied the path p is added to the solution in Line (25).

Finally, we note that in the first iteration, the algorithm does not enter the while loop because the qP set is empty. After the first iteration, in Line (25), the computed path p is added in S. From the second iteration, the algorithm enters the while loop to check if a collision appends.

2.4.2 Multi Vehicle Shortest Path Algorithm

The Multi-Vehicle Shortest Path Algorithm (MVSPA) is a constructive-based heuristic that builds a possible solution by computing a shortest path for each pair $(s, d) \in SD$ in a way to avoid collisions. We remark that a collision occurs when two paths p and p' share a common temporal node i^t. It is a simple and fast heuristic that, given the computation queue qSD, with elements ordered according to a specific criterion, it starts computing a shortest path of each pair one by one, using Dijkstra's algorithm on graph G^T. The computation queue qSD consists of an ordered list of SD pairs used to select the order in which to compute the paths. More specifically, in each iteration, the approach extracts a pair (s, d) from the queue qSD and computes a shortest path that connects the source node s and the destination node d. After the computation of a path p all the Temporal Nodes and the Temporal Arcs used in p are penalised. All the arcs that, if selected for another path, will violate the constraints (2.35) are penalised as well. After that, to avoid collisions, it is checked if the computed path p contains Temporal Nodes or Temporal Arcs used in another path p'. If one or more collisions are identified, the path p' is removed from the solution, and the pair (s', d'), namely the source and the destination of p', is added at the end of the computation queue qSD. This process is repeated until a feasible solution is found or a maximum number of iterations is reached.

Algorithm 2 Multi Vehicle Shortest Path Algorithm

```
1: **Input**: $G^T$
2: $S \leftarrow \emptyset$
3: $penList \leftarrow \emptyset$
4: generate $qSD$
5: **while** $qSD$ is not empty or MaxIteration achieved **do**
6:     $(s, d)$ = extract($SD$)
7:     $p$ = Dijkstra($(s, d)$, $penList$)
8:     add $p$ in $S$
9:     update $penList$ for arcs and node of $p$
10:    **for all** arc $(i^t, j^h) \in A^T$ **do**
11:        **if** add $(i^t, j^h)$ to $S$ violate constraint (2.35) **then**
12:            penalise $(i^t, j^h) \in A^T$ in $penList$
13:        **end if**
14:    **end for**
15:    **for all** $p' \in S$ **do**
16:        **if** $p'$ collide with $p$ **then**
17:            update $penList$ for removing arcs and node of $p'$
18:            remove $p'$ from $S$
19:            push($(s', d')$, $qSD$)
20:        **end if**
21:    **end for**
22: **end while**
```

Algorithm 2 contains the pseudo-code of our constructive approach. The algorithm takes as input the graph G^T (Line (1)). It creates an empty solution S in Line (2), and the set of penalties is initialised in Line (3). The qSD computation queue is generated by sorting the SD set (Line (4)). For all pairs (s, d) (Line (5)) a shortest path p is computed on the graph G^T using Dijkstra's algorithm (Line (7)) while taking into account the penalty set. The path p is added to the solution S in Line (8) and, for all the Temporal Arcs and the Temporal Nodes of p a penalty is added to *penList* in Line (9). For all the Temporal Arcs $(i^t, j^h) \in A^T$, that if added to the solution, violate the constraints (2.35) is added a penalty (Lines (10-14)) for arc (i^t, j^h) in *penList*. The arcs in the *penList* are penalised by increasing their cost by $T_{\max}$. Further information about the $T_{\max}$ value is provided in Section 2.5.1.

Finally, from Line (15) to (21), the algorithm identifies and removes the collisions between the paths, if any. In particular, for each path p' in solution S (Line (15)) if path p' collides with path p (Line (16)) then the list of penalties is updated in Line (17), the path p' is removed from the solution in Line (18) and, the pair (s', d') is added in qSD in Line (19).

Let us now consider one possible execution of the MVSPA on an instance in which all the shortest paths of all pairs $(s, d) \in SD$, computed on G, are disjoint between them. In this case, the optimal solution contains all these paths. Thus, we can affirm that a lower bound for the problem consists of the shortest path with maximum end time between all pairs (s, d). This is obvious since the case proposed here can be considered as a relaxation of the CSPP where the collision and the capacity constraints are removed. This result is stated in Proposition 1.

Proposition 1. *Given a directed graph $G = (N, A)$, and a set of pairs of nodes of N denoted as $SD \subset N \times N$, where the $t_{sd} = 0$ $\forall(s, d) \in SD$, we can affirm that:*

$$c(sp(s, d)) \leq c(p(s, d)),\ \forall(s, d) \in SD \tag{2.52}$$

where p is a generic path connecting s and d. Thus, the longest shortest paths between all pairs $(s, d) \in SD$, namely the $LB = max(c(sp(s, d)))$, is a lower bound for the CSPP.

Proof. Let $sp(s, d)$ be a shortest path and let $p(s, d)$ be a generic path for pair $(s, d) \in SD$. Let $c(p(s, d))$ be the cost of the path connecting the pair (s, d). To prove this implication, we propose a proof based on a *reductio ad absurdum.*

Let S^* be the optimal solution for the CSPP where $max(c(p(s, d))) < LB$ for all $(s, d) \in SD, p \in S^*$. This state means that for all path $p \in S^*$ is true that $c(p(s, d)) < LB$ $\forall(s, d) \in SD$; thus there exists a path $p(s, d)$ for pair $(s, d) \in SD$ that is shorter than a shortest path $sp(s, d)$ between the same pair, which is absurd with respect to (2.52). □

The idea behind the lower bound is that, if the shortest paths in G, of all elements (s, d), are disjoint, and $t_{sd} = 0$ for all $(s, d) \in SD$, then the optimal solution cost will be the maximum end time between all the shortest paths. We note that $t_{sd} = 0$ for all $(s, d) \in SD$ means that all vehicles start at the same time and so the maximum end time between all paths is not affected by the delayed starting time.

In this ideal scenario of all disjoint shortest paths, the MVSPA can produce the optimal solution independently of the order in which the pairs $(s, d) \in SD$ are computed. Unfortunately, in the general case, the computational order of the (s, d) pairs does affect the solutions of the MVSPA.

Proposition 2. *Changing the computation order of pairs (s, d) can affect the value of the solution returned by the MVSPA.*

Proof. Consider the example shown in Figure 2.4. Using the computation order $((s_1, d_1), (s_2, d_2), (s_3, d_3))$ for MVSPA, the solution value is 5 (pair (s_1, d_1) end at time 3, pair (s_2, d_2) end at time 5, and pair (s_3, d_3) end at time 5). Using instead, the order $((s_2, d_2), (s_1, d_1), (s_3, d_3))$, the MVSPA yields a solution of value 4 (pair (s_2, d_2) ends at time 2, pair (s_1, d_1) ends at time 4, and pair (s_3, d_3) end at time 2). So, by changing the order in which the pairs (s, d) are computed it is possible to improve the value of the solution.

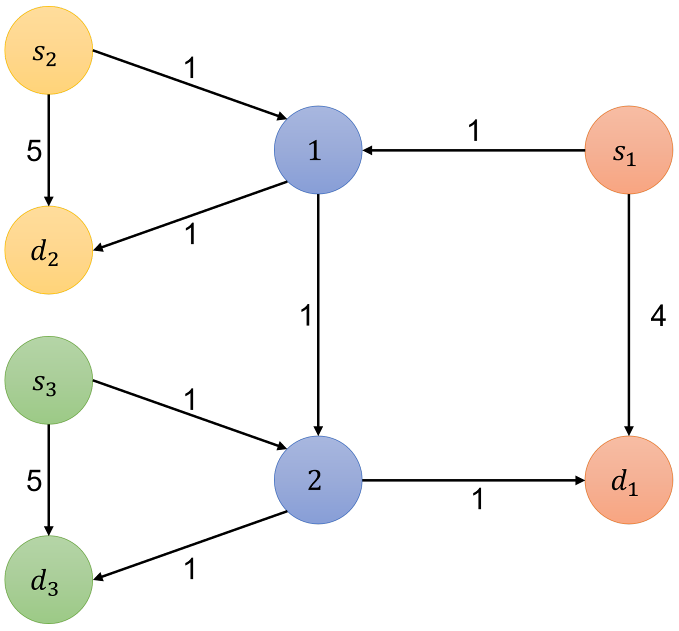


Figure 2.4: Example showing that changing the computation order of pair (s, d) can improve the solution quality.

Furthermore, we note that the solution produced with computation order ((s_2, d_2), (s_1, d_1), (s_3, d_3)) is also the optimum solution S^*. □

however, even considering the previous proposition, we cannot state that even trying all the possible permutations of (s, d) as input for the MVSPA, there is one that induces the optimal solution S^*. More formally, we can affirm that:

Proposition 3. *The MVSPA does not guarantee the optimality of the solution independently of the computation order of pairs* $(s, d) \in SD$.

Proof. In Figure 2.5, it is depicted an example showing that it does not exist a sequence of (s, d) that induces the optimal solution S^*.

In Figure 2.5, we have two pairs source-destination: (s_1, d_1) and (s_2, d_2) to connect. The shortest paths computed independently are, respectively, $sp(s_1, d_1) = \{s_1, 1, 2, d_1\}$ and $sp(s_2, d_2) = \{s_2, 1, 2, d_2\}$; the maximum end time of these two paths is equal to 3, and can be considered as a lower bound of the problem. We note that the two paths are in conflict. The MVSPA, regardless of the computation order, produces a solution of value 6 because the first pair uses the path seen before, with end time 3, and the second one instead, uses a path with end time 6 (i.e. $p(s_1, d_1) = \{s_1, 1, 2, d_1\}$ and $p(s_2, d_2) = \{s_2, d_2\}$). If, instead, we select as paths, respectively, $p(s_1, d_1) = \{s_1, 1, d_1\}$ and $p(s_2, d_2) = \{s_2, 2, d_2\}$ that are not shortest paths for the two pairs, we obtain a maximum end time of 4, that is lower than 6 and is also the optimum.

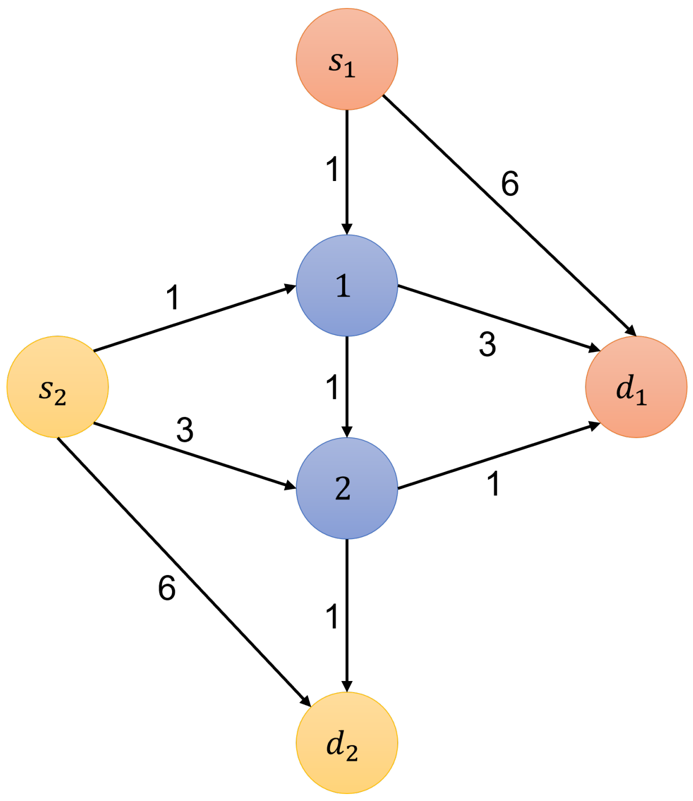


Figure 2.5: Example showing that changing the computation order of pair (s, d) does not guarantee the optimality of the solution through the MVSPA.

$\square$

Furthermore, if we remove from the graph the direct arcs (s_1, d_1) and (s_2, d_2) of cost 6, the MVSPA computes an inadmissible solution (If (s_1, d_1) is computed first the algorithm selects as path for (s_1, d_1) the path $p_1 = \{s_1, 1, 2, d_1\}$ and is not possible to compute a path for (s_2, d_2)), while the solution S^* seen before is still the optimal one.

Therefore, the effectiveness of the MVSPA can be improved by selecting a specific computation order for the pairs in SD. The idea is to maximize the number of pairs added in the solution at the beginning of the computation without collisions. To order qSD the directed input graph G^T is used to create an undirected graph $G^{IS} = (V^{IS}, E^{IS})$. The graph G^{IS} is used to generate an Independent Set (IS) ([82]) useful to compute a starting order for the set SD. For each pair $(s, d) \in SD$ a vertex v_{sd} is created in V^{IS}. Then, for each couple of vertices $v_{s_1 d_1}, v_{s_2 d_2} \in V^{IS}$ an arc is created in E^{IS} if and only if the two shortest paths $sp(s_1, d_1), sp(s_2, d_2)$ in G share a node. On graph G^{IS} the Maximum Independent Set (MIS) is computed. This set contains a subset of vertices in V^{IS} such that no two of these vertices are connected by an arc in E^{IS}. In the original graph G, the MIS is associated with the set of pairs (s, d) that have independent paths. To order the SD set, all the elements $(s, d) \in SD$ such that $v_{sd} \in MIS$ precede the elements $(s, d) \in SD$ such that $v_{sd} \notin MIS$.

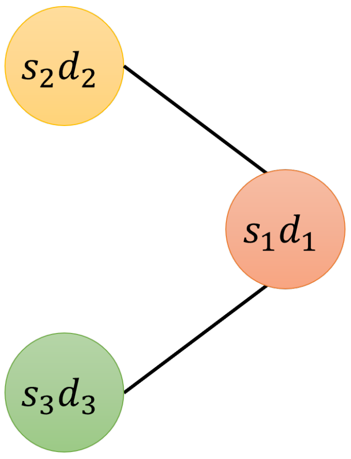


Figure 2.6: Example showing the conversion performed on a graph where the shortest paths of three vehicles have the shown collisions.

In particular, three pairs (s, d) are converted into three nodes (s_1d_1, s_2d_2, s_3d_3). Given that, for example, the shortest path between (s_1, d_1) is in conflict with both the shortest path of (s_2, d_2) and the shortest path of (s_3, d_3), in the new graph two arcs (s_1d_1, s_2d_2) and (s_1d_1, s_3d_3) are created. Finally, given that the shortest path between (s_2, d_2) is not in conflict with the shortest path of (s_3, d_3), in the new graph no arcs are created between (s_2d_2, s_3d_3). The Max IS computed on this graph is [s_2d_2, s_3d_3], so the computation order used by the variant of the MVSPA is: first the elements of the MIS, and then the remaining pairs selected randomly (i.e. [(s_2d_2, s_3d_3) $\in MIS$, (s_1d_1) $\notin MIS$]).

This variant of the approach substitutes Line (4) with the following commands: 'create a new graph G^{IS}', 'compute the MIS of G^{IS}', 'add in qSD the pair (s, d) belonging to the MIS', and 'add in qSD the pair (s, d) not belonging to the MIS randomly sorted'.

2.5 Computational Results

To test the performances of our algorithms we have generated a set of benchmark instances for the CSPP. Instances are characterized by:

- a square grid graph of size $i \times i$. We test $n \in \{3, 4, ..., 10\}$. One node is placed in each position of the grid sequentially, from the top left node to the bottom right;
- a density value, meaning the average number of arcs incident on a node. We test the following values: $\{2, 2.5, 3, 3.5, 4\}$. Consider the following example to create arcs with a density value of 3.0. For each node $i \in N$, we randomly create an average of 3.0 arcs between i and the closest nodes. It means that if i is a node in the middle of the grid, we consider its close nodes the following: the one on the top, the one at the bottom, the one on the left, the one on the

right, and also the nodes on the top-right, the top-left, the bottom-right, and the bottom-left. If i is the top left corner of the grid, we consider its close nodes only those at the bottom, on the right, and on the bottom right. Between the close nodes, we randomly select 3.0 nodes and create an arc between i and them such that the average number of arcs for each node of the graph is 3.0. In the instance creation process with a density value of 2.5, we randomly select if on a specific node are created 2 or 3 arcs such that the overall number of incident arcs in each node is 2.5. We note that an arc between i and a node on the diagonal position (e.g. top-left node) is created with a probability of 0.1. With the other nodes, the probability is 0.9. The length of the new arc is equal to the Euclidean distance between the two nodes plus a random value that can be at most 20% of the euclidean distance;

- the number of vehicles in the problem. We test the following values: $\{\sqrt{|N|}$, $\frac{1}{2}|N|$, $|N|$, $\frac{3}{2}|N|$, $2|N|\}$. We note that these values are rounded to the next integer value. We create each vehicle $(s, d) \in SD$ as follows: one node $s \in N$ is randomly chosen between all nodes $n \in N$ to be the source of the vehicle (s, d). Three different nodes $d_1, d_2, d_3 \in N : d_1 \neq d_2 \neq d_3 \neq s$ are selected as candidate destinations. Through Dijkstra's algorithm, we compute a shortest path between s and the three candidate destination. We select as the destination node for vehicle (s, d) the one with the longest shortest path. The starting time t_{sd} of vehicle (s, d) is selected equal to 0. If the source node of (s, d) is already used as the source node with starting time t, the value t_{sd} is increased by α until $t_{sd} > t$ and so the node s is free.

Note that there are $8 \times 5 \times 5 = 200$ different instance configurations. For each configuration, 10 instances with different random seeds are generated. The random seeds affect the selection of nodes to create an arc, the length of the arcs, and the selection of the source and destination nodes for vehicle creation. The random values follow the uniform distribution. In total 2000 instances are generated and tested.

Each instance is created as follows: first, a number of nodes equal to the grid size are created in a position related to the grid. For each node n_i, a number of arcs, equal to the density value, are created between n_i and its neighbor in the grid. For each arc, the euclidean distance between the nodes is computed. This value is divided by six to define the time length of the arc; this step is performed to guarantee consistency with the minimum time interval, as explained in Section 2.2.1. Finally, the capacity of the arc is set equal to its time length plus 1. This is because in an arc there could be at most one vehicle in each time instant required to cross the arc plus one vehicle that is entering the arc. After the graph has been created, for each instance the defined number of vehicles is created avoiding that two vehicles can share both source and destination nodes. For each vehicle is also defined a release time.

A summary of the characteristics of these instances is shown in Table 2.1. It shows: the grid group of instances ('Group') showing also the size of the grid of the graph G, the number of nodes ('$|N|$'), the possible number of arcs ('$|A|$'), the possible number of source-destination pairs ('$|SD|$'), and the number of conflicts of the instance ('Conflicts'). This value represents the number of conflicts between shortest paths; in other words, once computed a shortest path for each different vehicle, it counts the number of common nodes between different shortest paths. The larger the number of common nodes, the larger the probability of collision between vehicles, and so the larger the complexity of the instance.

Group	**|N|**	**|A|**	**|SD|**	**Conflicts**
G3	9	19 – 32	3, 4, 9, 13, 18	19.34
G4	16	34 – 64	4, 8, 16, 24, 32	49.02
G5	25	53 – 100	5, 12, 25, 37, 50	97.66
G6	36	76 – 144	6, 18, 36, 54, 72	167.56
G7	49	104 – 196	7, 24, 49, 73, 98	271.96
G8	64	136 – 256	8, 32, 64, 96, 128	411.12
G9	81	173 – 324	9, 40, 81, 121, 162	572.39
G10	100	212 – 400	10, 50, 100, 150, 200	780.30

Table 2.1: Summary of the instances information.

All the proposed approaches are tested on each instance, obtaining good results in terms of fitness value and computational time. The tests are performed on an Arch Linux OS, with an Intel Core i9-11950H 2.60GHz CPU and 32 GB of RAM. The approaches are implemented in Java 15 and the mathematical formulations are implemented using CPLEX 22.10. The time limit imposed for all the algorithms is 600 seconds.

2.5.1 $T_{\max}$ value

A second important value that must be defined is the maximum time value considered, namely $T_{\max}$. This value must be big enough to ensure that all vehicles can reach their destination and, at the same time should be as small as possible, in a way to reduce the size of the problem. After some preliminary experiments, we define $T_{\max}$ equal to the value returned by the heuristic approach proposed in Section 2.4.2. The $T_{\max}$ value is used also as M in all the formulations proposed.

2.5.2 Computational experiments

The following tables show the results of all the approaches proposed in Sections 2.3 and 2.4 divided into two sets: small instances and medium instances. The small set contains instances based on a grid graph of size from 3x3 to 6x6. The medium set contains instances based on a grid graph of size from 7x7 to 10x10. For the small instances, we show the results of all the mathematical formulations and the heuristic approaches. However, for medium instances, only the results obtained from the execution of the heuristic approaches are shown. This is because starting from grid size 7x7 the formulations cannot provide enough admissible solutions within the time limit. For small instances a comparison between the exact and the heuristic approaches is performed, showing the Relative Percentage Deviation (RPD) from the best solution found. For medium instances, the comparison is between the heuristic approaches and the lower bound proposed in Section 2.4.2 showing also the gap from the lower bound.

For the MVSPA we report the average results considering as initial computation queue the independent set as described in Section 2.4.2. In particular, to compute the independent set between paths, we use an ILP formulation implemented on CPLEX. Furthermore, to assess the effectiveness of the approach we also report the results of the MVSPA that uses a random starting computation queue. The algorithm that uses the ILP formulation to compute the independent set is shown in columns 'MVSPA ILP-IS'. Instead, the execution of the MVSPA that uses a random sequence as starting queue is shown in column 'MVSPA Rand'. Furthermore, in the three formulations, we give the model an initial solution provided by the DSPA. All results are the average values between the ten instances generated using the ten random seeds. The results of the three formulations are shown in column 'TFF' for the Time Flow Formulation, 'BPF' for the Bin Packing Formulation, and 'AFF' for the Alternative Flow Formulation based on the graph transformation.

Table 2.2 reports the average results of the three formulations and the two heuristic approaches on small instances. The first two columns contain the instance group and the lower bound value. Then, the objective function and the RPD to the best solution found are shown. In the smallest set, namely $G3$, the three formulations are the approaches that obtain the best results with a RPD almost equal to 0%. Also in set $G4$, the formulations outperform the heuristic approaches in terms of the quality of the solutions. Therefore, in set $G4$ the best approach is the Time-Flow Formulation. In sets $G5$ and $G6$ the only formulation able to obtain feasible solutions is the AFF. It is the approach that obtains the best results for set $G5$ with a RPD of 0.38%. On set $G6$, it lacks performances even if it is the only formulation able to obtain results in this instance set. We note that the first two formulations are comparable in terms of the quality of the solution. Furthermore, the AFF can obtain results also on bigger instances outperforming the other two formulations. The heuristic approaches obtain good solutions with a RPD between 2.0% and 3.3% for set $G3$ and $G4$. This highlights its ability to find near-optimal solutions. When the size of the instance increase, the three heuristics are in line in terms of solution quality but the best one for small instances is the MVSPA that uses the ILP formulation for the independent set to compute the initial computation order.

Group	LB	TFF		BPF		AFF		MVSPA ILP-IS		MVSPA Rand		DSPA	
		ff	RPD	ff	RPD	ff	RPD	ff	RPD	ff	RPD	ff	RPD
G3	16.63	16.98	0.00%	16.98	0.05%	16.98	0.00%	17.42	2.22%	17.39	2.27%	17.58	3.48%
G4	27.20	27.49	1.24%	27.88	2.69%	27.65	1.68%	28.07	3.29%	28.06	3.31%	28.49	4.96%
G5	37.10	-	-	-	-	37.36	0.38%	37.99	2.15%	37.98	1.96%	38.19	2.74%
G6	44.87	-	-	-	-	46.74	3.92%	46.17	0.50%	46.12	0.54%	46.42	1.15%

Table 2.2: Comparison of the performances of the three formulations, and the heuristic approaches in small instances, namely $G3 - G6$. The maximum time limit for the formulations is 600 seconds. The TFF and the BPF cannot obtain admissible solutions within the time limit on instances $G5$ and $G6$.

Table 2.3 contains the number of best solutions found by the three formulations and the heuristic approaches in all instances. As can be seen, in set $G3$, the approaches that perform better are the TFF and the AFF that obtain in all cases optimum solutions. The BPF, on set $G3$, except for a few cases can find the optimum solution. We note that the value shown in table 2.2 are average values subject to truncation error. For this reason, even if all the formulations provide the same ff values, the number of optimum of the BPF is lower. On set $G4$ the best approach is still the TFF while the BPF loses effectiveness and in several cases reaches the time limit. Therefore the AFF can obtain a higher number of optimums also in sets $G5$ and $G6$. Within the heuristic approaches, the algorithm that performs better is the MVSPA, while the one with the worst results is the DPSA. We note that the results of the MVSPA with an Independent set or random start are comparable; in small instances, the one with a random start performs better, while in the biggest cases the MVSPA with the exact computation of the independent set obtains, in average, more best solutions.

Group	TFF	BPF	AFF	MVSPA ILP-IS	MVSPA Rand	DSPA
G3	25.00	22.50	25.00	19.00	18.80	16.60
G4	24.70	14.80	24.50	15.80	17.60	15.00
G5	-	-	22.00	16.60	17.80	15.40
G6	-	-	15.40	18.00	16.90	15.50
G7	-	-	-	18.60	19.00	15.20
G8	-	-	-	16.60	17.50	15.60
G9	-	-	-	18.00	16.90	16.70
G10	-	-	-	17.80	16.90	13.50

Table 2.3: Comparison of the number of best solutions found by the three formulations, and the heuristic approaches.

For instances of medium size, the results are shown in Table 2.4. This table contains the set name in column 'Group', and the lower bound value in column 'LB'. The lower bound used here as reference is equal to the longest shortest path computed for all vehicle $(s, d) \in SD$. Then, for each algorithm are shown the objective function, the gap to the lower bound, and the RPD to the best solution found. For these instances, all the approaches obtain results comparable in terms of the gap to the lower bound and RPD. Therefore, on average, the best results are provided by the MVSPA, particularly the one with a random start. All the proposed approaches obtain results that are in line, so further tests must be performed to address the effectiveness of each algorithm. Anyway, after the validation of the approaches performed on small instances, we can affirm that all three heuristics are able of finding good solutions in a reasonable amount of time.

Group	**LB**	**MVSPA ILP-IS**			**MVSPA Rand**			**DSPA**		
		ff	**gap**	**RPD**	**ff**	**gap**	**RPD**	**ff**	**gap**	**RPD**
G7	57.69	59.18	2.59%	0.31%	59.20	2.62%	0.42%	59.80	3.65%	1.55%
G8	67.74	69.22	2.19%	0.49%	69.12	2.04%	0.32%	70.65	4.29%	2.35%
G9	75.74	77.63	2.49%	0.29%	77.72	2.61%	0.00%	78.78	4.01%	0.68%
G10	85.94	88.52	3.01%	0.47%	88.48	2.95%	0.23%	91.38	6.33%	3.38%

Table 2.4: Comparison of the performances of the heuristic approaches in medium instances.

Finally, Table 2.5 shows the computation time for all algorithms in all instances. The time is expressed in seconds. As expected, the heuristic approaches outperform the three formulations in all cases. The formulation with the best computation time is the AFF, meaning that the graph transformation process drastically impacts the formulation of the problem. On the other hand, all the heuristic approaches are comparable in terms of time required even when the size of the instances increases, but the DSPA is the approach with the lowest computation time, it ends in less than 0.04 seconds in all cases.

Group	**TFF**	**BP**	**AFF**	**MVSPA ILP-IS**	**MVSPA Rand**	**DSPA**
G3	1.04	87.41	16.98	0.01	0.00	0.00
G4	35.22	274.68	27.65	0.03	0.03	0.00
G5	-	-	37.36	0.13	0.13	0.00
G6	-	-	46.74	0.39	0.41	0.00
G7	-	-	-	1.10	1.16	0.00
G8	-	-	-	2.55	2.61	0.01
G9	-	-	-	5.01	4.98	0.01
G10	-	-	-	9.38	10.23	0.03

Table 2.5: Comparison of the computation time required by the three formulations, and the heuristic approaches in all instances. The time is expressed in seconds.

2.6 Conclusion

The Shortest Path Problem remains, over years, one of the most famous and discussed problems in literature ([62]). It consists of finding a path from a source to a destination such that the path length is minimized. The SPP can model many real-world applications, as evidenced by its variants.

In this chapter, we present the Concurrent Shortest Path Problem (CSPP), a variant of the Shortest Path Problem in which we want to define multiple paths of minimum length connecting all source-destination pairs while avoiding collisions between paths. This variant is thought to model various scenarios that go from the traffic handling of a smart city ([87], [54]), to the definition of paths of minimum length for robots and forklifts in a warehouse ([52], [90]). Generally, all applications where several vehicles have to reach their destinations without colliding between them can be modeled with the CSPP.

In this thesis, we propose three mathematical formulations for the problem: one is based on a bin packing formulation, and the others are based on the flow formulation. The former considers the time as a dimension for the problem, while the latter removes the time by applying a graph transformation process. Furthermore, to handle large-size instances, we propose two heuristic approaches, namely DSPA, and MVSPA. We defined a set of benchmark instances randomly generated. Computational results show that the proposed heuristics can produce good solutions in a reasonable amount of time. Furthermore, results show that the three formulations can be applied only in small instances.

This new variant of the SPP opens various possibilities for future research. Particularly, can be investigated the definition of new heuristic and meta-heuristic approaches that can handle big scenarios taking into account that these algorithms should be used in a real-time system. Thus, the computation time plays a crucial role, more than the solution quality. The definition of instances that change dynamically, can drastically increase the complexity of the problem. Furthermore, could be interesting to introduce a maximum completion time within which to complete all tasks. Nevertheless, further studies to address the impact of the α value on time discretization are interesting as well as the definition of new instances created from real-world graph networks,

Chapter 3

The k-Color Shortest Path Problem

3.1 Introduction

The k-Colour Shortest Path Problem (k-CSPP) is a variant of the classic Shortest Path Problem. It consists of finding a shortest path on a weighted edge-coloured graph, where the maximum number of different colours used in a feasible solution is fixed to be k. The k-CSPP has several real-world applications. Particularly, for network reliability, it addresses the problem of reducing the connection cost while improving the reliability of the network. Furthermore, for what concerns multi-modal transport systems problems, it addresses the problem of finding a path of minimum length while avoiding the usage of more than k different vehicles. Ferone et al.[37] present the first formal definition of the k-CSPP. They propose a mathematical model based on a flow formulation and a Branch and Bound (B&B) Algorithm to solve the problem. In their second work [35], Ferone et al. propose a novel Dynamic Programming (DP) algorithm that uses a path-labelling approach with an A^*-like technique as an exploration strategy. The authors also define a new set of instances with fewer colours. Testing their approaches on two test sets highlight that the DP algorithm outperforms previous approaches in terms of the number of optimum solutions and the computational time.

In this thesis, we propose a heuristic approach, namely Colour-Constrained Dijkstra Algorithm (CCDA), which can identify optimal or near-optimal solutions regardless of the size of the instances. We propose a graph reduction technique, namely the Graph Reduction Algorithm (GRA), which removes more than 90% of the nodes and edges from the input graph. Finally, using a Mixed-Integer Linear Programming (MILP) model, we present an exact approach, namely Reduced Integer Linear Programming Algorithm (RILP), that combines the two approaches described previously and guarantees optimal solutions in a reasonable running time. Several tests are performed to verify the effectiveness of the proposed approaches. The computational results indicate that the produced approaches perform well, in terms of both the solution's quality and computation times.

3.2 Mathematical Model

In the rest of this thesis, we will use $P(n_s, n_d)$ to refer to a general path from node n_s to node n_d and $l(P(n_s, n_d))$ to refer to the length of the path $P(n_s, n_d)$. The mathematical model, based on a flow formulation, is presented here to clarify the characteristics of the problem (see the model proposed by Ferone et al. in [37]). Let G be a graph $G = (N, E)$, with $|N|$ nodes and $|E|$ edges. $C : E \longrightarrow \mathbb{N}$ and $d : E \longrightarrow \mathbb{R}_0^+$ are two functions: the first is an edge-colouring function, where the colour of the edge e is defined as a positive integer $C(e), \forall e \in E$. The second is a non-negative distance function defined for each edge $e \in E$. We define $C = \bigcup_{e \in E}\{C(e)\}$ as the set containing all the colours of G. For each colour $h \in C$, we define $E_h = \{e \in E \ s.t. \ C(e) = h\}$ as the set containing all the edges with colour equal to h.

The k-CSPP consists of finding a shortest path $P^* = (n_s, ..., n_i, ..., n_d)$ from a source node n_s to a destination node n_d, with $n_s, n_d \in N$, such that the number of different colours traversed on the path does not exceed k. The flow formulation proposed by Ferone et al. considers two Boolean decision variables: the first, x_{ij}, is related to each edge of E; this variable will be equal to 1 if edge (i, j) belongs to P^*, and it will be equal to 0 otherwise. The second decision variable, y_h, is related to each possible colour such that $y_h = 1$ if at least one edge of colour h is traversed in P^*, and $y_h = 0$ otherwise. The parameter b_i defined for each $i \in N$ is:

$$b_i = \begin{cases} -1, & \text{if } i = n_s \\ +1, & \text{if } i = n_d \\ 0, & \text{otherwise.} \end{cases}$$

Thus, the problem can be formulated as follows:

$$z = min \sum_{(i,j)\in E} d(i,j)x_{ij}, \tag{3.1}$$

subject to

$$\sum_{(j,i)\in E} x_{ji} - \sum_{(i,j)\in E} x_{ij} = b_i \quad \forall i \in N, \tag{3.2}$$

$$x_{ij} \leq y_h \quad \forall h \in C, \forall (i,j) \in E_h, \tag{3.3}$$

$$\sum_{h\in C} y_h \leq k, \tag{3.4}$$

$$x_{ij} \in \{0,1\} \quad \forall (i,j) \in E, \tag{3.5}$$

$$y_h \in \{0,1\} \quad \forall h \in C. \tag{3.6}$$

The objective function (3.1) minimises the total length of the solution path P^*. The constraints in (3.2) are the flow conservation constraints. The constraints in

(3.3) enable the passage of the flow only on edges whose colour is selected. Constraint (3.4) is used to limit the maximum number of usable colours to k. Finally, constraints (3.5) and (3.6) are used to define the Boolean domain of the decision variables.

3.3 Colour-Constrained Dijkstra Algorithm

The heuristic approach proposed in this thesis, namely Colour-Constrained Dijkstra Algorithm (CCDA), is an extension of the well-known Dijkstra Algorithm [30] that, taking into account the colour function C, tries to define admissible solutions for the k-Colour Shortest Path Problem. In particular, the approach is designed to compute admissible solutions by modifying dynamically, during the computation, the costs of the edges according to their colour. During the computation, the Dijkstra Algorithm updates a shortest path from a source node n_s to all the other nodes of the graph, considering, in each step, the minimum distance from the source node to the current node n_i, plus the edge cost between n_i and the next node n_j in the path sequence. Our approach modifies the cost between n_i and n_j, adding a penalisation value, namely *penalty*, if the colour of the edge (n_i, n_j) is not used in the previously computed path $P(n_s, n_i)$. Starting from a penalisation value equal to 0, the approach iterates several times, increasing the penalisation value each time. It starts from 0 to check if the canonical shortest path is also a solution for the k-CSPP. This is because the smaller the *penalty* value, the closer the final path will be to a shortest path without penalisation, and so the better the fitness value of the solution will be. Therefore, the higher the *penalty* value, the higher the probability of avoiding new colours and thus, the higher the probability of identifying admissible solutions. The penalty values used are those included in the list *penList*. To introduce the values contained in this list, we must define the following values for each instance:

$$\begin{aligned} minEdgeCost &= min\{d(i,j)|(i,j) \in E\}, \\ maxEdgeCost &= max\{d(i,j)|(i,j) \in E\}, \\ sumEdgeCost &= \sum_{(i,j)\in E} d(i,j), \\ meanEdgeCost &= \frac{sumEdgeCost}{|E|}. \end{aligned}$$

The penalty list used is the following:

$$penList = (0, \frac{minEdgeCost}{4}, \frac{minEdgeCost}{2}, minEdgeCost, minEdgeCost * 2, \frac{meanEdgeCost}{4}, \frac{meanEdgeCost}{2}, meanEdgeCost, maxEdgeCost).$$

As can be seen in Figure 3.1, the CCDA proposed is very simple, but as described in Section 3.6, it is very effective. In particular, it takes as input the graph G, the source and destination nodes n_s and n_d, and the penalties list *penList* in Step (1). The algorithm starts iterating over the various penalty values in Step (3), trying each time, in Steps (4) and (5), to compute a shortest path with a different value of the penalty for the usage of new colours. If the current penalty value allows the Penalised Dijkstra Algorithm to find an admissible solution (Step (6)), the approach ends, returning, in Step (8), the identified solution; otherwise, it reiterates with a new penalty value (Step (7)).

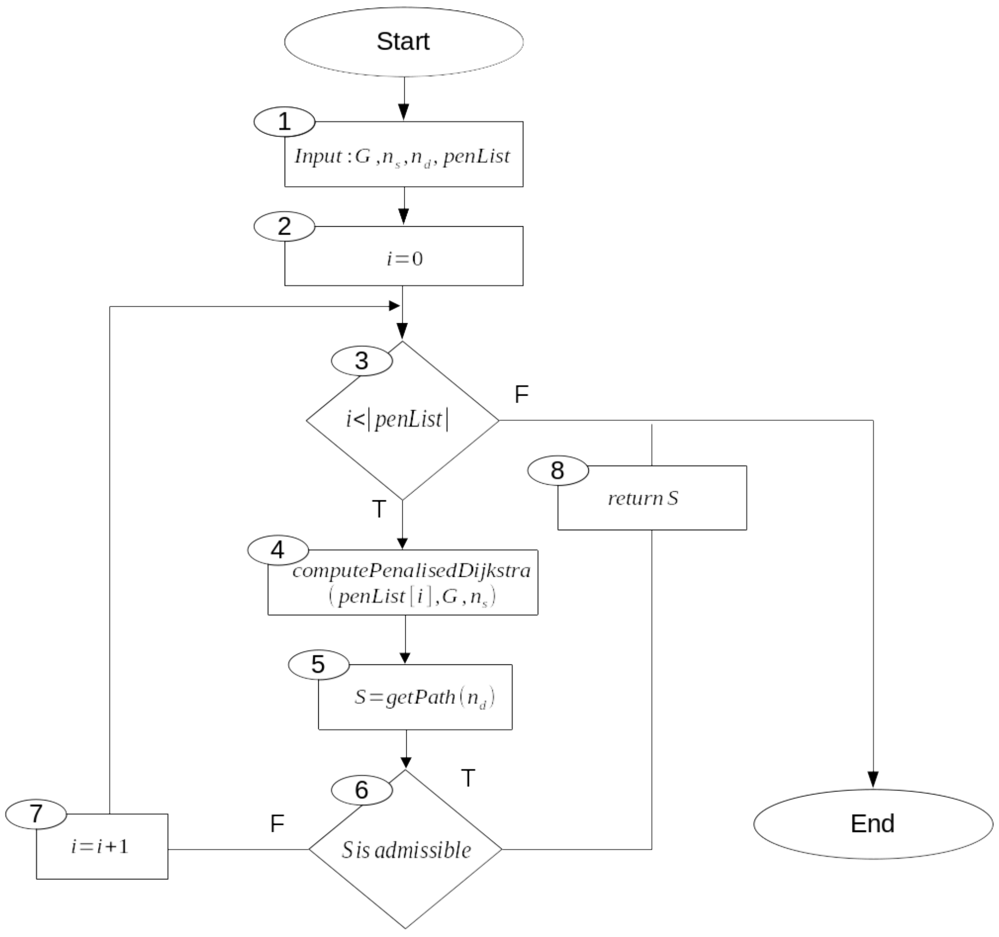


Figure 3.1: Flowchart of the Colour-Constrained Dijkstra Algorithm (CCDA), described in Section 3.3, that extends the well-known Dijkstra Algorithm to handle the edge colours and produces solutions for the k-CSPP.

A key role in this approach is played by the list of used colours. The Penalised Dijkstra Algorithm, used in Step (4), differs from the well-known standard algorithm because it takes into account the colours used in the path from the source to all the other nodes; thus, instead of memorising only the previous node and the minimum distance from the source, our approach also considers the colours used in the path. For example, when the algorithm is computing a shortest path for the node n_j, with n_i as the previous node, it checks if the colour of the edge n_i, n_j was used for the path connecting n_s and n_i: if it was, then the edge is considered with its cost; otherwise, its cost is increased by the penalty value. The penalty value is given as input to the Penalised Dijkstra Algorithm described before, and increases for each iteration of the CCDA. In this way, according to the current penalty, it is possible to produce a shortest path that uses fewer colours compared to the original algorithm, with a small increase in the computational complexity of the algorithm.

3.4 Graph Reduction Algorithm

For the k-CSPP, the complexity grows as the input size increases. Unfortunately, unlike the classic SPP for which the complexity grows polynomially, for the k-CSPP, the complexity grows exponentially [10]. This means that reducing the size of the input is crucial for lowering the computation time and creating exact approaches that are usable in large scenarios. To solve this problem, in this thesis, we propose a Graph Reduction Algorithm (GRA) that is able to reduce the size of an instance in order to apply the ILP formulation and obtain optimal solutions for many instances, even in the case of large sizes.

The GRA consists of removing from the original graph G as many nodes and edges as possible that cannot be in the optimal solution. In particular, our reduction algorithm starts by computing an upper bound (UB) for the problem using a heuristic or meta-heuristic approach; we use the Colour-Constrained Dijkstra Algorithm proposed in Section 3.3. The algorithm removes from the original graph G all the nodes and edges that, if they are forced to be in the solution, cause a fitness value higher than the upper bound. Now, given that the k-CSPP is a minimisation problem, the lower the value of the UB, the higher the number of nodes and edges removed. More formally, a heuristic solution path $P^h(n_s, n_d)$, with a length of $l(P^h(n_s, n_d))$, is computed. Then, $\forall n_i \notin P^h(n_s, n_d)$, if the shortest path length from the source node n_s to n_i plus the shortest path length from n_i to the destination node n_d is greater than $l(P^h(n_s, n_d))$, then the node n_i and all its incident edges are removed from the graph G.

A formal proof that this algorithm does not remove from the graph G nodes that can belong to the optimum solution S^*, or any solution S which cost is better than the upper bound provided, follows.

Proposition 4. *Consider a graph $G = (N, E)$, where n_s, $n_d \in N$ are respectively defined as the source and destination nodes. Consider a generic path $P^h(n_s, n_d)$ and a generic node $n_i \in N$ such that $SP(n_s, n_i)$ is a shortest path from the source node n_s to n_i, and $SP(n_i, n_d)$ is a shortest path from the node n_i to the destination node n_d.*

If

$$l(SP(n_s, n_i)) + l(SP(n_i, n_d)) > l(P^h(n_s, n_d)), \tag{3.7}$$

then

$$\forall \text{ path } P \in \{P(n_s, n_d) \text{ s.t. } n_i \in P(n_s, n_d)\}, \quad l(P) > l(P^h(n_s, n_d)). \tag{3.8}$$

Proof. In order to prove this implication, we propose a proof based on a *reductio ad absurdum.* It is possible to assume that if (3.7) is satisfied for the node n_i, then there exists a generic path $P^1(n_s, n_d)$ such that $n_i \in P^1$ and $l(P^1) \leq l(P^h(n_s, n_d))$. The path P^1 can be divided into two paths $P^2(n_s, n_i)$ and $P^3(n_i, n_d)$. Thus, we can write $l(P^h(n_s, n_d)) > l(P^1) = l(P^2(n_s, n_i)) + l(P^3(n_i, n_d)) > l(SP(n_s, n_i)) + l(SP(n_i, n_d))$, which is absurd with respect to (3.7). □

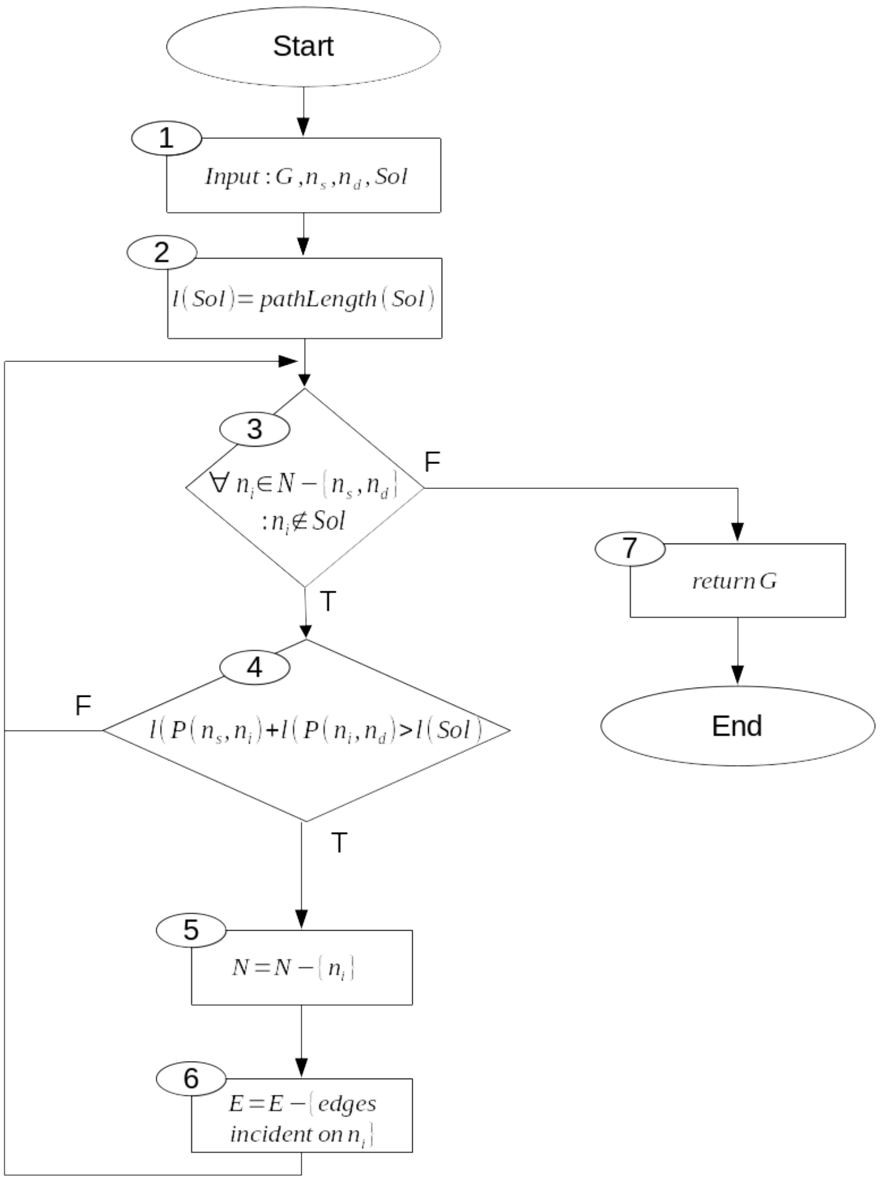
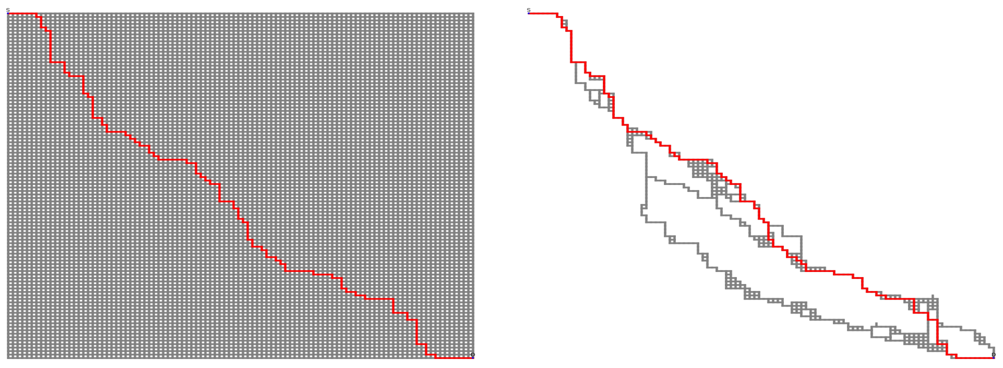

Start
1
$Input: G, n_s, n_d, Sol$
2
$l(Sol) = pathLength(Sol)$
3
$\forall n_i \in N - \{n_s, n_d\} : n_i \notin Sol$
F
T
7
$return\, G$
End
4
$l(P(n_s, n_i) + l(P(n_i, n_d) > l(Sol)$
F
T
5
$N = N - \{n_i\}$
6
$E = E - \{edges$ $incident\, on\, n_i\}$

Figure 3.2: Flowchart of the Graph Reduction Algorithm described in Section 3.4. This algorithm, depending on the quality of the upper bound provided as input, is able to drastically reduce the size of the instances, speeding up the computation time for further algorithms applied to the problem.

Figure 3.2 presents the flowchart of the reduction algorithm proposed in this section. The GRA takes as input, in Step (1), the graph G, the source and destination nodes n_s and n_d, and an initial solution Sol. It computes the length of the solution as the upper bound for the problem in Step (2). In Step (3), for each node $n_i \in N$ such that $n_i \notin Sol$, $n_i \neq n_s$ and $n_i \neq n_d$, if the length of the shortest path between n_s and n_i, namely $l(P(n_s, n_i))$, plus the length of the shortest path between n_i and n_d, namely $l(P(n_i, n_d))$, is greater than $l(Sol)$ (Step (4)), then the node n_i is removed from the set of nodes N (Step (5)), and all the edges incident on n_i are also removed from E in Step (6). In Step (7), the algorithm returns the graph without the nodes and edges that are removed during this process.

Figure 3.3 shows an example of the results produced by the reduction algorithm. Figure 3.3a shows an initial graph with a heuristic solution highlighted in red, and Figure 3.3b shows the reduced graph with the optimum solution highlighted in red. In this figure, the instance Grid_100x100_5940_27000 is represented, here the graph is reduced by more than 90% with respect to the original size of the graph, and none of the nodes and edges of the optimal solution are removed in the reduction process.



(a) Heuristic solution on G (b) Optimum solution on reduced G

Figure 3.3: Comparison between heuristic and optimum solutions on the original graph and the graph reduced using the approach proposed in Section 3.4. As can be seen, the reduction process removes more than the 90% of the graph without affecting the optimal solution.

Sometimes, after the application of GRA, a node n_i can be identified in the reduced graph that, if it is removed, causes a disconnection of the graph and the creation of two connected components. This scenario occurred quite often during our tests, as can be seen from Figure 3.3b. This scenario can be considered to improve the performance of some solution approaches. For example, an idea to investigate in future research could be the definition of a two-stage algorithm that firstly, computes in parallel several paths as partial solutions using these nodes that cause disconnection as the source or the destination nodes. Lastly, in a second phase, it could try to combine the partial solution by avoiding inadmissible scenarios. Any approach used to compute the path between the source node n_s and the destination node n_d can be applied twice in parallel, first from n_s to n_i and second from n_i to n_d. This parallelisation can drastically reduce the computation time. However, in the k-CSPP, this idea cannot be applied. This is because the colour variables must be considered for the global path and not only in the sub-graph. This approach can cause the creation of an inadmissible path that uses more than k colours. Despite this, it may be interesting to consider this scenario in further research.

3.5 Reduced ILP

The purpose of the exact approach proposed in this thesis is to drastically reduce the computational time compared to the other exact approaches proposed in the literature. To achieve this objective, we combine the effectiveness of the ILP formulation and the speediness of our heuristic algorithm. In order to compute an optimal solution, we propose the Reduced ILP (RILP) algorithm. Our algorithm starts by computing an admissible solution using the heuristic approach described in Section 3.3. If this solution is optimal, that is, its name is the same as the shortest path from n_s to n_d, the algorithm stops and returns the identified optimal solution. If the heuristic solution is not optimal, it can be used as an upper bound for the reduction algorithm proposed in Section 3.4. We use the GRA to remove, from the graph G, all the nodes and edges that are useless in the discovery of the optimal solution. Finally, on the reduced graph, our approach executes the ILP formulation to identify the optimal solution.

Figure 3.4 shows the flowchart of the RILP approach. The technique takes as input the graph G, the source node n_s and the destination node n_d, and the penalty list *penList* (Step (1)). Then, using the proposed heuristic algorithm, a solution *Sol*, considered as the upper bound for the algorithm, is computed in Step (2). A shortest path on graph G is computed in Step (3), and its length is compared with the upper bound solution in Step (4). If the length of the heuristic solution is equal to the length of the shortest path, then the algorithm stops, and the obtained solution is returned (Step (9)). Otherwise, the GRA is applied on graph G in Step (5). After the reduction phase has removed from the graph all the useless nodes and edges, the ILP formulation is applied in Step (6) to obtain the optimal solution. Finally, the solution obtained in Step (7) is returned in Step (8).

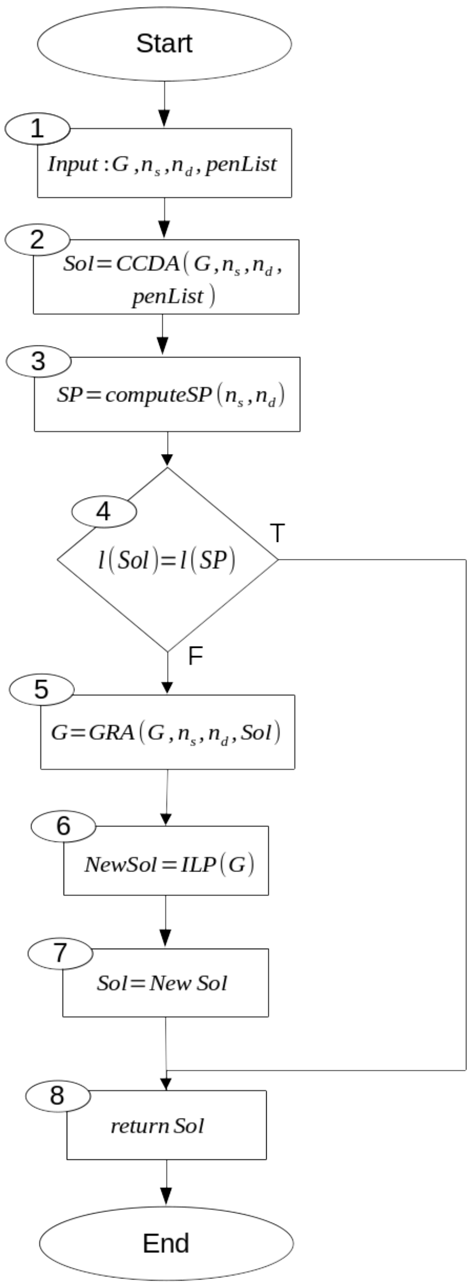

Start
1
$Input: G, n_s, n_d, penList$
2
$Sol = CCDA(G, n_s, n_d, penList)$
3
$SP = computeSP(n_s, n_d)$
4
$l(Sol) = l(SP)$
T
F
5
$G = GRA(G, n_s, n_d, Sol)$
6
$NewSol = ILP(G)$
7
$Sol = New\ Sol$
8
$return\ Sol$
End

Figure 3.4: Flowchart of the exact approach described in Section 3.5. The algorithm, exploiting the properties of the Colour-Constrained Dijkstra Algorithm and the Graph Reduction Algorithm, guarantees optimal solutions in a reasonable amount of time.

3.6 Results

To investigate the effectiveness of the proposed approaches, in this section, we show and discuss the results of our computational experiments. To validate the solutions produced by the approaches developed in this thesis, we have compared our solutions with the best ones from the literature. We use the two sets of benchmark instances proposed by Ferone et al. in [37] and [35]. These sets of instances, namely set A (Tables 3.1 and 3.3) and set B (Tables 3.2 and 3.4), are characterised by grid-graph-based instances and random-graph-based instances, respectively.

These four sets of instances are made up of 10 different instances for each scenario; the scenarios are identified in the 'Name' column of each table. In particular, in the first column, the name that represents the group of 10 instances is shown. Then, in order, the following attributes are reported: the number of nodes, the number of edges, the number of colours divided by the number of edges (given as a percentage), the number of colours, and finally, the maximum number of colours (k) that can be used to generate a solution. Note that in each instance, each value is an integer number even if the mean value reported is real. For all of the following tables, for each row, we provide the mean values of the ten instances.

Name	$\|N\|$	$\|E\|$	**%Colour**	$\|C\|$	**k**
A-G1	10000	39600	15%	5940	194.90
A-G2	10000	39600	20%	7920	195.40
A-G3	20000	79400	15%	11910	298.10
A-G4	20000	79400	20%	15880	298.90
A-G5	62500	249000	15%	37350	498.50
A-G6	62500	249000	20%	49800	499.60
A-G7	125000	498500	15%	74775	765.10
A-G8	125000	498500	20%	99700	766.30
A-G9	250000	998000	15%	149700	1005.10
A-G10	250000	998000	20%	199600	1005.30
A-G11	500000	1997000	15%	299550	1533.70
A-G12	500000	1997000	20%	399400	1535.00

Table 3.1: Summary of the characteristics of grid instances belonging to set A.

Name	$\lvert N\rvert$	$\lvert E\rvert$	**%Colour**	$\lvert C\rvert$	**k**
B-G1	10000	39600	1%	396	154.30
B-G2	10000	39600	2%	792	174.10
B-G3	20000	79400	1%	794	250.00
B-G4	20000	79400	2%	1588	273.70
B-G5	62500	249000	1%	2490	455.90
B-G6	62500	249000	2%	4980	480.20
B-G7	125000	498500	1%	4985	713.90
B-G8	125000	498500	2%	9970	740.40
B-G9	250000	998000	1%	9980	960.00
B-G10	250000	998000	2%	19960	985.70
B-G11	500000	1997000	1%	19970	1479.50
B-G12	500000	1997000	2%	39940	1508.70

Table 3.2: Summary of the characteristics of grid instances belonging to set B.

Name	$\lvert N\rvert$	$\lvert E\rvert$	**%Colour**	$\lvert C\rvert$	**k**
A-R1	75000	750000	15%	112500	6.10
A-R2	75000	750000	20%	150000	6.00
A-R3	75000	1125000	15%	168750	5.20
A-R4	75000	1125000	20%	225000	5.20
A-R5	75000	1500000	15%	225000	5.50
A-R6	75000	1500000	20%	300000	5.50
A-R7	100000	1000000	15%	150000	5.70
A-R8	100000	1000000	20%	200000	6.30
A-R9	100000	1500000	15%	225000	5.30
A-R10	100000	1500000	20%	300000	5.30
A-R11	100000	2000000	15%	300000	5.60
A-R12	100000	2000000	20%	400000	5.60
A-R13	125000	1250000	15%	187500	6.30
A-R14	125000	1250000	20%	250000	6.30
A-R15	125000	1875000	15%	281250	5.30
A-R16	125000	1875000	20%	375000	5.30
A-R17	125000	2500000	15%	375000	5.40
A-R18	125000	2500000	20%	500000	5.40

Table 3.3: Summary of the characteristics of random instances belonging to set A.

Name	$\|N\|$	$\|E\|$	**%Colour**	$\|C\|$	**k**
B-R1	75000	750000	1%	7500	5.60
B-R2	75000	750000	2%	15000	5.60
B-R3	75000	1125000	1%	11250	4.80
B-R4	75000	1125000	2%	22500	4.80
B-R5	75000	1500000	1%	15000	4.20
B-R6	75000	1500000	2%	30000	4.20
B-R7	100000	1000000	1%	10000	6.10
B-R8	100000	1000000	2%	20000	6.30
B-R9	100000	1500000	1%	15000	5.00
B-R10	100000	1500000	2%	30000	5.00
B-R11	100000	2000000	1%	20000	4.70
B-R12	100000	2000000	2%	40000	4.70
B-R13	125000	1250000	1%	12500	5.90
B-R14	125000	1250000	2%	25000	5.90
B-R15	125000	1875000	1%	18750	5.30
B-R16	125000	1875000	2%	37500	5.30
B-R17	125000	2500000	1%	25000	4.10
B-R18	125000	2500000	2%	50000	4.10

Table 3.4: Summary of the characteristics of random instances belonging to set B.

Information about the instances used to test our approaches is shown in Tables 3.1, 3.2, 3.3 and 3.4; these tables, for each scenario composed of 10 instances, report the average values.

As can be deduced from the tables, the main difference between sets A and B, for both grid and random instances, is related to the number of colours: in set A, the number of colours is equal to 15% or 20% of the number of edges, while in set B, the number of colours is equal to 1% or 2% of the number of edges. The main difference between grid instances and random instances, in addition to the graph structure, lies in the value of the parameter k. For the grid instances, this value increases proportionally with the size of the graph, while for the random instances, the value of the parameter k is small (lower than 10) and independent of the size of the instances.

In the remainder of this section, we show the results of our computational experiments. The tests are performed using CentOS Linux 7 with an Intel Xeon CPU E5-2650 v3 2.30GHz and 126 GB of RAM. The approaches are implemented in Java 1.8, and the mathematical formulation is implemented using CPLEX 12.10. The time limit imposed on all the algorithms is 900 seconds.

3.6.1 Comparison Results

The computational results are organised as follows. Tables 3.5, 3.6, 3.7 and 3.8 show a comparison between the exact approach proposed in this thesis, namely RILP, the mathematical formulation presented in Section 3.2, namely ILP, and the state-of-the-art Dynamic Programming approach proposed by Ferone et al. [35]. Results are reported for both the grid and random instances of sets A and B. Furthermore, Tables 3.9, 3.10, 3.11 and 3.12 show a comparison between our CCDA and the RILP approach to assess the effectiveness of our heuristic compared to the optimal solutions. Tables 3.5 and 3.6 compare the ILP formulation, RILP, and the Dynamic Programming technique. For all the approaches, these tables show the computation time (in seconds) and the number of optimum solutions identified. For ILP and RILP, the objective function is also shown. For ILP the number of feasible solutions identified is reported. For DP and RILP, the number of feasible solutions is not shown because all of the feasible solutions are also optimum. Finally, for DP, the objective function is not shown for the reason described previously. A summary row is given at the end of both tables.

As can be seen in Tables 3.5 and 3.6, for grid instances, the ILP formulation can obtain optimal solutions only in small instances; therefore, it loses effectiveness as the instance size increases. In fact, for medium and large instances, it can barely obtain admissible solutions, often reaching the time-limit value. However, our exact approach also performs better than the Dynamic Programming technique; it can identify, in all cases, the optimum solution in a small amount of time, while the DP approach is not able to identify a feasible solution for the grid instances of set A, especially the largest ones. On the grid instances of set B, even though the DP approach is able to identify the optimum solution in all cases, our approach requires, on average, less time. In particular, the time required by the DP algorithm increases quickly as the instance size increases. This demonstrates RILP's effectiveness regardless of the size of the instances. Tables 3.7 and 3.8 compare the ILP formulation, RILP, and the Dynamic Programming technique. For all the approaches, the tables show the computation time (in seconds) and the number of optimum solutions identified. All the methods can identify the optimum solutions in all cases. The first column reports the average fitness value. A summary row is given at the end of the table. For the random scenarios (Tables 3.7 and 3.8), all three approaches can always identify optimum solutions. Therefore, in these instances, the results highlight that the DP approach is the best algorithm in terms of computation time. Even though both the RILP and DP approaches are faster than the ILP, for both set A and set B, the DP approach is the fastest method; in all cases, it requires less than 2 seconds. The DP approach only required more time when it found unfeasible instances. This is because this technique has an advantage in scenarios where the number of colours k is small. It is interesting to note that RILP is more stable than the other methods; in fact, regardless of the size, kind, density, and characteristics of the instance, it is always able to identify the optimum solution in a reasonable amount of time.

Name	ILP				RILP			DP	
	ff	**time**	**#Feas**	**#Opt**	**ff**	**time**	**#Opt**	**time**	**#Opt**
A-G1	6203.40	37.66	10	10	6203.40	0.27	10	0.92	10
A-G2	6204.40	35.49	10	10	6204.40	0.24	10	60.81	9
A-G3	9336.60	232.49	10	9	9670.40	0.41	10	3.81	10
A-G4	9670.80	202.28	10	10	9670.80	0.37	10	12.69	10
A-G5	25073.56	827.00	9	-	15448.10	0.79	10	130.83	8
A-G6	25538.00	TL	8	1	15448.70	0.81	10	130.01	8
A-G7	39181.30	TL	10	-	23876.40	2.05	10	64.99	9
A-G8	40581.80	TL	10	-	23876.00	2.20	10	62.00	9
A-G9	52656.56	TL	9	-	30808.10	4.89	10	367.50	4
A-G10	54160.90	TL	10	-	30808.00	5.52	10	362.37	4
A-G11	79212.20	TL	10	-	47639.70	11.90	10	232.51	7
A-G12	82025.50	TL	10	-	47639.70	11.90	10	275.19	6
Mean		**645.73**	**9.67**	**3.33**		**3.45**	**10**	**141.97**	**7.83**

Table 3.5: Comparison of the performances of the ILP formulation, RILP, and the Dynamic Programming technique on grid instances from set A.

Name	ILP				RILP			DP	
	ff	**time**	**#Feas**	**#Opt**	**ff**	**time**	**#Opt**	**time**	**#Opt**
B-G1	6196.70	18.64	10	9	6197.50	0.16	10	0.03	10
B-G2	6200.00	32.21	10	9	6199.60	0.20	10	0.07	10
B-G3	9656.70	119.63	10	9	9665.40	0.56	10	0.11	10
B-G4	9656.60	124.20	10	9	9665.30	0.37	10	0.09	10
B-G5	-	TL	-	-	15444.40	0.86	10	0.92	10
B-G6	25779.40	TL	10	1	15444.50	0.72	10	0.79	10
B-G7	-	TL	-	-	23874.60	2.15	10	1.01	10
B-G8	40506.70	TL	10	-	23875.20	2.30	10	2.41	10
B-G9	-	TL	-	-	30806.30	4.60	10	50.67	10
B-G10	54227.70	TL	10	-	30805.80	4.51	10	18.37	10
B-G11	-	TL	-	-	47638.80	13.90	10	46.33	10
B-G12	81828.50	TL	10	-	47639.00	11.35	10	69.49	10
Mean		**630.04**	**6.67**	**3.08**		**3.47**	**10**	**15.86**	**10**

Table 3.6: Comparison of the performances of the ILP formulation, RILP, and the Dynamic Programming technique on grid instances from set B.

Name	**ff Opt**	**ILP**		**RILP**		**DP**	
		time	**#Opt**	**time**	**#Opt**	**time**	**#Opt**
A-R1	250.18	172.90	10	17.92	10	1.56	10
A-R2	**210.80**	**95.14**	**9**	**6.60**	**9**	**2.35**	**9**
A-R3	**187.80**	**131.23**	**9**	**6.97**	**9**	**60.24**	**9**
A-R4	**187.80**	**132.51**	**9**	**7.37**	**9**	**60.24**	**9**
A-R5	179.55	231.90	10	8.21	10	0.47	10
A-R6	179.55	248.22	10	7.98	10	0.49	10
A-R7	232.91	139.55	10	12.50	10	0.65	10
A-R8	**224.80**	**128.55**	**9**	**10.29**	**9**	**3.95**	**9**
A-R9	195.27	232.27	10	13.05	10	0.47	10
A-R10	195.27	259.18	10	13.15	10	0.50	10
A-R11	164.45	265.73	10	10.98	10	0.49	10
A-R12	164.45	269.23	10	10.85	10	0.50	10
A-R13	242.18	178.60	10	17.56	10	0.43	10
A-R14	242.18	208.26	10	18.08	10	0.43	10
A-R15	209.73	287.02	10	20.60	10	0.63	10
A-R16	209.73	336.75	10	20.59	10	0.64	10
A-R17	164.36	408.05	10	20.36	10	0.71	10
A-R18	164.36	453.29	10	19.30	10	0.72	10
Mean		**232.13**	**9.78**	**13.46**	**9.78**	**7.53**	**9.78**

Table 3.7: Comparison of the performances of the ILP formulation, RILP, and the Dynamic Programming technique. The methods were applied to random instances from set A. The scenarios in which there is an instance that does not allow admissible solutions for the given k value are shown in bold.

Name	ff Opt	ILP		RILP		DP	
		time	#Opt	time	#Opt	time	#Opt
B-R1	234.40	206.78	10	15.16	10	0.28	10
B-R2	226.90	270.65	10	18.00	10	0.29	10
B-R3	205.00	283.22	10	16.03	10	0.29	10
B-R4	221.78	294.64	10	21.38	10	0.29	10
B-R5	235.11	248.61	10	18.71	10	0.91	10
B-R6	239.22	190.36	10	13.21	10	1.01	10
B-R7	271.30	210.80	10	17.68	10	0.32	10
B-R8	225.10	240.70	10	12.86	10	0.31	10
B-R9	209.50	267.22	10	14.81	10	0.40	10
B-R10	205.44	290.80	10	17.86	10	0.40	10
B-R11	231.44	276.19	10	19.87	10	0.80	10
B-R12	235.44	233.95	10	19.65	10	0.84	10
B-R13	233.40	180.45	10	12.75	10	0.45	10
B-R14	229.40	241.53	10	15.89	10	0.47	10
B-R15	221.90	297.48	10	18.77	10	1.01	10
B-R16	201.56	291.08	10	16.49	10	1.16	10
B-R17	230.89	293.31	10	21.60	10	0.81	10
B-R18	252.67	241.53	10	19.13	10	0.82	10
Mean		**253.30**	**10**	**17.21**	**10**	**0.60**	**10**

Table 3.8: Comparison of the performances of the ILP formulation, RILP, and the Dynamic Programming technique. The methods were applied on random instances from set B.

A detailed analysis of the heuristic approach proposed in Section 3.3 is reported in Tables 3.9 and 3.10 for grid instances and in Tables 3.11 and 3.12 for random instances. These tables report, for the CCDA, the gap to the RILP algorithm, the computation time, and the number of optimum solutions identified. The number of admissible solutions identified is not shown because our approach can obtain a feasible solution for each instance. For reference, the fitness function value for the RILP algorithm is also reported. At the end of each table, a summary row is provided.

For grid instances (Tables 3.9 and 3.10), the CCDA provides very good results; it is able to obtain optimal or near-optimal solutions with an average gap equal to 0.05% of the optimum for set A and 0.07% of the optimum for set B in less than two seconds. The results obtained for the random instances (Tables 3.11 and 3.12) are in line with the results obtained for the grid instances. In fact, the CCDA obtains solutions with an average gap equal to 0.59% of the optimum for set A and 0.13% of the optimum for set B. Even though this gap is worse than the gap obtained for the grid instances, the number of optimum solutions obtained is considerably higher. Furthermore, our heuristic approach is always able to obtain admissible solutions regardless of the size of the graph and the number of colours.

Name	**ff Opt**	**CCDA**		
		gap	**time**	**#Opt**
A-G1	6203.40	0.04%	0.04	8
A-G2	6204.40	0.14%	0.04	7
A-G3	9670.40	0.13%	0.09	5
A-G4	9670.80	0.11%	0.09	6
A-G5	15448.10	0.03%	0.31	4
A-G6	15448.70	0.02%	0.32	5
A-G7	23876.40	0.02%	0.80	3
A-G8	23876.00	0.02%	0.80	3
A-G9	30808.10	0.04%	2.44	2
A-G10	30808.00	0.04%	2.49	3
A-G11	47639.70	0.02%	6.66	1
A-G12	47639.70	0.02%	5.67	1
Mean		**0.05%**	**1.65**	**4.00**

Table 3.9: Analysis of the results obtained by applying the CCDA heuristic to the grid instances of set A.

Name	**ff Opt**	**CCDA**		
		gap	**time**	**#Opt**
B-G1	6197.50	0.11%	0.02	3
B-G2	6199.60	0.15%	0.03	3
B-G3	9665.40	0.21%	0.06	2
B-G4	9665.30	0.13%	0.07	2
B-G5	15444.40	0.07%	0.28	-
B-G6	15444.50	0.02%	0.23	2
B-G7	23874.60	0.04%	0.64	-
B-G8	23875.20	0.05%	0.79	-
B-G9	30806.30	0.02%	1.75	-
B-G10	30805.80	0.02%	1.88	1
B-G11	47638.80	0.04%	7.00	-
B-G12	47639.00	0.03%	5.68	1
Mean		**0.07%**	**1.54**	**1.17**

Table 3.10: Analysis of the results obtained by applying the CCDA heuristic to the grid instances of set B.

Name	**ff Opt**	**CCDA**		
		gap	**time**	**#Opt**
A-R1	250.18	0.29%	5.07	9
A-R2	**210.80**	**0.00%**	**3.87**	**9**
A-R3	**187.80**	**0.00%**	**3.79**	**9**
A-R4	**187.80**	**0.00%**	**3.79**	**9**
A-R5	179.55	1.72%	4.25	7
A-R6	179.55	1.72%	4.41	7
A-R7	232.91	0.00%	7.74	10
A-R8	**224.80**	**0.44%**	**5.91**	**9**
A-R9	195.27	0.33%	7.53	7
A-R10	195.27	0.33%	7.53	7
A-R11	164.45	0.06%	4.39	9
A-R12	164.45	0.06%	4.43	9
A-R13	242.18	0.45%	11.21	7
A-R14	242.18	0.45%	11.05	7
A-R15	209.73	0.17%	12.53	9
A-R16	209.73	0.17%	12.41	9
A-R17	164.36	2.21%	10.08	8
A-R18	164.36	2.21%	10.11	8
Mean		**0.59%**	**7.23**	**8.28**

Table 3.11: Analysis of the results obtained by applying the CCDA heuristic to the random instances of set A. The scenarios in which an instance does not allow admissible solutions for the given k value are shown in bold.

Name	**ff Opt**	**CCDA**		
		gap	**time**	**#Opt**
B-R1	234.40	0.13%	9.90	10
B-R2	226.90	0.00%	10.88	10
B-R3	205.00	0.00%	10.05	10
B-R4	221.78	0.50%	13.62	10
B-R5	235.11	0.00%	12.52	10
B-R6	239.22	0.09%	7.68	10
B-R7	271.30	0.07%	10.98	9
B-R8	225.10	0.18%	7.20	9
B-R9	209.50	0.19%	8.32	9
B-R10	205.44	0.00%	11.21	9
B-R11	231.44	0.00%	11.42	10
B-R12	235.44	0.47%	10.70	10
B-R13	233.40	0.13%	7.05	8
B-R14	229.40	0.13%	10.04	8
B-R15	221.90	0.00%	11.04	9
B-R16	201.56	0.00%	10.21	8
B-R17	230.89	0.48%	13.74	10
B-R18	252.67	0.00%	13.03	10
Mean		**0.13%**	**10.53**	**9.39**

Table 3.12: Analysis of the results obtained by applying the CCDA heuristic to the random instances of set B.

After a deep analysis, we determined that the reason for the excellent performance of the DP approach on random instances is related to the k value. For the grid instances, this value goes from 147 to 1544, while for random instances, this value goes from 4 to 7. Given that the DP algorithm performs better when the k value is small, it obtains better results on random scenarios. Therefore, this also demonstrates that our algorithms, including both the exact and heuristic algorithms, are stable in terms of the quality of the results and the computation time, regardless of the size of the graph, the number of colours, and the k value.

Unfeasible Solutions

After an analysis of the performed tests, we noticed that among the random instances of set A, there are four instances for which it was not possible to identify an admissible solution. These cases are shown in bold in Tables 3.7 and 3.11. In the two papers of Ferone et al. [37], [35], if we look at random instances of set A, there are four cases for which neither optimal nor feasible solutions are identified. These instances are given below:

- 1 = Random_75000x75000_150000_27003;

- 2 = Random_75000x1125000_168750_27006;
- 3 = Random_75000x1125000_225000_27006;
- 4 = Random_100000x1000000_200000_27014.

Given that these are the only instances for which our approaches could not obtain optimal or admissible solutions, we performed further analyses to understand the reasons behind this problem. In particular, we implemented a simplified version, for one single path, of the model proposed by Yuan et al. [94]. Their model can identify the minimum number of colours that must be used to define several paths between a set of source nodes and a set of destination nodes. We simplified the proposed model for a single path and tried to use this formulation to determine the minimum number of colours required to identify a path in the given instances. We implemented the model in CPLEX, relaxing the integer constraints of the model, and then we executed it. The results, shown in Table 3.13, highlight that for each instance, at least one more colour is required to obtain a feasible solution (the values reported in the table were rounded to the next integer value). Next, we modified the instances, increasing the k colour value of 1, and executed our approaches again. For these new instances, our approaches are able to identify the optimal solution in all cases, as shown in Table 3.13.

Instance	**k**	**min k**	**ff Opt**	**ILP time**	**RILP time**	**CCDA time**
1	5	6	272	229.10	32.43	4.54
2	5	6	221	550.57	8.29	5.64
3	5	6	221	846.51	111.55	5.65
4	5	6	298	483.71	9.65	7.23

Table 3.13: Four random instances of set A that, with the original k value, do not admit solutions for the k-CSPP. The 'Instance' column refers to the instance list reported before. By increasing the k value to the value shown in column 'min k', the instances admit a feasible solution. All three approaches obtain the optimum, which is reported in column 'ff Opt'. The time required by each method is also reported.

3.6.2 Reduction Analysis

In this section, we propose two further analyses to study the GRA and determine how its application could modify the input graph. First, we analyse the size of the graph G and the percentage of nodes and edges removed after the application of the reduction approach proposed in Section 3.4. As can be seen in Tables 3.14 and 3.15 and in Figures 3.5 and 3.6, respectively, for the grid and random instances, the percentage of the graph removed goes from 95% to 99% for grid instances and from 89% to 99% for random instances. The effectiveness of this GRA is due to the effectiveness of the CCDA, which, as shown previously in this section, obtains optimal or near-optimal solutions in all cases. As can be seen, for grid instances, the larger the instance, the higher the reduction percentage; this is probably because the larger the graph, the higher the number of nodes and edges that are too far from the optimal path to be part of a good solution. Furthermore, the high quality of solutions provided by the CCDA translates into a threshold for the GRA that removes most of the graph. As shown in Figure 3.5, for grid instances, the reduction percentage increases linearly with respect to the size of the graph, while the random instances (Figure 3.6) have a more random behaviour, as expected. Therefore, for the random instances of both set A and set B, the reduction is globally very high and stable, except for some boundary cases.

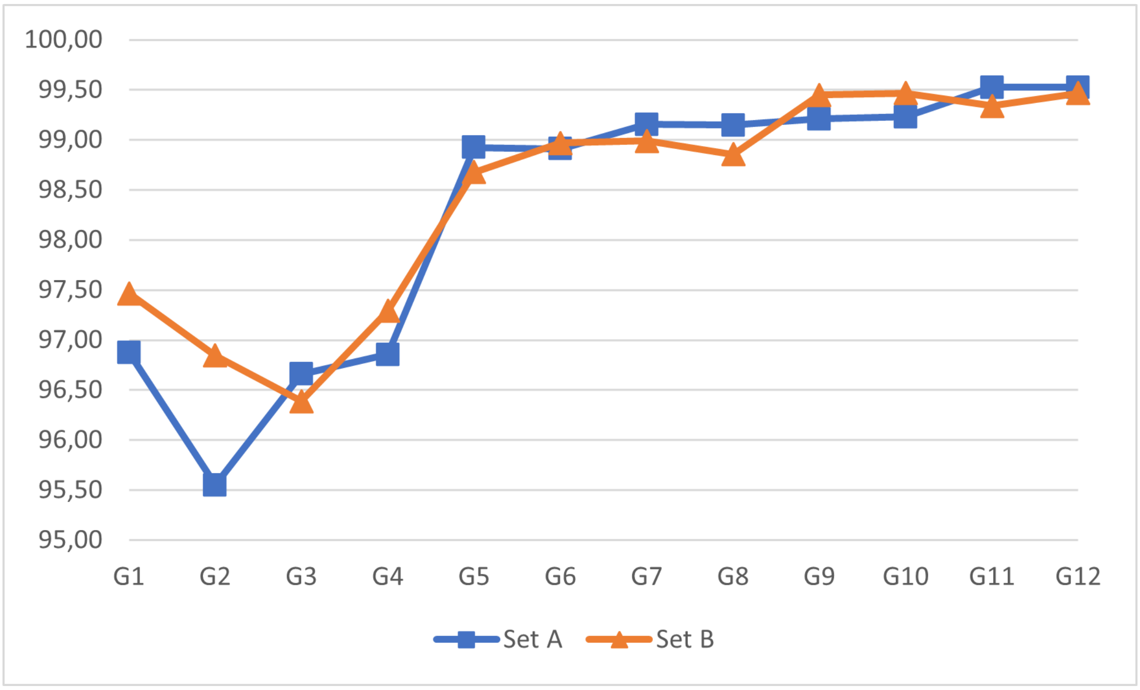


Figure 3.5: Reduction percentage for grid instances. In this figure, each value represents the mean reduction percentage of ten instances of the same size, generated with different seeds. The values for the grid instances of both set A and set B are higher than 95%. Furthermore, after an initial decrease, this percentage increases as the instance size increases, achieving values close to 99.5%.

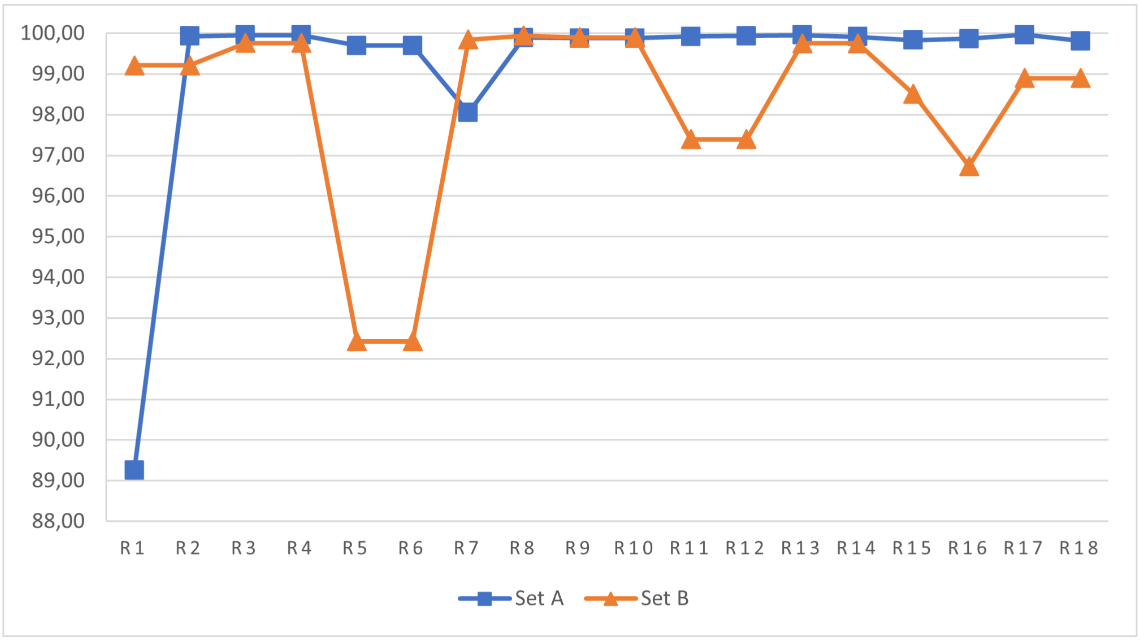


Figure 3.6: Reduction percentage for random instances. In this figure, each value represents the mean reduction percentage of ten instances of the same size, generated with different seeds. The values for the random instances of both set A and set B are higher than 89%. Furthermore, while for set A this percentage is very close to 99%, excluding the instance R1, for set B, it has a more random behaviour, as expected considering the instance type.

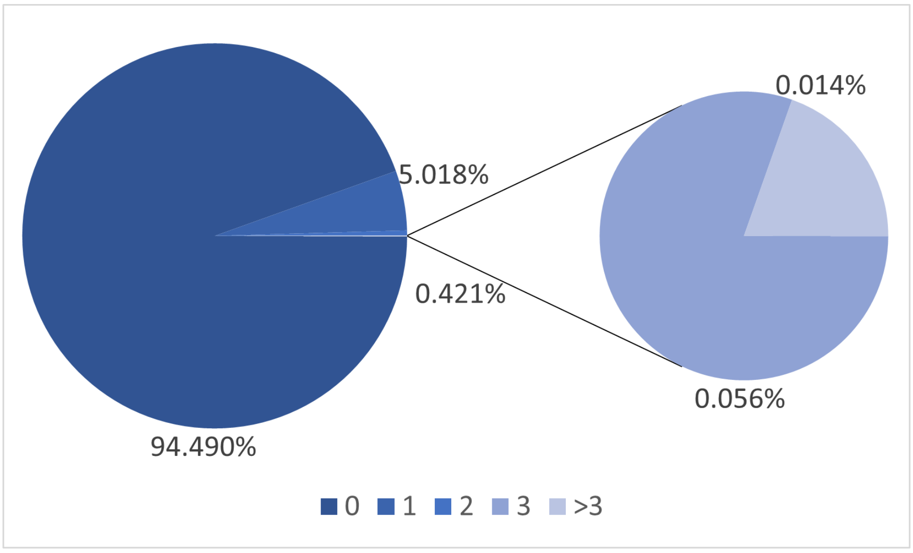
Set A	**%Reduct**	**Time**	**Set B**	**%Reduct**	**Time**
A-G1	96.88	0.13	B-G1	97.47	0.11
A-G2	95.55	0.07	B-G2	96.85	0.08
A-G3	96.66	0.10	B-G3	96.39	0.11
A-G4	96.85	0.11	B-G4	97.29	0.11
A-G5	98.93	0.32	B-G5	98.68	0.33
A-G6	98.91	0.32	B-G6	98.97	0.32
A-G7	99.15	0.91	B-G7	98.99	0.94
A-G8	99.15	0.73	B-G8	98.86	0.79
A-G9	99.21	1.79	B-G9	99.45	1.64
A-G10	99.23	1.68	B-G10	99.47	1.66
A-G11	99.53	3.89	B-G11	99.34	3.99
A-G12	99.53	3.71	B-G12	99.47	3.90
Mean	**98.30**	**1.15**	**Mean**	**98.43**	**1.16**

Table 3.14: Reduction percentages and computation times of the reduction process for grid instances of both sets. At the end of the table, a summary row with mean values is given.

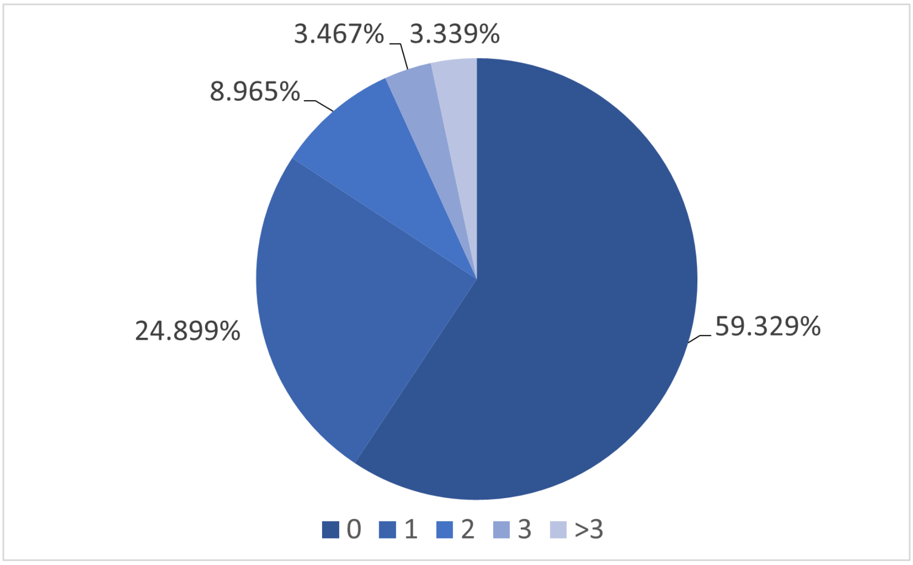
Set A	**%Reduct**	**Time**	**Set B**	**%Reduct**	**Time**
A-R1	89.25	2.67	B-R1	99.21	2.68
A-R2	99.93	2.65	B-R2	99.21	2.69
A-R3	99.95	3.17	B-R3	99.76	3.16
A-R4	99.95	3.18	B-R4	99.76	3.15
A-R5	99.69	3.48	B-R5	92.43	3.54
A-R6	99.70	3.51	B-R6	92.43	3.53
A-R7	98.05	4.35	B-R7	99.84	4.52
A-R8	99.89	4.56	B-R8	99.94	4.54
A-R9	99.88	5.18	B-R9	99.90	5.39
A-R10	99.88	5.28	B-R10	99.90	5.38
A-R11	99.92	5.80	B-R11	97.39	6.17
A-R12	99.94	5.87	B-R12	97.39	5.99
A-R13	99.96	6.58	B-R13	99.75	7.24
A-R14	99.92	6.85	B-R14	99.75	6.88
A-R15	99.83	8.22	B-R15	98.51	8.12
A-R16	99.86	8.22	B-R16	96.73	8.18
A-R17	99.96	8.59	B-R17	98.89	9.33
A-R18	99.81	8.82	B-R18	98.89	8.98
Mean	**99.19**	**5.39**	**Mean**	**98.32**	**5.53**

Table 3.15: Reduction percentages and computation times of the reduction process for random instances of both sets. At the end of the table, a summary row with mean values is given.

The second analysis performed on the GRA takes into account the number of residual edges per colour remaining in the graph after the reduction process. This analysis is interesting because it allows us to understand not only the new size of the graph but also the distribution of the edges according to their colours. Before the reduction process, the distribution of edges per colour is uniform between instances of the same set. An effective reduction process, like the one proposed in this thesis, removes most of the edges and colours; the remaining colours are only associated with a few edges. This means that on the reduced graphs, it is quite easy to identify the optimal solutions.

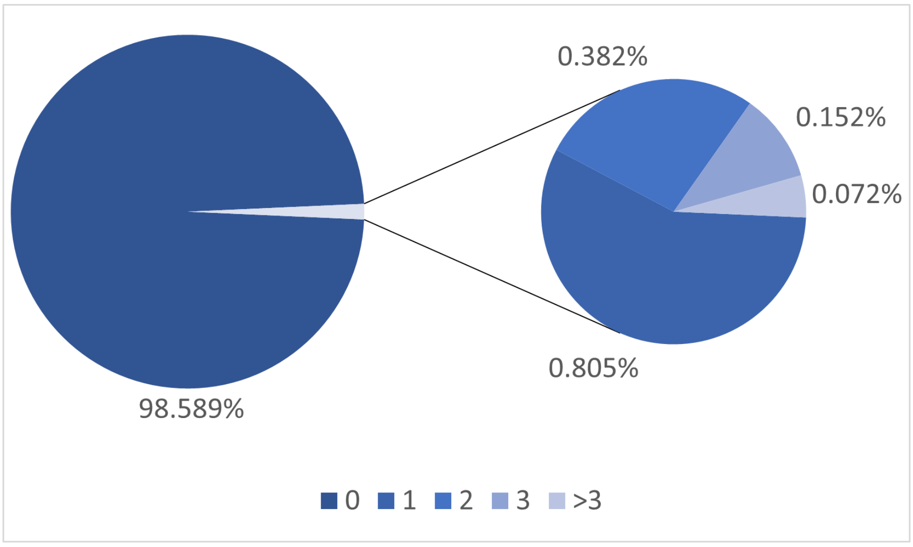


(a)

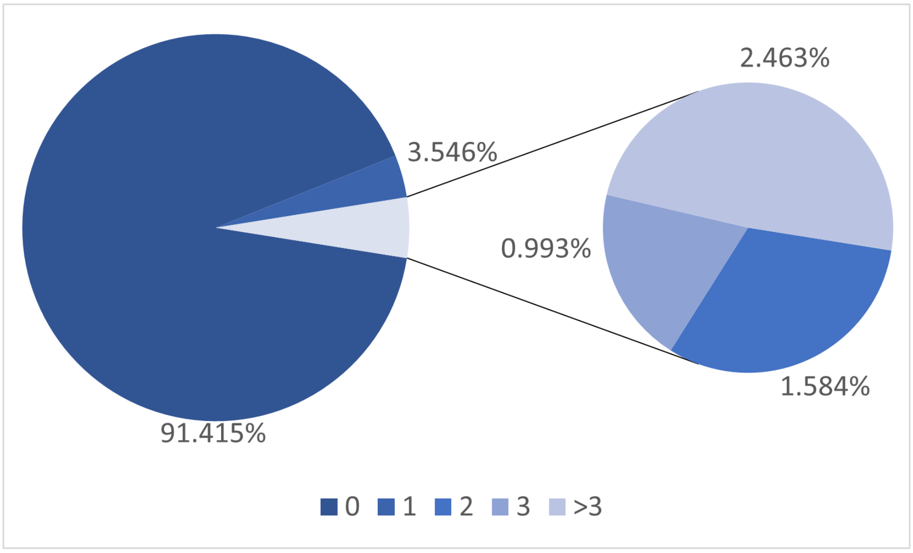


(b)

Figure 3.7: The number of residual edges per colour after the execution of the reduction process for grid instances of (a) set A and (b) set B. In particular, in (a), the chart indicates that, after the reduction, most of the colours (94.490%) have 0 edges left, while 5.018% have 1 edge left, and so on.



(a)



(b)

Figure 3.8: The number of residual edges per colour after the execution of the reduction process for random instances of (a) set A and (b) set B. In particular, in (a), the chart indicates that, after the reduction, most of the colours (98.589%) have 0 edges left, while 0.805% have 1 edge left, and so on.

As can be seen in Figure 3.7 and 3.8, most of the colours for both random and grid instances are removed because they do not have any remaining edges; however, if there are any remaining edges for a given colour, the number of edges is small and almost always lower than 3. These figures show in blue the percentage of colours removed; the other percentages represent the percentage of colours that, after the reduction, have one, two, three, or more than three edges remaining. It is clear that the percentages decrease as the number of edges per colour increases. The percentage for more than three colours is low compared to the rest of the percentages;

this means that the reduction process proposed in this thesis considerably reduces the complexity of the instances, allowing the application of exact approaches like the one proposed in Section 3.5. The main exceptions are the grid instances of set B. In some of these instances, the reduction percentage is a bit lower, and thus, the number of edges per colour is higher. Therefore, even in this case, the effectiveness of the GRA is clear. The Appendix includes tables containing detailed information about this analysis.

3.7 Conclusions

In this chapter, we analyse the k-Colour Shortest Path Problem. The k-CSPP is defined on a weighted edge-coloured graph, where the maximum number of different colours that can be used is fixed to be k. It is a variant of the Shortest Path Problem that consists of finding a shortest path for which the maximum number of different usable colours is defined *a priori*. The k-CSPP can be used to model several real-world scenarios, in particular in the field of network optimisation; it is able to improve the reliability of networks while reducing the connection cost between nodes.

To solve this problem, defined in [37], we first propose a heuristic approach, namely the Colour-Constrained Dijkstra Algorithm (CCDA), that, in a small amount of time, provides very good solutions, regardless of the size of the instances. We propose a graph reduction technique, called the GRA, that is able to remove on average more than 90% of the nodes and edges from the input graph, drastically reducing the size of the instances. Finally, we present an exact approach, called Reduced ILP (RILP), that, taking advantage of the heuristic and the Graph Reduction Algorithm, can provide optimal solutions in a reasonable amount of time, even for large instances.

Several tests are performed to verify the effectiveness of the proposed approaches. First, we compare our algorithms with state-of-the-art methods using the benchmark instances proposed in [35] and [37]. Furthermore, we perform other tests to understand the behaviour of our approaches for different kinds of instances. We analyse how the reduction process affects the instances in terms of the residual nodes and residual edges per colour.

Starting from the approach proposed in this thesis, for future research, it may be interesting to work on the definition of new heuristic or meta-heuristic approaches such as, for example, a combination of our reduction process and the DP approach. The effectiveness of the reduction technique introduced in this thesis encourages us to investigate the applicability of this technique to other path identification problems.

Chapter 4

The Close Enough Generalized Routing Problem

4.1 Introduction

The second main topic handled in this thesis concerns the definition of tours. A tour is a path that starts and ends in a depot node and connects it with all the targets of the graph. If a target is equipped with a proximity sensor (e. g. an R-FID tag), it can be served from any point within a certain distance. The Close-Enough Arc Routing Problem (CEARP) [31], and the Close-Enough Travelling Salesman Problem (CETSP) [27] are proposed to handle these new possibilities. In these two generalizations, the aim consists of finding a shortest tour that starts and ends at the depot node and intersects the neighborhood of each target once. These problems have several practical applications in the context of Unmanned Aerial Vehicles for military and civil missions like supply delivery (food, munition, etc.), geographic region monitoring, and military surveillance [12], [93], [78].

In this thesis, we propose a new generalization of the Close-Enough Travelling Salesman Problem, namely Close-Enough Generalized Routing Problem (CEGRP). It, combining the properties of the Close-Enough Travelling Salesman Problem and the Close-Enough Arc Routing Problem, can handle all the scenarios in which the movement of a UAV can be constrained or unconstrained according to its position on the graph. CEGRP models the scenarios in which a vehicle (e.g. a drone) must cover a set of targets that can be placed in a constrained location. If we consider locations such as schools, hospitals, military or residential areas, flying over these areas is usually prohibited for several reasons such as safety, public order, privacy, etc. In the CEGRP, we introduce the concept of a flight zone. We differentiate two cases: one in which the drone can fly freely, called Free Flight Zone (FFZ), and a second one in which the possibility of flight is limited to specific corridors (e.g., roads) or prohibited, called Constrained Flight Zone (CFZ).

In this thesis, we formally define the problem, and we also propose a heuristic approach, namely, the Convert and Conquer Algorithm (C&C) that performs a set of conversion steps to solve the CEGRP by reducing it into the Generalized Travelling Salesman Problem (GTSP) [68]. The GTSP is an extension of the TSP, where, given a set of targets and a partition of them into groups, we want to find a minimum length tour that includes exactly one city from each group. We propose a new set of benchmark instances to test the C&C algorithm. The results address the validity of the proposed generalization and the effectiveness of the proposed heuristic approach.

4.2 Problem Definition

To provide a generalized approach able to model scenarios that combine the CETSP and the CEARP, in this thesis, we propose a new problem that can handle both these two problems and also further applications given by a combination of the previous ones. It, namely Close-Enough Generalized Routing Problem, is formally defined on the ideas of Free Flight Zones and Constrained Flight Zones.

4.2.1 Flight Zones

An application example for the CETSP can be seen in a drone that, to reach a target, can fly directly without constraints. Unfortunately, this scenario is not realistic at all. In the real world, flying over a school, a hospital, a military zone, or a residential district, could be prohibited. This is due to many reasons related, for example, to people's safety, privacy, or public order. An example of this scenario is shown in Figure 4.1. Until today, to model this scenario are defined zones where flight is prohibited or at least constrained. If the usage of prohibited areas is selected as a modeling strategy, in some cases, serving a target with its neighborhood fully inside this kind of zone, is impossible. If the usage of constrained areas is selected as a modeling strategy, the problem could be divided into two subproblems. In particular, a CETSP is defined only for targets outside the areas with constraints, and a CEARP is defined for targets inside it. This one is a valid solution but can limit the goodness of the solutions. It also requires further computation effort to merge the solutions of the sub-problems into a feasible solution for the original instance.

In this new formulation, we define a new type of flight zone, where flight is not fully prohibited, but is constrained to predefined moving corridors. An example of this scenario is presented in Figure 4.2. This new zone, called Constrained Flight Zone, can model scenarios where the movement is constrained, using defined moving corridors or prohibited, using areas without moving corridors. The complementary part of the CFZ, namely Free Flight Zone, represents the portion of the graph where the movement is free.

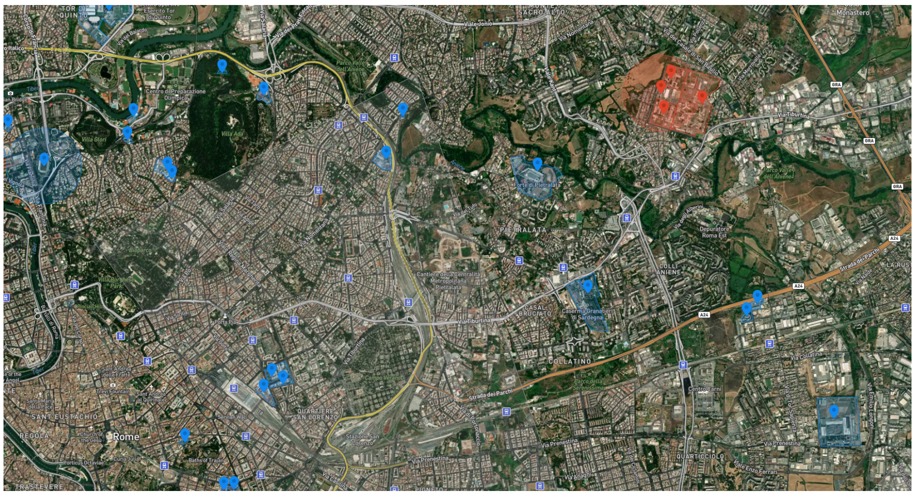


Figure 4.1: This figure shows an example of areas where flight is constrained or prohibited in Rome. The red one on the top left corner corresponds to a prohibited flight zone (the Rebibbia prison). The other blue areas represent areas where the flight is allowed but with more attention to people's safety, and privacy.

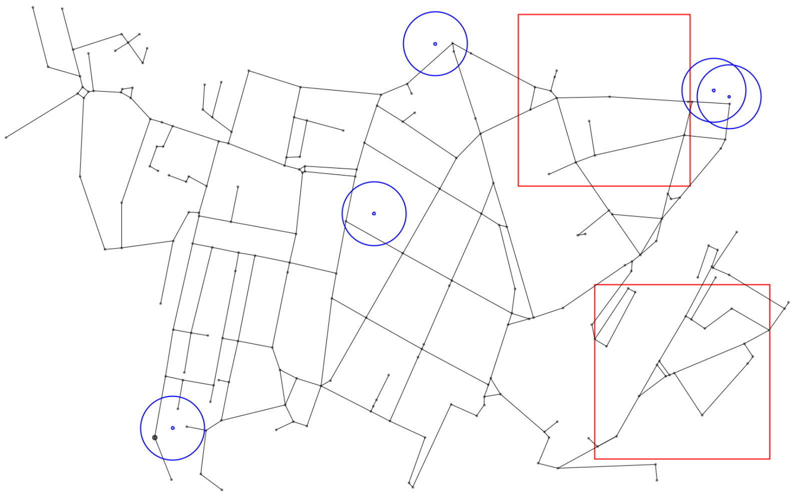


Figure 4.2: This figure shows an example instance for the CEGRP, where the graph contains different flight zones. In black are represented the edges, in blue are reported the targets with their neighborhood and, in red, are reported the Constrained Flight Zone. On the bottom left area of the image can be seen an example of a Prohibited Flight Zone. This area is represented as a CFZ without moving corridors inside.

4.2.2 The CEGRP

The CEGRP is defined on a graph $G = (N, E, T, Z)$, where N represents the set of nodes, E represents the set of edges, T represents the set of targets to serve, each target $t \in T$ has a neighborhood N_t of radius r, and finally, Z represents the set of Constrained Flight Zones described before. Let $d : E \longrightarrow \mathbb{R}_0^+$ be a non-negative distance function defined for each edge $e \in E$. The problem aims to find a path, of minimum length, that starts and ends in a depot node n_d and intersects the neighborhood of each target $t \in T$. An example instance of this problem can be seen in Figure 4.2.

Interesting to highlight is that extending the idea of CFZ to the whole graph, reduces the CEGRP into an instance of the CEARP; on the other hand, instead, if the idea of FFZ is extended to the whole graph, the CEGRP is reduced into an instance of the CETSP. Furthermore, for the CEGRP, the CETSP, and the CEARP represent, respectively, a Lower Bound and an Upper Bound for the problem. This demonstrates that the CEGRP, is a generalization and a combination of the CETSP and the CEARP. Finally, we can affirm that our new problem can model several real-world applications of both the original problems and many others that cannot be modeled using the existing formulations.

The CEGRP is an NP-Hard problem. We can motivate this assumption as follows. This problem is a generalization of the CETSP and the CEARP, which are generalizations of, respectively, TSP and ARP. Given that, as largely discussed ([26], [13]), the TSP is NP-Complete, we can say that the CEGRP is at least as complex as the CETSP and so it is NP-Hard.

4.3 Convert and Conquer Algorithm

The CEGRP is a new routing problem that consists of finding a shortest tour that intersects the neighborhood of each target, regardless of the position of the target, in a CFZ, or an FFZ. In this thesis, we propose a heuristic approach, namely Convert and Conquer Algorithm (C&C). The C&C first reduces an instance of the CEGRP into an instance of the Generalized Travelling Salesman Problem (GTSP) and then solves it using an effective algorithm for the GTSP, namely 2-Opt. We define this approach starting from the assumption that the CEGRP combines the properties of the CETSP and the CEARP; then, if we perform some transformation on the original graph, we can use existing, and effective, algorithms to provide good solutions for the original problem.

Figure 4.3 shows the flow chart of the Convert phase of the C&C Algorithm. The method takes as input a graph G in Step (1). It removes, in Steps (2-5), from G all the edges and nodes that are outside all the CFZ. This is because, as said before, these edges and nodes are useless. A new set of edges, that surround each CFZ are created in Step (6). The approach performs, in Step (7), a discretization step to select a fixed number of points as candidate positions to cover each target, and then, using these points, it creates a new set of edges to make the graph connected in Step (8). At the end of these steps, a new graph G' is created in Steps (9-11). Finally, in Step (12), the algorithm returns the new instance for the GTSP.

In the following sections, the main phases of the C&C algorithm are described in detail.

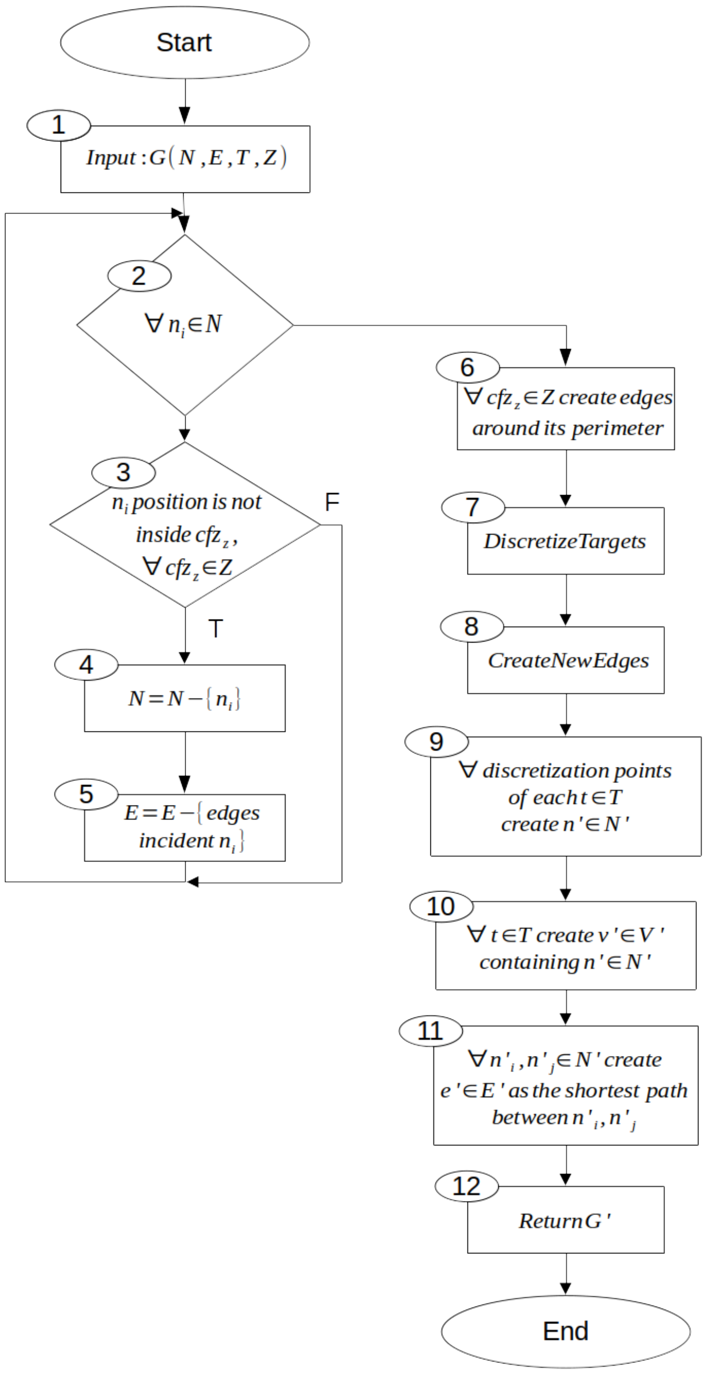

Start
1
$Input: G(N, E, T, Z)$
2
$\forall n_i \in N$
3
n_i position is not inside cfz_z, $\forall cfz_z \in Z$
F
T
4
$N = N - \{n_i\}$
5
$E = E - \{edges$ incident $n_i\}$
6
$\forall cfz_z \in Z$ create edges around its perimeter
7
DiscretizeTargets
8
CreateNewEdges
9
$\forall$ discretization points of each $t \in T$ create $n' \in N'$
10
$\forall t \in T$ create $v' \in V'$ containing $n' \in N'$
11
$\forall n'_i, n'_j \in N'$ create $e' \in E'$ as the shortest path between n'_i, n'_j
12
Return G'
End

Figure 4.3: This figure shows the flow chart of the convert phase in the Convert and Conquer Algorithm. Given an instance graph G for the CEGRP, this approach produces an instance graph G' for the GTSP.

4.3.1 Graph Reduction

The first step of the conversion process consists of a reduction of the original graph by removing the edges and the nodes inside the FFZ. This can be done because, in the FFZ, a vehicle can move without constraints, so, for the triangle inequality, the shortest path between two points is the straight path between them. Furthermore, this step also leads to a reduction of the graph complexity. In Figure 4.4 we can see an example of this step applied on the instance shown in Figure 4.2. We can see that the edges, partially inside the CFZ and partially outside, are split, removing the part outside and leaving only the part inside the CFZ. Finally, we can note that, around the CFZ, a set of new nodes and edges are created to allow the drone to fly on the perimeter of the CFZ.

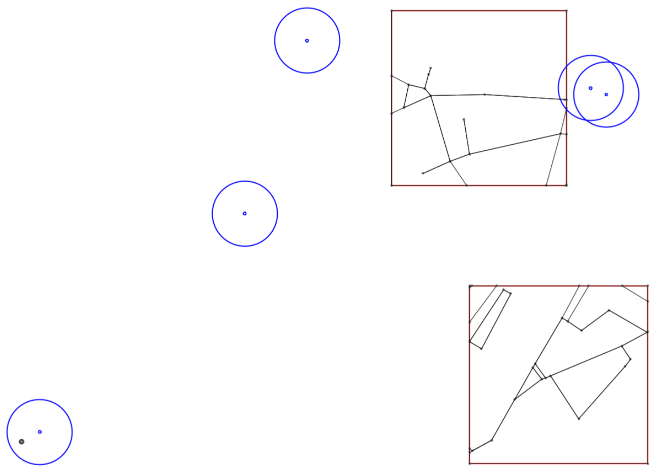


Figure 4.4: In this figure is shown an example of the application of the *ReduceGraph* Step. As can be seen, all the edges outside a CFZ are removed and new edges, that surround the CFZ are created.

4.3.2 Discretize Targets

Once removed, from the original graph, the useless edges and nodes, the algorithm performs an extremely important task, namely the target's discretization process. This step consists of defining a set of DP_t discretization points for each target. These points represent where the target neighborhood can be intersected by the vehicle. The number $|DP_t|$ is set *a priori* and is equal for all targets. In literature, several works propose effective approaches to identify these points [13].

The approach, used in this thesis to identify these points, has different behavior depending on whether the target $t \in T$ is inside a CFZ, or not. For what concern targets $t \in T$ completely inside FFZ, the discretization points are selected randomly on the circumference that bounds the neighborhood. In particular, given a target $t \in T$, and the number $|DP_t|$ of points to select, a random angle is chosen, this will be the first point $dp_t \in DP_t$; to select the others, we multiply the angle with a scalar value i, such that $i = 1, 2, ..., |DP_t|$. The product is performed in modular arithmetic. For what concerns the targets $t \in T$ inside the CFZ, the discretization points $dp_t \in DP_t$ are selected considering the intersections between the edges of the CFZ and the circumference that bounds the neighborhood of each target. Given that, in this case, the points are not generated, we select only the first $|DP_t|$ points identified by the intersections. If a target $t \in T$ has its neighborhood partially in a CFZ and partially in a FFZ, both approaches are applied. First, points by edge's intersection are selected, and then, to achieve a total of $|DP_t|$ points, the remaining points, are randomly generated on the portion of the neighborhood that is in the FFZ. For each discretization point $dp_t \in DP_t$ of each target $t \in T$ a node $n \in N$ is created.

Figure 4.5 shows an example of targets with the new set of discretization points defined and highlighted in pink. As can be seen, for each target, $|DP_t| = 4$ discretization points are selected.

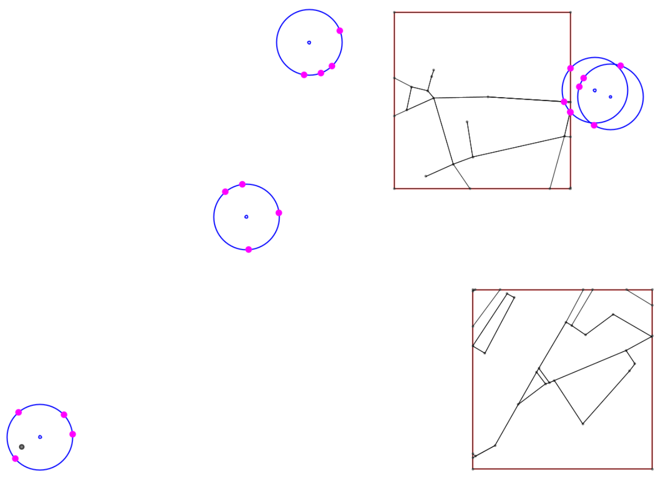


Figure 4.5: This figure shows the result of the application of the *DiscretizeTargets* Step. As can be seen, the pink dot placed on the neighborhood boundary represents the discretization points.

4.3.3 Create Edges

The third step of the conversion process consists of the creation of new edges; these new edges have to connect all the targets of the graph between each other. In particular, for every two targets $t, t' \in T$ completely contained in a FFZ, this step creates an edge between each couple nodes $n, n' \in N$ related to the discretization points $dp_t \in DP_t$ and $dp_{t'} \in DP_{t'}$. For targets completely contained in a CFZ, no edges are created at all. Finally, if a target t is partially contained in a CFZ, an edge between the nodes $n \in N$ related to the discretization points $dp_t \in DP_t$ of t and all the other nodes created starting from the discretization points for all $t' \in T$ not completely contained in a CFZ is created. We note that a new edge created connecting two targets could cross one or more CFZ. To avoid inadmissible scenarios, only edges that do not cross CFZ, are created.

Furthermore, to make the graph connected, an edge connecting the nodes $n \in N$, related to the discretization points $dp_t \in DP_t$ of all targets $t \in T$ that are not contained in a CFZ, to each node $n' \in N$ on the border of each CFZ (corner included) is created. Finally, an edge between each pair of nodes corresponding to the points on the border of two different CFZ is created, only if this new edge does not cross other CFZ. An example of the application of this step is shown in Figure 4.6. Consider for example the target in the bottom left part of the graph. The approach creates an edge between each discretization point of that target and all other discretization points of other ones. Then, a few edges between the same points and the corner of the CFZ are created. Also, the intersection points of the edges inside the CFZ and the border of the zones are connected with a new edge to the discretization points of the bottom left target. Finally, the two CFZ are connected between them, by creating a set of edges between the corner nodes and the ones on the perimeter of the zones.

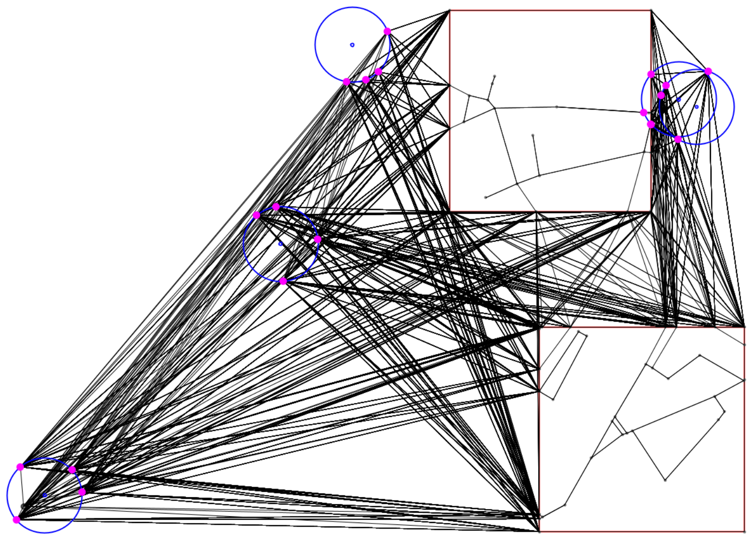


Figure 4.6: This figure shows an example of the application of the *CreateEdges* Step. As can be seen, each pair of discretization points of two different targets, each pair of CFZ, and each couple of discretization points and CFZ are connected with a new edge.

4.3.4 Graph Conversion

Once completed all the previous steps, a last further process has to be done to complete the conversion of the original graph into an instance for the GTSP. This process converts the graph $G = (N, E, T, Z)$ obtained after all the steps described before, into a new Graph $G' = (N', E', V')$, which is an instance for the GTSP. In particular, for each node $n \in N$ related to each discretization point $dp_t \in DP_t$ of each target $t \in T$, a new node $n' \in N'$ is created. A cluster $v' \in V'$ is created for each target $t \in T$, containing all the nodes $n' \in N'$ created starting from the discretization points $dp_t \in DP_t$ of that specific target $t \in T$. Finally, E' represents the new set of edges where $\forall e' \in E' : e' = \{e_s, ..., e_d\}, e_i \in E$. This means that each $e' \in E'$ is a subset of the edges of the original graph after the execution of the three steps described before. In particular, each edge e' that connects two nodes n'_i and n'_j of G' is just a shortest path between the two nodes $n_i, n_j \in N$ of the graph G. We note that in the remaining of this chapter $e' \in E'$ can be represented also as (i, j). Lastly, $c(e')$ represents the length of the edge e'. A generic tour T_{GTSP}, that is a solution for the GTSP, can be simply converted into a tour T_{CEGRP}, considering the relation between each edge $e' \in T_{GTSP} : e' \in E'$ and the set of corresponding edges $e \in E$ $(e' = \{e_s, ..., e_d\})$. Figure 4.7 presents the result of the application of the conversion process on instance shown in Figure 4.2.

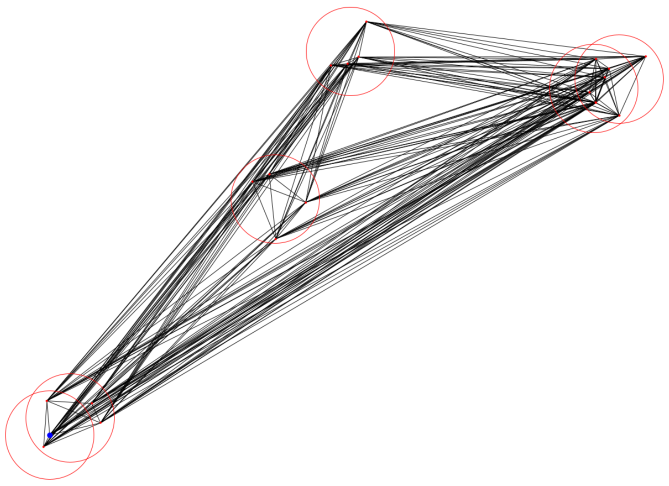


Figure 4.7: This figure presents the instance Graph for the GTSP obtained from the instance of Figure 4.2 on which the C&C conversion process is applied. The red dots represent the nodes, and the red circles group the nodes of the same cluster. The blue dot represents the depot node.

4.3.5 ILP formulation for the GTSP

The Convert & Conquer algorithm reduces an instance for the CEGRP into an instance for the GTSP. In this way, it tries to provide a solution for a problem that combines two different problems that are commonly solved using different approaches [31], [27]. After the application of the conversion process described before, we define two techniques to compute a solution for the GTSP. The former described here, consists of the ILP formulation for the GTSP.

We used the formulation proposed by Noon et al. [68]. Given $G' = (N', E', V')$, let $x_{e'} = x_{ij}$ be a binary variable where

$$x_{ij} = \begin{cases} 1, & \text{if } (i, j) \in\ E' \text{ is used in solution} \\ 0, & \text{otherwise} \end{cases}$$

Given these variables, the formulation consists of

$$\min \sum_{(i,j)\in E'} c_{ij}x_{ij} \tag{4.1}$$

$$\text{s.t.} \sum_{i\in v'} \sum_{j\notin v':(i,j)\in E'} x_{ij} = 1 \qquad \forall v' \in V' \tag{4.2}$$

$$\sum_{i\notin v'} \sum_{j\in v':(i,j)\in E'} x_{ij} = 1 \qquad \forall v' \in V' \tag{4.3}$$

$$\sum_{i\in N':(i,j)\in E'} x_{ij} - \sum_{k\in N':(j,k)\in E'} x_{jk} = 0 \qquad \forall j \in N' \tag{4.4}$$

$$\sum_{(i,j)\in E_S} x_{ij} \le |S| - 1 \qquad \forall S \subset N : 2 \le |S| \le |N| - 1 \tag{4.5}$$

$$x_{ij} = \{0,1\} \qquad \forall (i,j) \in E' \tag{4.6}$$

The objective function (4.1) consists of minimizing the cost of all the edges used in the solution. Constraints (4.2) and (4.3) impose that for each cluster of nodes $v' \in V'$ one edge enters and one edge exits the nodes of the cluster $v' \in V'$. Constraints (4.4) guarantee the balancing in the number of entering and exiting edges in each node. Finally, constraints (4.5) prevent the definition of sub-tour in the path. Let S be a possible subset of nodes. Let E_S be the edges of the graph that belong to the subset S. Constraints (4.5) impose that the edges used in a possible subset must be lower or equal to the number of nodes that belong to the subset minus one. We note that the number of constraints (4.5) grows exponentially with the size of the graph. This is because all possible subsets of nodes correspond to the power set of the nodes, which size is exponential with respect to the number of elements [59].

4.3.6 2-Opt

Due to the complexity of the problem, the ILP formulation for the GTSP cannot obtain optimum solutions within the time limit. Even if it can obtain feasible solutions in all instances, it cannot certify the optimum except for some cases. For this reason, we present a second technique based on a 2-Opt algorithm [33] to compute a path connecting the depot node n_d with one node for each cluster $v' \in V'$. A 2-Opt algorithm [33] is a simple local search approach whose main idea consists of taking a tour that crosses over itself and reordering it so that it does not. The 2-Opt algorithm is mainly proposed to compare the computation times and the quality of the solutions of the ILP formulation with a heuristic approach.

Given a set of nodes N', a set of cluster V', and a sequence of nodes S such that S contains a node $n' \in N'$ for each cluster $v' \in V'$, a complete 2-Opt local search tries every possible 2-opt of two nodes in the sequence. An exchange is performed by reversing a sub-sequence delimited by two nodes only if the length of the sequence obtained by the 2-opt is less than the actual one. After a 2-opt, the algorithm starts again to check every possible combination. Furthermore, it is possible to change the node $n' \in N'$ in which a specific cluster $v' \in V'$ is covered. The algorithm stops when all possible combinations are performed with no improvements or when a maximum number of iterations equal to $MaxIter = |V'| * 100$ are executed.

Algorithm 3 shows the pseudo-code of the 2-Opt algorithm used at the end of the Convert & Conquer algorithm. The approach starts with the input graph G' in Step (1). It initializes the sequence of nodes to cover in Step (2), and then it generates a random sequence of nodes, one for each cluster $v' \in V'$ (Step (3-5)). While the stop criteria is not reached (Step (9)), the algorithm tries to improve the initial random solution. In Step (12) and (13), for all different pairs of nodes s_i and s_j the approach generates a new sequence S^* (Step (14)) that is composed as follows. From 0 to $i-1$ the new sequence S^* is equal to the solution S_{best}, Step (15). From i to j in the new sequence S^* are inserted the nodes of S_{best} in reverse order, Step (16). Finally, from $j+1$ to the end of the sequence, S^* is equal to the solution S_{best}, Step (17). The length of the new sequence *dist* is computed in Step (18), and if this value is lower than the length of the incumbent best solution in Step (19), the 2-opt is saved, and the new solution is updated to be the best incumbent in Step (21). In Step (22), the algorithm returns to the while condition to try to find a new 2-opt that improves the solution. If no 2-opt are found, the algorithm tries to refine the solution by looking for the node that minimizes the path length within the ones of the same cluster. In Step (27), for each node s_i of the solution, and for each node n' of the same cluster v' of s_i in Step (28), a new solution that uses n' instead of s_i is generated in Steps (29) and (30). The length of the new sequence *dist* is computed in Step (31), and if this value is lower than the length of the best incumbent solution in Step (32), the new solution is updated to be the best incumbent in Step (34). In Step (35), the algorithm returns to the while condition to start the process from scratch. When the stop criteria are reached, the algorithm returns the best solution found in Step (41).

During the comparison of the results in Section 4.4, we consider two variants of the 2-Opt algorithm. The former, applies an improvement as soon as it finds it, so all 2-opt are applied. The latter applies in each iteration, the best possible 2-opt.

Algorithm 3 2-Opt

```
1: Input: $G'$
2: $S_{best} = \emptyset$
3: for all $v' \in V'$ do
4:     add in $S_{best}$ a random $n' \in v'$
5: end for
6: $i = 0$
7: $improved = true$
8: best $\leftarrow l(S_{best})$
9: while $i \leq MaxIter$ and $improved$ do
10:     $i = i + 1$
11:     $improved = false$
12:     for all $s_i \in S_{best}$ do
13:         for all $s_j \in S_{best} - s_i$ do
14:             $S^* \leftarrow \emptyset$
15:             $S^* \leftarrow S_{best}$, from $s_0$ to $s_{i-1}$
16:             $S^* \leftarrow S_{best}$, from $s_i$ to $s_j$ in reverse order
17:             $S^* \leftarrow S_{best}$, from $s_j$ to $s_{end}$
18:             dist $= l(S^*)$
19:             if dist $<$ best then
20:                 $improved = true$
21:                 $S_{best} \leftarrow S^*$
22:                 goto *Step 9*
23:             end if
24:         end for
25:     end for
26:     if not $improved$ then
27:         for all $s_i \in S_{best}$ do
28:             for all $n' \in v' : s_i \in v', n' \neq s_i$ do
29:                 $S^* \leftarrow S_{best}$
30:                 $S^*_i = n'$
31:                 dist $= l(S^*)$
32:                 if dist $<$ best then
33:                     $improved = true$
34:                     $S_{best} \leftarrow S^*$
35:                     goto *Step 9*
36:                 end if
37:             end for
38:         end for
39:     end if
40: end while
41: return $S_{best}$
```

4.4 Computational Results

This section reports the results obtained by applying the approach proposed in Section 4.3 on a set of benchmark instances defined for this problem. These instances are created starting from a practical application for the problem. It consists of defining a tour of minimum length for a drone that has to cover a set of R-FID tags in a port scenario. The R-FID tags are placed on a set of containers that have to be monitored. The drone has to cover all the targets while moving in constrained areas, namely the row of containers, and free areas (the empty spaces between them).

The instances are created starting from a grid graph. The grid size goes from 10x10 up to 100x100. Thus, the graph size goes from 100 nodes and 180 edges up to 10000 nodes and 19800 edges. For each graph, a set of targets that goes from 10 up to 100 is generated. For each target, we consider three different radii: 30, 50, and 100 meters. Finally, for each one of these scenarios, a set of CFZ of random size is defined. This number goes from 1 up to 10. The percentage of space inside the CFZ is on average 25% of the total space of the graph. An instance is created for each combination of these characteristics, and the total number of instances created is 180. A summary of the properties of the instances is reported in Table 4.1. More information about the instances is shown in the Appendix.

Grid Size	**|T|**	**Radius**	**|Z|**
10x10	10	30	1
50x50	20	50	2
100x100	50	100	3
	100		5
			10

Table 4.1: Summary of the characteristics of the benchmark instances. For each combination of each characteristic, an instance is created to assess the performance of the proposed approach.

In the remainder of this section, we show the results of our computational experiments. The tests are performed on an Arch Linux OS with an Intel Core i9-11950H 2.60GHz CPU and 32 GB of RAM. The approaches are implemented in Java 15, and the mathematical formulation for the GTSP is implemented using CPLEX 22.10. The sub-tour elimination constraints 4.5 used in the formulation proposed in Section 4.3.5 are implemented as lazy constraints using the CPLEX solver. The time limit for all the algorithms is 600 seconds.

The results proposed, consider the same conversion phase for the C&C algorithm and compare the objective function, the computation times, the gap to and the percentage of the best solution found by the 2-Opt algorithm, and the ILP formulation of the GTSP. Focusing on the 2-Opt algorithm, we report both the results for the variant that applies all the possible 2-opt and the one that applies only the best one in each iteration. Best 2-opt means that in each iteration, all possible 2-opt are computed. The one with the highest improvement of the objective function is applied to the path. The results of the ILP formulation are named ‘ILP’. The results of the 2-Opt algorithm that applies all possible 2-opt and only the best 2-opt are indicated, respectively, by ‘2-Opt AS’ and ‘2-Opt BS’.

Name	**Conversion**	**ILP**	**2-Opt AS**	**2-Opt BS**
10x10-10t	0.04	563.41	0.00	0.00
50x50-10t	0.25	564.41	0.00	0.00
100x100-10t	1.01	522.54	0.00	0.00
Mean 10t	**0.44**	**550.12**	**0.00**	**0.00**
10x10-20t	0.08	600.12	0.01	0.01
50x50-20t	1.06	600.25	0.01	0.01
100x100-20t	4.55	600.78	0.01	0.02
Mean 20t	**1.90**	**600.39**	**0.01**	**0.01**
10x10-50t	4.24	600.31	0.79	0.85
50x50-50t	12.27	600.20	0.66	1.22
100x100-50t	32.48	600.23	0.79	1.44
Mean 50t	**15.60**	**600.25**	**0.75**	**1.17**
10x10-100t	19.46	603.64	36.66	38.70
50x50-100t	93.67	605.07	35.46	60.11
100x100-100t	233.18	606.29	46.04	71.49
Mean 100t	**115.44**	**605.00**	**39.38**	**56.77**

Table 4.2: Summary of the computation times required by all the approaches. Each line contains the average value for the instances of the size and number of targets specified in the column ‘Name’. After each set of instances with the same number of targets, a summary row is shown. All the times are shown in seconds.

Table 4.2 contains the information about the computation time required by the algorithm proposed in Section 4.3. In particular, the conversion time required by the convert phase of the C&C algorithm is reported in column ‘Conversion’. Then, the times required by the ILP formulation of the GTSP, the 2-Opt algorithm that applies all possible 2-opt, and the one that applies only the best ones are shown. All the times are in seconds. As can be seen the time required by the conversion phase increase exponentially as increases the size of the graph and the number of targets. The number of targets has an important impact due to the number of discretization points used in the conversion phase. The ILP formulation achieves on average the time limit of 600 seconds in all instances except for some boundary cases. The two variants of the heuristic approach are considerably faster with respect to the ILP formulation. The fastest algorithm is the 2-Opt which applies all possible 2-opt. This is because it does not look for the best improvement of the solutions, and so, in bigger instances, it results in a reduction of the computation time of 30%.

Table 4.3 contains the average results obtained by the proposed approaches. For each algorithm the table reports the value of the objective function, the percentage gap with respect to the best solution found over all the algorithms, and the percentage of the best solutions found. In Table 4.3, each line contains the average value for the instances of the size and number of targets specified in the 'Name' column (e. g. the first line contains the average value between all instances with graph size 10x10 and 10 targets). After each set of instances with the same number of targets, a summary row is shown. All the algorithms can obtain at least admissible solutions in all instances except for the ILP formulation that in instances with 100 targets reaches the time limit without obtaining admissible solutions.

Name	**ILP**			**2-Opt AS**			**2-Opt BS**		
	ff	**gap**	**% best**	**ff**	**gap**	**% best**	**ff**	**gap**	**% best**
10x10 10t	2592.18	0.00%	100.00%	2819.99	8.76%	0.00%	2772.97	6.73%	6.67%
50x50 10t	14554.30	0.00%	100.00%	16487.86	13.83%	0.00%	15513.49	6.93%	13.33%
100x100 10t	29636.97	0.00%	93.33%	31524.50	6.49%	6.67%	32081.10	8.05%	6.67%
Mean	**15594.48**	**0.00%**	**97.78%**	**16944.12**	**9.69%**	**2.22%**	**16789.19**	**7.24%**	**8.89%**
10x10 20t	2764.63	0.01%	80.00%	3663.90	27.00%	6.67%	3866.49	33.22%	13.33%
50x50 20t	16111.24	0.08%	86.67%	20301.55	34.04%	6.67%	19500.24	29.14%	6.67%
100x100 20t	33312.51	0.01%	93.33%	43366.76	35.79%	6.67%	38538.27	21.55%	0.00%
Mean	**18076.66**	**0.04%**	**86.67%**	**23548.74**	**32.28%**	**6.67%**	**21743.40**	**27.97%**	**6.67%**
10x10 50t	4344.64	7.20%	93.33%	9520.53	57.15%	6.67%	10050.73	73.59%	0.00%
50x50 50t	31353.66	14.17%	80.00%	30751.45	35.56%	13.33%	33761.83	45.11%	6.67%
100x100 50t	58278.24	3.22%	73.33%	65777.92	29.93%	13.33%	67440.29	36.72%	13.33%
Mean	**31325.52**	**8.20%**	**82.22%**	**36929.02**	**40.88%**	**11.11%**	**39386.19**	**51.80%**	**6.67%**
10x10 100t	-	-	-	18184.90	2.99%	73.33%	21474.72	15.06%	26.67%
50x50 100t	-	-	-	44555.28	0.45%	86.67%	50060.92	14.84%	13.33%
100x100 100t	-	-	-	95054.25	0.71%	86.67%	107915.56	15.51%	13.33%
Mean	**-**	**-%**	**-%**	**56153.01**	**1.38%**	**82.22%**	**63627.83**	**15.14%**	**17.78%**

Table 4.3: Summary of the results obtained by all the approaches. Each line contains the average value for the instances of the size and number of targets specified in the column 'Name'. After each set of instances with the same number of targets, a summary row is shown.

The results shown in Table 4.3 are extremely interesting. First, we can highlight that the ILP formulation for small instances can obtain, in all cases, the best possible solution even if it cannot certify the optimum in the given time limit. The '2-Opt AS' is the approach that obtains the worse results on the small instances. It obtains a gap to the best of 9.69% and a percentage of the best solution equal to 2.22%. The '2-Opt BS', even if the quality of the solutions is not comparable with the one produced by the ILP formulation, is better if compared to the '2-Opt AS'. Increasing the size of the instances, the ILP formulation is as well the algorithm that performs better both in terms of gap and the percentage of the best solution found. Interesting to highlight is that the heuristic approach, when increasing the size of the graph, obtains a higher number of best solutions, but also a higher gap to them. This means that, even if in some cases, the algorithm can find the best solution, in many others the quality of the solutions is poor and far from the optimum. A drastic change in the behaviours of the algorithms occurs in the instances with 100 targets where the

ILP formulation, within the time limit, cannot obtain admissible solutions. In these instances, only the heuristic approaches can obtain admissible solutions. Here the algorithm that performs better is the '2-Opt AS'. The search strategy applied by the '2-Opt BS' performs better in small instances where the search space is smaller. Therefore, when the size of the instances increases, selecting the best possible 2-opt induces the convergence into a local optimum while selecting randomly the 2-opt opens possibilities that induce better solutions.

In conclusion, we can affirm that the best algorithm in terms of computation time is '2-Opt AS'; with a small amount of time, the '2-Opt BS' can obtain better solutions on small instances. Therefore the search strategy loses effectiveness as increase the size of the instances. Instead, in terms of the quality of the solutions, the best approach remains the ILP formulation.

4.5 Conclusion

The problems that belong to the class of the Close Enough family are getting, year by year, more attention. This is thanks to the fast growth of wireless technologies, such as Bluetooth, Wi-Fi, and tag R-FID. In this thesis, we presented a new problem, namely Close-Enough Generalized Routing Problem (CEGRP) that, combining the characteristics of the CETSP and the CEARP, tries to handle all the scenarios in which the movements of an Unnamed Aerial Vehicle can be constrained, unconstrained or prohibited. Using the definition of the Constrained Flight Zone and the Free Flight Zone we handle, respectively, constrained and unconstrained movement for the vehicle. Furthermore, an effective conversion algorithm, namely, the Convert & Conquer Algorithm, is proposed. It converts an instance for the CEGRP into an instance for the Generalized Travelling Salesman Problem, by applying a set of conversion steps. This conversion allows us to apply on the converted instances, effective approaches for the GTSP.

We define a set of benchmark instances for the CEGRP of increasing size to assess the effectiveness of our algorithms. The results highlight how the conversion technique proposed can efficiently reduce the CEGRP into the GTSP. The results also demonstrate how even a simple technique such as the 2-Opt algorithm can provide good solutions in a reasonable amount of time.

The problem proposed in this thesis opens many possibilities for future research. First, new approaches can be defined to handle the problem not only in mixed instances but also in the two boundary conditions, which consist of considering only CFZ or only FFZ. New meta-heuristic approaches such as Genetic Algorithm [66] can be applied to obtain good solutions in mixed instances. A further important aspect consists of the position of the discretization points. In particular, a new approach that can dynamically change these points (Second Order Cone Programming Model [12]) could drastically improve the quality of the solutions. Nevertheless, we are still working on the definition of new instances of bigger size, of different types (e.g. random instances), and with a higher area of CFZ.

Chapter 5

Conclusion and Future Works

In this dissertation, we discussed in depth the problem of finding, in a reasonable amount of time, several paths or tours on graphs, defining advanced solving methods, as the ones proposed for the k-Colour Shortest Path Problem (k-CSPP), and describing new variants of the problem, such as the Concurrent Shortest Path Problem (CSPP) and the Close-Enough Generalized Routing Problem (CEGRP).

A new NP-Hard variant of the Shortest Path Problem, namely CSPP, is proposed in Chapter 2. It aims to find multiple paths of minimum length connecting all source-destination pairs while avoiding collisions between paths. This variant can model various scenarios such as handling the traffic in smart cities ([87], [54]), or define paths of minimum length for robots and forklifts in a warehouse ([52], [90]). The CSPP well fits all applications where several vehicles have to reach their destinations without colliding between them. In this thesis, we present three mathematical formulations for the problem. The first one is based on a flow formulation that considers a discretization of the time dimension. The second one is based on a bin packing formulation, and the third one is based on a flow formulation applied to a transformed graph to remove the time dimension. Furthermore, to handle large-size instances, we propose two heuristic approaches, namely DSPA, and MVSPA. We define a set of benchmark instances randomly generated. Results show that the two formulations can be applied to small instances to obtain optimal solutions, but cannot provide admissible solutions on the biggest ones due to the complexity of the problem. Furthermore, computational results show that the proposed heuristics can produce good solutions in a reasonable amount of time. This new variant of the SPP opens various possibilities for future research. The definition of new instances of different types and bigger sizes, or instances that change dynamically, can drastically increase the complexity of the problem and provide a challenging scenario for the proposed approaches. Furthermore, interesting can be the introduction of a maximum completion time within which to complete all tasks. Considering that these algorithms should be used in a real-time system, the computation time gets a crucial role, more than the solution quality. For this reason, it gets importance the definition of new heuristic and meta-heuristic approaches that can handle big scenarios in a considerably small amount of time.

In Chapter 3 we analyse the k-Colour Shortest Path Problem and provide advances in finding the optimum solution for the problem. The k-CSPP consists of finding a shortest path on a weighted edge-coloured graph for which the maximum number of different usable colours is defined *a priori* to be k. It can model several real-world scenarios in the field of network optimisation and multi-modal transport system. As a contribution to this problem, defined in [37], we first propose a heuristic approach, namely Colour-Constrained Dijkstra Algorithm (CCDA), that, in a few seconds, provides very good solutions, regardless of the size of the instances. We present a graph reduction technique that can remove on average more than 90% of the nodes, edges, and colours from the input graph. We define also an exact approach, called Reduced ILP (RILP), that, by combining the heuristic and the Graph Reduction Algorithm with an ILP formulation, can provide optimal solutions in a reasonable amount of time. To verify the effectiveness of the proposed approaches we performed several tests using the benchmark instances defined in [35] and [37]. We compare our algorithms with state-of-the-art methods presented by Ferone et al.. The results highlight that our algorithms outperform the state-of-the-art methods for grid instances. In general, our approaches are more stable in terms of the quality of the results and the computation time, regardless of the size of the graph, the number of colours, and the k value. We investigate the impact of the reduction technique in terms of the residual nodes and residual edges per colour. For future investigation, it may be interesting to work on the definition of new heuristic or meta-heuristic approaches such as, for example, a combination of our reduction process and the DP algorithm.

In the last chapter of this dissertation, while focusing on the definition of tours for drones, we discuss how the Close Enough problems can model several real-world scenarios. However, the CETSP, and the CEARP are not always accurate, as the drone is not always free to fly. We present the Close-Enough Generalized Routing Problem (CEGRP) that, combining the characteristics of the CETSP and the CEARP, tries to handle all the scenarios in which the movements of a drone can be constrained, unconstrained, or prohibited. We define the Constrained Flight Zone and the Free Flight Zone to handle, respectively, constrained and unconstrained movement. We present an effective conversion algorithm, namely, the Convert & Conquer Algorithm. It converts an instance for the CEGRP into an instance for the Generalized Travelling Salesman Problem, by applying a set of conversion steps. To assess the performance of the C&C algorithm, we define a set of benchmark instances for the CEGRP of increasing size. The results highlight how the conversion technique proposed can efficiently reduce the CEGRP into the GTSP. This problem opens many possibilities for future research. First, the above problem has only been examined in mixed instances (e.g. both FFZ and CFZ are present). Future developments include studying the two limiting cases. New approaches can be defined such as Genetic Algorithms [66], to obtain good solutions in any instance. A further important aspect consists of the position of the discretization points. In particular, a new approach that can dynamically change these points using a Second Order Cone Programming Model [12] could drastically improve the quality of the solutions.

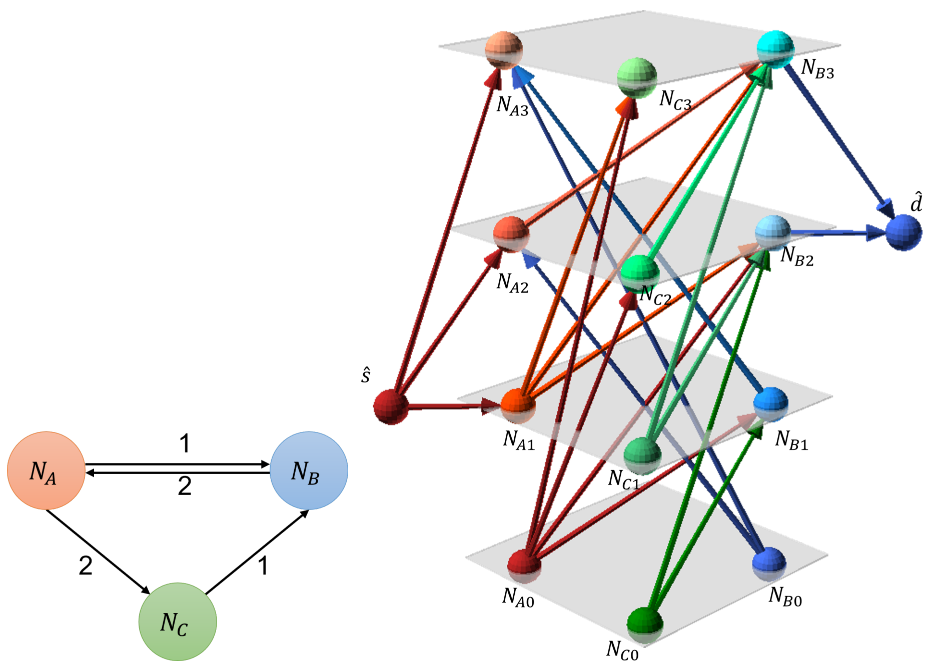
This dissertation is structured to provide an overview of some of the most important variants of the Shortest Path Problem and Close Enough problems. We present various advances in the definition of paths and tours on graphs. Given the emerging use of robots, drones, and autonomous vehicles in logistics and transportation, the results presented in this thesis aim to propose further studies on the SPP and its variants.

Chapter 6

Appendix

CSPP

To better understand the process described in Section 2.3.4, we provide in the following figure a complete example of the transformation process.



(a) Original Graph G (b) Transformed graph

Figure 6.1: Graph transformation complete example

Consider as G the graph shown in Figure 6.1a, so $N = \{N_A, N_B, N_C\}$ and $A = \{(N_A, N_B), (N_B, N_A), (N_A, N_C), (N_C, N_B)\}$, where for each arc the cost is the one shown in the figure and the capacity is equal to its cost. Let $SD = \{(N_A, N_B, 1)\}$ be the pair of source-destination to connect where N_A is the source node, N_B is the destination node, and $t_{sd} = 1$ is the starting time. After the transformation, considering $T_{max} = 3$, the graph G' of the CSPP will be:

$$N' = \{\hat{s}, N_{A0}, N_{A1}, N_{A2}, N_{A3}, N_{C0}, N_{C1}, N_{C2}, N_{C3}, N_{B0}, N_{B1}, N_{B2}, N_{B3}, \hat{d}\}$$

$$A' = \{(\hat{s}, N_{A1}), (\hat{s}, N_{A2}), (\hat{s}, N_{A3}), (N_{A0}, N_{B1}), (N_{A0}, N_{B2}), (N_{A1}, N_{B2}), (N_{A1}, N_{B3}), (N_{A2}, N_{B3}), (N_{A0}, N_{C2}), (N_{A0}, N_{C3}), (N_{A1}, N_{C3}), (N_{B0}, N_{A2}), (N_{B0}, N_{A3}), (N_{B1}, N_{A3}), (N_{C0}, N_{B1}), (N_{C0}, N_{B2}), (N_{C1}, N_{B2}), (N_{C1}, N_{B3}), (N_{C2}, N_{B3}), (N_{B2}, \hat{d},), (N_{B3}, \hat{d})\}$$

We note that to simplify the figure and the example proposed, we consider in G' only the arcs with a maximum length of $2 * c_{ij}$, where $(i, j) \in A$. Following the described procedure, the sets H that belong to the Compatibility Set $\mathcal{H}$ will be:

$$\begin{aligned}
H_0 &= \{(\hat{s}, N_{A1}), (\hat{s}, N_{A2}), (\hat{s}, N_{A3})\}\\
H_1 &= \{(N_{B2}, \hat{d}), (N_{B3}, \hat{d})\}\\
H_2 &= \{(N_{A0}, N_{B1}), (N_{A0}, N_{B2})\}\\
H_3 &= \{(N_{A0}, N_{B1}), (N_{A0}, N_{B2}), (N_{A1}, N_{B2}), (N_{A1}, N_{B3})\}\\
H_4 &= \{(N_{A0}, N_{B2}), (N_{A1}, N_{B2}), (N_{A1}, N_{B3}), (N_{A2}, N_{B3})\}\\
H_5 &= \{(N_{A1}, N_{B3}), (N_{A2}, N_{B3})\}\\
H_6 &= \{(N_{A0}, N_{C2}), (N_{A0}, N_{C3})\}\\
H_7 &= \{(N_{A0}, N_{C2}), (N_{A0}, N_{C3}), (N_{A1}, N_{C3})\}\\
H_8 &= \{(N_{A0}, N_{C2}), (N_{A0}, N_{C3}), (N_{A1}, N_{C3})\}\\
H_9 &= \{(N_{A0}, N_{C3}), (N_{A1}, N_{C3})\}\\
H_{10} &= \{(N_{C0}, N_{B1}), (N_{C0}, N_{B2})\}\\
H_{11} &= \{(N_{C0}, N_{B1}), (N_{C0}, N_{B2}), (N_{C1}, N_{B2}), (N_{C1}, N_{B3})\}\\
H_{12} &= \{(N_{C0}, N_{B2}), (N_{C1}, N_{B2}), (N_{C1}, N_{B3}), (N_{C2}, N_{B3})\}\\
H_{13} &= \{(N_{C1}, N_{B3}), (N_{C2}, N_{B3})\}\\
H_{14} &= \{(N_{B0}, N_{A2}), (N_{B0}, N_{A3})\}\\
H_{15} &= \{(N_{B0}, N_{A2}), (N_{B0}, N_{A3}), (N_{B1}, N_{A3})\}\\
H_{16} &= \{(N_{B0}, N_{A2}), (N_{B0}, N_{A3}), (N_{B1}, N_{A3})\}\\
H_{17} &= \{(N_{B0}, N_{A3}), (N_{B1}, N_{A3})\}
\end{aligned}$$

and the corresponding $f_H(H)$ are: $f_H(H_0) = 1$, $f_H(H_1) = 1$, $f_H(H_2) = 1$, $f_H(H_3) = 1$, $f_H(H_4) = 1$, $f_H(H_5) = 1$, $f_H(H_6) = 2$, $f_H(H_7) = 2$, $f_H(H_8) = 2$, $f_H(H_9) = 2$, $f_H(H_{10}) = 1$, $f_H(H_{11}) = 1$, $f_H(H_{12}) = 1$, $f_H(H_{13}) = 1$, $f_H(H_{14}) = 2$, $f_H(H_{15}) = 2$, $f_H(H_{16}) = 2$, $f_H(H_{17}) = 2$.

We can define the Compatibility Set $\mathcal{H}$ as:

$$\mathcal{H} = \{H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7, H_8, H_9, H_{10}, H_{11}, H_{12}, H_{13}, H_{14}, H_{15}, H_{16}, H_{17}\}$$

Finally, the vehicle set SD is transformed in $\hat{SD}$, where $\hat{SD} = \{(\hat{s}, \hat{d})\}$ with $\hat{s}$ is the super-source and $\hat{d}$ is the super-destination.

k-CSPP

Set	**Complete Name**	**Name**
Grid A	Grid_100x100_5940	A-G1
	Grid_100x100_7920	A-G2
	Grid_100x200_11910	A-G3
	Grid_100x200_15880	A-G4
	Grid_250x250_37350	A-G5
	Grid_250x250_49800	A-G6
	Grid_250x500_74775	A-G7
	Grid_250x500_99700	A-G8
	Grid_500x500_149700	A-G9
	Grid_500x500_199600	A-G10
	Grid_500x1000_299550	A-G11
	Grid_500x1000_399400	A-G12
Grid B	Grid_100x100_396	B-G1
	Grid_100x100_792	B-G2
	Grid_100x200_794	B-G3
	Grid_100x200_1588	B-G4
	Grid_250x250_2490	B-G5
	Grid_250x250_4980	B-G6
	Grid_250x500_4985	B-G7
	Grid_250x500_9970	B-G8
	Grid_500x500_9980	B-G9
	Grid_500x500_19960	B-G10
	Grid_500x1000_19970	B-G11
	Grid_500x1000_39940	B-G12

Table 6.1: The set of grid instances, the complete names of the grid instances, and the names used in Section 3.6 for reference.

Set	Complete Name	Name
Random A	Random_75000x750000_112500	A-R1
	Random_75000x750000_150000	A-R2
	Random_75000x1125000_168750	A-R3
	Random_75000x1125000_225000	A-R4
	Random_75000x1500000_225000	A-R5
	Random_75000x1500000_300000	A-R6
	Random_100000x1000000_150000	A-R7
	Random_100000x1000000_200000	A-R8
	Random_100000x1500000_225000	A-R9
	Random_100000x1500000_300000	A-R10
	Random_100000x2000000_300000	A-R11
	Random_100000x2000000_400000	A-R12
	Random_125000x1250000_187500	A-R13
	Random_125000x1250000_250000	A-R14
	Random_125000x1875000_281250	A-R15
	Random_125000x1875000_375000	A-R16
	Random_125000x2500000_375000	A-R17
	Random_125000x2500000_500000	A-R18
Random B	Random_75000x750000_07500	B-R1
	Random_75000x750000_15000	B-R2
	Random_75000x1125000_11250	B-R3
	Random_75000x1125000_22500	B-R4
	Random_75000x1500000_15000	B-R5
	Random_75000x1500000_30000	B-R6
	Random_100000x1000000_10000	B-R7
	Random_100000x1000000_20000	B-R8
	Random_100000x1500000_15000	B-R9
	Random_100000x1500000_30000	B-R10
	Random_100000x2000000_20000	B-R11
	Random_100000x2000000_40000	B-R12
	Random_125000x1250000_12500	B-R13
	Random_125000x1250000_25000	B-R14
	Random_125000x1875000_18750	B-R15
	Random_125000x1875000_37500	B-R16
	Random_125000x2500000_25000	B-R17
	Random_125000x2500000_50000	B-R18

Table 6.2: The set of random instances, the complete names of the random instances, and the names used in Section 3.6 for reference.

Instances	$\lvert C\rvert$	**0**	**1**	**2**	**3**	**>3**
A-G1	5940	88.859%	10.281%	0.806%	0.049%	0.005%
A-G2	7920	87.674%	9.994%	1.806%	0.404%	0.122%
A-G3	11910	87.301%	11.084%	1.405%	0.176%	0.034%
A-G4	15880	90.800%	8.494%	0.662%	0.039%	0.004%
A-G5	37350	96.076%	3.833%	0.088%	0.003%	0.000%
A-G6	49800	97.015%	2.933%	0.051%	0.001%	0.000%
A-G7	74775	96.973%	2.974%	0.052%	0.001%	0.000%
A-G8	99700	97.703%	2.261%	0.035%	0.001%	0.000%
A-G9	149700	96.824%	3.093%	0.081%	0.002%	0.000%
A-G10	199600	97.697%	2.259%	0.044%	0.001%	0.000%
A-G11	299550	98.265%	1.718%	0.017%	0.000%	0.000%
A-G12	399400	98.696%	1.294%	0.010%	0.000%	0.000%
Mean	**112627.08**	**94.490%**	**5.018%**	**0.421%**	**0.056%**	**0.014%**
B-G1	396	25.480%	33.258%	23.283%	10.783%	7.197%
B-G2	792	42.917%	33.093%	14.558%	6.035%	3.396%
B-G3	794	23.186%	25.088%	16.788%	10.579%	24.358%
B-G4	1588	48.558%	31.115%	13.199%	5.038%	2.091%
B-G5	2490	49.297%	32.787%	12.173%	4.096%	1.647%
B-G6	4980	75.811%	20.990%	2.890%	0.285%	0.024%
B-G7	4985	57.482%	31.045%	9.194%	1.864%	0.415%
B-G8	9970	73.224%	21.530%	4.323%	0.777%	0.146%
B-G9	9980	74.151%	21.974%	3.463%	0.381%	0.031%
B-G10	19960	86.555%	12.449%	0.952%	0.043%	0.002%
B-G11	19970	69.418%	22.541%	5.627%	1.659%	0.755%
B-G12	39940	85.872%	12.924%	1.128%	0.069%	0.007%
Mean	**9653.75**	**59.329%**	**24.899%**	**8.965%**	**3.467%**	**3.339%**

Table 6.3: Percentage of colours that have 0, 1, 2, 3, or more than 3 residual edges after the GRA is applied for grid instances.

Instances	$\lvert C\rvert$	**0**	**1**	**2**	**3**	**>3**
A-R1	112500	79.402%	10.456%	6.210%	2.644%	1.289%
A-R2	150000	99.959%	0.041%	0.000%	0.000%	0.000%
A-R3	168750	99.976%	0.024%	0.000%	0.000%	0.000%
A-R4	225000	99.982%	0.018%	0.000%	0.000%	0.000%
A-R5	225000	99.836%	0.163%	0.001%	0.000%	0.000%
A-R6	300000	99.877%	0.123%	0.000%	0.000%	0.000%
A-R7	150000	96.249%	2.980%	0.661%	0.098%	0.013%
A-R8	200000	99.963%	0.037%	0.000%	0.000%	0.000%
A-R9	225000	99.921%	0.079%	0.000%	0.000%	0.000%
A-R10	300000	99.941%	0.059%	0.000%	0.000%	0.000%
A-R11	300000	99.980%	0.020%	0.000%	0.000%	0.000%
A-R12	400000	99.985%	0.015%	0.000%	0.000%	0.000%
A-R13	187500	99.932%	0.068%	0.000%	0.000%	0.000%
A-R14	250000	99.949%	0.051%	0.000%	0.000%	0.000%
A-R15	281250	99.917%	0.083%	0.000%	0.000%	0.000%
A-R16	375000	99.937%	0.063%	0.000%	0.000%	0.000%
A-R17	375000	99.883%	0.117%	0.001%	0.000%	0.000%
A-R18	500000	99.912%	0.088%	0.000%	0.000%	0.000%
Mean	**262500**	**98.589%**	**0.805%**	**0.382%**	**0.152%**	**0.072%**
B-R1	7500	89.877%	7.353%	2.128%	0.536%	0.105%
B-R2	15000	94.124%	5.032%	0.745%	0.093%	0.006%
B-R3	11250	97.663%	2.138%	0.188%	0.012%	0.000%
B-R4	22500	98.779%	1.170%	0.050%	0.001%	0.000%
B-R5	15000	75.343%	8.757%	3.615%	1.569%	10.716%
B-R6	30000	80.872%	6.906%	1.749%	0.392%	10.081%
B-R7	10000	98.003%	1.886%	0.109%	0.002%	0.000%
B-R8	20000	99.664%	0.335%	0.002%	0.000%	0.000%
B-R9	15000	99.163%	0.829%	0.008%	0.000%	0.000%
B-R10	30000	99.581%	0.415%	0.004%	0.000%	0.000%
B-R11	20000	80.435%	3.876%	3.824%	3.552%	8.314%
B-R12	40000	84.011%	6.488%	4.612%	2.712%	2.177%
B-R13	12500	96.801%	3.042%	0.154%	0.003%	0.000%
B-R14	25000	98.361%	1.599%	0.039%	0.001%	0.000%
B-R15	18750	88.728%	2.337%	1.634%	2.026%	5.276%
B-R16	37500	81.288%	4.806%	4.827%	3.966%	5.113%
B-R17	25000	90.161%	2.855%	2.649%	2.124%	2.210%
B-R18	50000	92.611%	4.000%	2.176%	0.879%	0.334%
Mean	**22500**	**91.415%**	**3.546%**	**1.584%**	**0.993%**	**2.463%**

Table 6.4: Percentage of colours that have 0, 1, 2, 3, or more than 3 residual edges after the GRA is applied for random instances.

CEGRP

Name	GridSize	|N|	|E|	|T|	Radius	|Z|	%Z
10_10_30	10x10	100	180	10	30	1,2,3,5,10	30.142
10_10_50	10x10	100	180	10	50	1,2,3,5,10	22.334
10_10_100	10x10	100	180	10	100	1,2,3,5,10	22.502
50_10_30	50x50	2500	4900	10	30	1,2,3,5,10	24.244
50_10_50	50x50	2500	4900	10	50	1,2,3,5,10	14.542
50_10_100	50x50	2500	4900	10	100	1,2,3,5,10	17.174
100_10_50	100x100	10000	19800	10	30	1,2,3,5,10	20.272
100_10_50	100x100	10000	19800	10	50	1,2,3,5,10	29.25
100_10_50	100x100	10000	19800	10	100	1,2,3,5,10	30.766

Table 6.5: The table shows the information about the instances with 10 targets. In order are shown: the instance name (Name), the size of the graph (GridSize), the number of nodes ($|N|$), the number of edges ($|E|$), the number of targets ($|T|$), the radius of the targets (Radius), the number of CFZ ($|Z|$) and the percentage of CFZ with reference to the graph surface ($\%Z$).

Name	GridSize	|N|	|E|	|T|	Radius	|Z|	%Z
10_20_30	10x10	100	180	20	30	1,2,3,5,10	25.68
10_20_50	10x10	100	180	20	50	1,2,3,5,10	24.742
10_20_100	10x10	100	180	20	100	1,2,3,5,10	21.522
50_20_30	50x50	2500	4900	20	30	1,2,3,5,10	29.832
50_20_50	50x50	2500	4900	20	50	1,2,3,5,10	22.294
50_20_100	50x50	2500	4900	20	100	1,2,3,5,10	25.6
100_20_50	100x100	10000	19800	20	30	1,2,3,5,10	23.108
100_20_50	100x100	10000	19800	20	50	1,2,3,5,10	28.82
100_20_50	100x100	10000	19800	20	100	1,2,3,5,10	25.354

Table 6.6: The table shows the information about the instances with 20 targets. In order are shown: the instance name (Name), the size of the graph (GridSize), the number of nodes ($|N|$), the number of edges ($|E|$), the number of targets ($|T|$), the radius of the targets (Radius), the number of CFZ ($|Z|$) and the percentage of CFZ with reference to the graph surface ($\%Z$).

Name	GridSize	|N|	|E|	|T|	Radius	|Z|	%Z
10_50_30	10x10	100	180	50	30	1,2,3,5,10	20.09
10_50_50	10x10	100	180	50	50	1,2,3,5,10	23.87
10_50_100	10x10	100	180	50	100	1,2,3,5,10	21.38
50_50_30	50x50	2500	4900	50	30	1,2,3,5,10	24.004
50_50_50	50x50	2500	4900	50	50	1,2,3,5,10	29.516
50_50_100	50x50	2500	4900	50	100	1,2,3,5,10	24.504
100_50_50	100x100	10000	19800	50	30	1,2,3,5,10	26.906
100_50_50	100x100	10000	19800	50	50	1,2,3,5,10	21.804
100_50_50	100x100	10000	19800	50	100	1,2,3,5,10	23.5

Table 6.7: The table shows the information about the instances with 50 targets. In order are shown: the instance name (Name), the size of the graph (GridSize), the number of nodes ($|N|$), the number of edges ($|E|$), the number of targets ($|T|$), the radius of the targets (Radius), the number of CFZ ($|Z|$) and the percentage of CFZ with reference to the graph surface ($\%Z$).

Name	GridSize	|N|	|E|	|T|	Radius	|Z|	%Z
10_100_30	10x10	100	180	100	30	1,2,3,5,10	25.4
10_100_50	10x10	100	180	100	50	1,2,3,5,10	22.024
10_100_100	10x10	100	180	100	100	1,2,3,5,10	24.182
50_100_30	50x50	2500	4900	100	30	1,2,3,5,10	28.214
50_100_50	50x50	2500	4900	100	50	1,2,3,5,10	21.724
50_100_100	50x50	2500	4900	100	100	1,2,3,5,10	22.562
100_100_50	100x100	10000	19800	100	30	1,2,3,5,10	21.722
100_100_50	100x100	10000	19800	100	50	1,2,3,5,10	28.338
100_100_50	100x100	10000	19800	100	100	1,2,3,5,10	26.478

Table 6.8: The table shows the information about the instances with 100 targets. In order are shown: the instance name (Name), the size of the graph (GridSize), the number of nodes ($|N|$), the number of edges ($|E|$), the number of targets ($|T|$), the radius of the targets (Radius), the number of CFZ ($|Z|$) and the percentage of CFZ with reference to the graph surface ($\%Z$).

Bibliography

[1] Pasquale Avella, Maurizio Boccia, and Antonio Sforza. Resource constrained shortest path problems in path planning for fleet management. *Journal of Mathematical Modelling and Algorithms*, 3(1):1–17, 2004.

[2] John E Beasley and Nicos Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

[3] Behnam Behdani and J Cole Smith. An integer-programming-based approach to the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 26(3):415–432, 2014.

[4] Cédric Bentz, Marie-Christine Costa, Christophe Picouleau, and Maria Zrikem. The shortest multipaths problem in a capacitated dense channel. *European Journal of Operational Research*, 178(3):926–931, 2007.

[5] N. Bianchessi, A. Corberan, I. Plana, M. Reula, and J.M. Sanchis. The profitable close-enough arc routing problem. *Computers and Operations Research*, 140, 2022.

[6] Nicola Bianchessi, Ángel Corberan, Isaac Plana, Miguel Reula, and José M. Sanchis. The min-max close-enough arc routing problem. *European Journal of Operational Research*, 2021.

[7] Youssef Bichiou and Hesham A Rakha. Real-time optimal intersection control system for automated/cooperative vehicles. *International Journal of Transportation Science and Technology*, 8(1):1–12, 2019.

[8] Yunus Can Bilge, Doğukan Çağatay, Begüm Genç, Mecit Sarı, Hüseyin Akcan, and Cem Evrendilek. All colors shortest path problem. *arXiv preprint, arXiv:1507.06865*, 2015.

[9] Andreas Björklund and Thore Husfeldt. Shortest two disjoint paths in polynomial time. In *International Colloquium on Automata, Languages, and Programming*, pages 211–222. Springer, 2014.

[10] Hajo Broersma, Xueliang Li, Gerhard J Woeginger, and Shenggui Zhang. Paths and cycles in colored graphs. *The Australasian Journal of Combinatorics*, 31:299–312, 2005.

[11] M.Eugénia Captivo, João C.N. Clímaco, and Marta M.B. Pascoal. A mixed integer linear formulation for the minimum label spanning tree problem. *Computers & Operations Research*, 36(11):3082–3085, 2009.

[12] Francesco Carrabs, Carmine Cerrone, Raffaele Cerulli, and Ciriaco D'Ambrosio. Improved upper and lower bounds for the close enough traveling salesman problem. In *International Conference on Green, Pervasive, and Cloud Computing*, pages 165–177. Springer, 2017.

[13] Francesco Carrabs, Carmine Cerrone, Raffaele Cerulli, and Manlio Gaudioso. A novel discretization scheme for the close enough traveling salesman problem. *Computers & Operations Research*, 78:163–171, 2017.

[14] Francesco Carrabs, Carmine Cerrone, Raffaele Cerulli, and Bruce Golden. An adaptive heuristic approach to compute upper and lower bounds for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 32(4):1030–1048, 2020.

[15] Francesco Carrabs, Carmine Cerrone, Raffaele Cerulli, and Selene Silvestri. On the complexity of rainbow spanning forest problem. *Optimization Letters*, 12(3):443–454, 2018.

[16] Francesco Carrabs, Carmine Cerrone, Raffaele Cerulli, and Selene Silvestri. The rainbow spanning forest problem. *Soft Computing*, 22(8):2765–2776, 2018.

[17] Francesco Carrabs, Raffaele Cerulli, Giovanni Felici, and Gaurav Singh. Exact approaches for the orderly colored longest path problem: Performance comparison. *Computers & Operations Research*, 101:275–284, 2019.

[18] Francesco Carrabs, Raffaele Cerulli, Rosa Pentangelo, and Andrea Raiconi. A two-level metaheuristic for the all colors shortest path problem. *Computational Optimization and Applications*, 71(2):525–551, 2018.

[19] Lorenzo Castelli, Raffaele Pesenti, and Walter Ukovich. Scheduling multimodal transportation systems. *European journal of operational research*, 155(3):603–615, 2004.

[20] Carmine Cerrone, Raffaele Cerulli, Bruce Golden, and Rosa Pentangelo. A flow formulation for the close-enough arc routing problem. In *International Conference on Optimization and Decision Science*, pages 539–546. Springer, 2017.

[21] Carmine Cerrone, Ciriaco D'Ambrosio, and Andrea Raiconi. Heuristics for the strong generalized minimum label spanning tree problem. *Networks*, 74(2):148–160, 2019.

[22] R Cerulli, A Fink, M Gentili, and A Raiconi. The k-labeled spanning forest problem. *Procedia-Social and Behavioral Sciences*, 108:153–163, 2014.

[23] Raffaele Cerulli, Andreas Fink, Monica Gentili, and Stefan Voß. Extensions of the minimum labelling spanning tree problem. *Journal of Telecommunications and Information Technology*, pages 39–45, 2006.

[24] Yuan-Lin Chen, Shun-Chung Wang, and Chong-An Wang. Study on vehicle safety distance warning system. In *2008 IEEE International Conference on Industrial Technology*, pages 1–6. IEEE, 2008.

[25] Sergio Consoli, Jose Andres Moreno Perez, and Nenad Mladenović. Comparison of metaheuristics for the k-labeled spanning forest problem. *International Transactions in Operational Research*, 24(3):559–582, 2017.

[26] A. Corberan, R. Eglese, G. Hasle, I. Plana, and J.M. Sanchis. Arc routing problems: A review of the past, present, and future. *Networks*, 77(1):88–115, 2021.

[27] Walton Pereira Coutinho, Roberto Quirino do Nascimento, Artur Alves Pessoa, and Anand Subramanian. A branch-and-bound algorithm for the close-enough traveling salesman problem. *INFORMS Journal on Computing*, 28(4):752–765, 2016.

[28] Thiago Gouveia da Silva, Eduardo Queiroga, Luiz Satoru Ochi, Lucídio dos Anjos Formiga Cabral, Serigne Gueye, and Philippe Michelon. A hybrid metaheuristic for the minimum labeling spanning tree problem. *European Journal of Operational Research*, 274(1):22–34, 2019.

[29] Luigi Di Puglia Pugliese, Francesca Guerriero, and Michael Poss. The resource constrained shortest path problem with uncertain data: A robust formulation and optimal solution approach. *Computers & Operations Research*, 107:140–155, 2019.

[30] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[31] Michael Drexl. On the generalized directed rural postman problem. *Journal of the Operational Research Society*, 65(8):1143–1154, 2014.

[32] Tali Eilam-Tzoreff. The disjoint shortest paths problem. *Discrete applied mathematics*, 85(2):113–138, 1998.

[33] Matthias Englert, Heiko Röglin, and Berthold Vöcking. Worst case and probabilistic analysis of the 2-opt algorithm for the tsp. *Algorithmica*, 68(1):190–264, 2014.

[34] Luis Fanjul-Peyro, Federico Perea, and Rubén Ruiz. Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. *European Journal of Operational Research*, 260(2):482–493, 2017.

[35] Daniele Ferone, Paola Festa, Serena Fugaro, and Tommaso Pastore. A dynamic programming algorithm for solving the k-color shortest path problem. *Optimization Letters*, pages 1–20, 2020.

[36] Daniele Ferone, Paola Festa, Francesca Guerriero, and Demetrio Laganà. The constrained shortest path tour problem. *Computers & Operations Research*, 74:64–77, 2016.

[37] Daniele Ferone, Paola Festa, and Tommaso Pastore. The k-color shortest path problem. In *Advances in Optimization and Decision Science for Society, Services and Enterprises*, pages 367–376. Springer, 2019.

[38] P. Festa, F. Guerriero, D. Laganà, and R. Musmanno. Solving the shortest path tour problem. *European Journal of Operational Research*, 230(3):464–474, 2013.

[39] Matthew Flint, Marios Polycarpou, and Emmanuel Fernandez-Gaucherand. Cooperative path-planning for autonomous vehicles using dynamic programming. *IFAC Proceedings Volumes*, 35(1):481–486, 2002.

[40] Lester R Ford Jr. Network flow theory. Technical report, Rand Corp Santa Monica Ca, 1956.

[41] Liping Fu, Dihua Sun, and Laurence R Rilett. Heuristic shortest path algorithms for transportation applications: State of the art. *Computers & Operations Research*, 33(11):3324–3343, 2006.

[42] Bruce L Golden, Subramanian Raghavan, and Edward A Wasil. *The vehicle routing problem: latest advances and new challenges*, volume 43. Springer Science & Business Media, 2008.

[43] Teresa Gomes and José Craveirinha. Efficient calculation of the most reliable pair of link disjoint paths in telecommunication networks. *European Journal of Operational Research*, 181(3):1055–1064, 2007.

[44] Teresa Gomes, José Craveirinha, and Luísa Jorge. An effective algorithm for obtaining the minimal cost pair of disjoint paths with dual arc costs. *Computers & Operations Research*, 36(5):1670–1682, 2009.

[45] Damon J Gulczynski, Jeffrey W Heath, and Carter C Price. The close enough traveling salesman problem: A discussion of several heuristics. In *Perspectives in Operations Research*, pages 271–283. Springer, 2006.

[46] Gregory Gutin and Daniel Karapetyan. Generalized traveling salesman problem reduction algorithms. *Algorithmic Operations Research*, 4(2):144–154, 2009.

[47] Minh Hoang Ha, Nathalie Bostel, André Langevin, and Louis-Martin Rousseau. An exact algorithm for the close enough traveling salesman problem with arc covering constraints. In *ICORES*, pages 233–238, 2012.

[48] Minh Hoang Ha, Nathalie Bostel, André Langevin, and Louis-Martin Rousseau. Solving the close-enough arc routing problem. *Networks*, 63(1):107–118, 2014.

[49] Abdallah A Hassan and Hesham A Rakha. A fully-distributed heuristic algorithm for control of autonomous vehicle movements at isolated intersections. *International Journal of Transportation Science and Technology*, 3(4):297–309, 2014.

[50] Markó Horváth and Tamás Kis. Solving resource constrained shortest path problems with LP-based methods. *Computers & Operations Research*, 73:150–164, 2016.

[51] Stefan Irnich and Guy Desaulniers. Shortest path problems with resource constraints. In *Column Generation*, pages 33–65. Springer, 2005.

[52] Charles J Jacobus, Glenn J Beach, and Steve Rowe. Automated warehousing using robotic forklifts, February 24 2015.

[53] Richard M Karp. On the computational complexity of combinatorial problems. *Networks*, 5(1):45–68, 1975.

[54] Christos Katrakazas, Mohammed Quddus, Wen-Hua Chen, and Lipika Deka. Real-time motion planning methods for autonomous on-road driving: State-of-the-art and future research directions. *Transportation Research Part C: Emerging Technologies*, 60:416–442, 2015.

[55] Yusuke Kobayashi and Christian Sommer. On shortest disjoint paths in planar graphs. *Discrete Optimization*, 7(4):234–245, 2010.

[56] G. Laporte and I.H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61(1):227–262, 1995.

[57] Gilbert Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European journal of operational research*, 59(3):345–358, 1992.

[58] Gilbert Laporte, Ardavan Asef-Vaziri, and Chelliah Sriskandarajah. Some applications of the generalized travelling salesman problem. *Journal of the Operational Research Society*, 47(12):1461–1467, 1996.

[59] Gilbert Laporte and Yves Nobert. Generalized travelling salesman problem through n sets of nodes: an integer programming approach. *INFOR: Information Systems and Operational Research*, 21(1):61–75, 1983.

[60] Bing Liu, Qing Shi, Zhuoyue Song, and Abdelkader El Kamel. Trajectory planning for autonomous intersection management of connected vehicles. *Simulation Modelling Practice and Theory*, 90:16–30, 2019.

[61] Qiang Luo, Lunhui Xun, Zhihui Cao, and Yanguo Huang. Simulation analysis and study on car-following safety distance model based on braking process of leading vehicle. In *2011 9th World Congress on Intelligent Control and Automation*, pages 740–743. IEEE, 2011.

[62] Amgad Madkour, Walid G Aref, Faizan Ur Rehman, Mohamed Abdur Rahman, and Saleh Basalamah. A survey of shortest-path algorithms. *arXiv preprint arXiv:1705.02044*, 2017.

[63] Yannis Marinakis, Athanasios Migdalas, and Angelo Sifaleras. A hybrid particle swarm optimization – variable neighborhood search algorithm for constrained shortest path problems. *European Journal of Operational Research*, 261(3):819–834, 2017.

[64] Silvano Martello and Paolo Toth. Bin-packing problem. *Knapsack problems: Algorithms and computer implementations*, pages 221–245, 1990.

[65] C.E. Miller, R.A. Zemlin, and A.W. Tucker. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[66] Melanie Mitchell. Genetic algorithms: An overview. In *Complex.*, volume 1, pages 31–39. Citeseer, 1995.

[67] Zahra Naji-Azimi, Majid Salari, Bruce Golden, S. Raghavan, and Paolo Toth. Variable neighborhood search for the cost constrained minimum label spanning tree and label constrained minimum spanning tree problems. *Computers & Operations Research*, 37(11):1952–1964, 2010.

[68] Charles Edward Noon. *The generalized traveling salesman problem.* PhD thesis, University of Michigan, 1988.

[69] Christos H Papadimitriou. The euclidean travelling salesman problem is np-complete. *Theoretical computer science*, 4(3):237–244, 1977.

[70] Stefan Poikonen, Xingyin Wang, and Bruce Golden. The vehicle routing problem with drones: Extended models and connections. *Networks*, 70(1):34–43, 2017.

[71] Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

[72] Neil Robertson and Paul D Seymour. Disjoint paths—a survey. *SIAM Journal on Algebraic Discrete Methods*, 6(2):300–305, 1985.

[73] Neil Robertson and Paul D Seymour. Graph minors. xiii. the disjoint paths problem. *Journal of combinatorial theory, Series B*, 63(1):65–110, 1995.

[74] Daniel J Rosenkrantz, Richard E Stearns, and Philip M Lewis, II. An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, 6(3):563–581, 1977.

[75] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *2001 European control conference (ECC)*, pages 2603–2608. IEEE, 2001.

[76] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R. Sturtevant. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence*, 219:40–66, 2015.

[77] X.H. Shi, Y.C. Liang, H.P. Lee, C. Lu, and Q.X. Wang. Particle swarm optimization-based algorithms for tsp and generalized tsp. *Information Processing Letters*, 103(5):169–176, 2007.

[78] Robert Shuttleworth, Bruce L Golden, Susan Smith, and Edward Wasil. Advances in meter reading: Heuristic solution of the close enough traveling salesman problem over a street network. In *The Vehicle Routing Problem: Latest Advances and New Challenges*, pages 487–501. Springer, 2008.

[79] Deepinder Sidhu, Raj Nair, and Shukri Abdallah. Finding disjoint paths in networks. In *ACM SIGCOMM Computer Communication Review*, volume 21, pages 43–51. Citeseer, 1991.

[80] Olivia J. Smith, Natashia Boland, and Hamish Waterer. Solving shortest path problems with a weight constraint and replenishment arcs. *Computers & Operations Research*, 39(5):964–984, 2012.

[81] Roni Stern, Nathan R Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne T Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Satish Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Twelfth Annual Symposium on Combinatorial Search*, 2019.

[82] Robert Endre Tarjan and Anthony E Trojanowski. Finding a maximum independent set. *SIAM Journal on Computing*, 6(3):537–546, 1977.

[83] Christian Tilk, Ann-Kathrin Rothenbächer, Timo Gschwind, and Stefan Irnich. Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, 261(2):530–539, 2017.

[84] Don Torrieri. Algorithms for finding an optimal set of short disjoint paths in a communication network. *IEEE Transactions on Communications*, 40(11):1698–1702, 1992.

[85] Antonios Tsourdos, Brian White, and Madhavan Shanmugavel. *Cooperative path planning of unmanned aerial vehicles*, volume 32. John Wiley & Sons, 2010.

[86] Lara Turner. Variants of the shortest path problem. *Algorithmic Operations Research*, 6(2):91–104, 2011.

[87] Pravin Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on automatic control*, 38(2):195–207, 1993.

[88] Xingyin Wang, Bruce Golden, and Edward Wasil. A steiner zone variable neighborhood search heuristic for the close-enough traveling salesman problem. *Computers & Operations Research*, 101:200–219, 2019.

[89] Jake Weiner, Andreas T. Ernst, Xiaodong Li, Yuan Sun, and Kalyanmoy Deb. Solving the maximum edge disjoint path problem using a modified lagrangian particle swarm optimisation hybrid. *European Journal of Operational Research*, 293(3):847–862, 2021.

[90] Peter R Wurman, Raffaello D'Andrea, and Mick Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI magazine*, 29(1):9–9, 2008.

[91] Jinhui Yang, Xiaohu Shi, Maurizio Marchese, and Yanchun Liang. An ant colony optimization method for generalized tsp problem. *Progress in Natural Science*, 18(11):1417–1422, 2008.

[92] Zhao Yang, Ming-Qing Xiao, Ya-Wei Ge, De-Long Feng, Lei Zhang, Hai-Fang Song, and Xi-Lang Tang. A double-loop hybrid algorithm for the traveling salesman problem with arbitrary neighbourhoods. *European Journal of Operational Research*, 265(1):65–80, 2018.

[93] Bo Yuan, Maria Orlowska, and Shazia Sadiq. On the optimal robot routing problem in wireless sensor networks. *IEEE transactions on knowledge and data engineering*, 19(9):1252–1261, 2007.

[94] Shengli Yuan, Saket Varma, and Jason P Jue. Minimum-color path problems for reliability in mesh networks. In *Proceedings of the 24th Annual Joint Conference of the IEEE Computer and Communications Societies*, volume 4, pages 2658–2669. IEEE, 2005.

[95] Xiaoyan Zhu and Wilbert E. Wilhelm. Implementation of a three-stage approach for the dynamic resource-constrained shortest-path sub-problem in branch-and-price. *Computers & Operations Research*, 40(1):385–394, 2013.