UNIVERSITY OF NAPLES "FEDERICO II"

SCHOOL OF POLYTECHNIC AND BASIC SCIENCES

DEPARTMENT OF MATHEMATICS AND APPLICATIONS
"RENATO CACCIOPPOLI"

PHD THESIS FOR THE PHD PROGRAMME IN
MATHEMATICS AND APPLICATIONS
XXXIII CYCLE

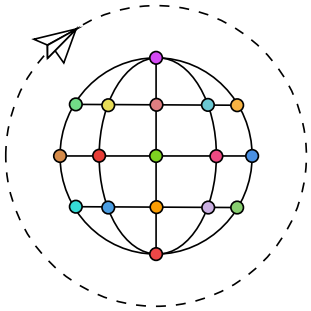
Optimizing and Reoptimizing: tackling static and dynamic combinatorial problems

CANDIDATE:
Serena Fugaro

SUPERVISOR:
Professor Paola Festa

Optimizing and Reoptimizing: tackling static and dynamic combinatorial problems

Serena Fugaro



Abstract

As suggested by the title, in this thesis both static and dynamic problems of Operations Research will be addressed by either designing new procedures or adapting well-known algorithmic schemes.

Specifically, the first part of the thesis is devoted to the discussion of three variants of the widely studied *Shortest Path Problem*, one of which is defined on dynamic graphs. Namely, first the *Reoptimization of Shortest Paths* in case of multiple and generic cost changes is dealt with an exact algorithm whose performance is compared with Dijkstra's label setting procedure in order to detect which approach has to be preferred.

Secondly, the *k-Color Shortest Path Problem* is tackled. It is a recent problem, defined on an edge-constrained graph, for which a Dynamic Programming algorithm is proposed here; its performance is compared with the state of the art solution approach, namely a Branch & Bound procedure.

Finally, the *Resource Constrained Clustered Shortest Path Tree Problem* is presented. It is a newly defined problem for which both a mathematical model and a Branch & Price procedure are detailed here. Moreover, the performance of this solution approach is compared with that of CPLEX solver.

Furthermore, in the first part of the manuscript, also the *Path Planning in Urban Air Mobility*, is discussed by considering both the definition of the *Free-Space Maps* and the computation of the trajectories. For the former purpose, three different but correlated discretization methods are described; as for the latter, a two steps resolution, offline and online, of the resulting shortest path problems is performed. In addition, it is checked whether the reoptimization algorithm can be used in the online step.

In the second part of this thesis, the recently studied *Additive Manufacturing Machine Scheduling Problem* with not identical machines is presented. Specifically, a Reinforcement Learning Iterated Local Search meta-heuristic featuring a Q-learning Variable Neighbourhood Search is described to solve this problem and its performance is compared with the one of CPLEX solver.

It is worthwhile mentioning that, for each of the proposed approaches, a thorough experimentation is performed and each Chapter is equipped with a detailed analysis of the results in order to appraise the performance of the method and to detect its limits.

Contents

List of Figures

List of Tables

List of Algorithms

CHAPTER 1

Introduction

The topics dealt with in this thesis are the result of a study carried out with two different approaches: on the one hand, we considered three variants of the classic *Shortest Path Problem* (SPP), which find application in extremely actual contexts. On the other hand, instead, we started from a real-world current scenario, namely the increasing use of 3D printing technology, and investigated one of the *Scheduling Problems* connected to it.

Thus, Part I of this thesis thoroughly describes three variants of SPP, namely the *Reoptimization of Shortest Paths*, the *k-Color Shortest Path Problem* and the *Resource Constrained Clustered Shortest Path Tree Problem*; then, Part II is devoted to the investigation of the *Additive Manufacturing Machine Scheduling Problem*. This Chapter features a brief outline of the well-known variants of Shortest Path Problems (Section 1.1) and the description of background and motivations for all the above mentioned problems (Sections 1.2 to 1.4).

As a final remark, in the following the term "arcs" will be used when referring to digraphs, while "edges" will be used for connections in undirected graphs.

1.1 Brief Overview of Shortest Path Problems

The Operations Research community has made a relevant effort to investigate the Shortest Path Problem since it can model several scenarios. Indeed, due to its wide applicability, SPP appears as a sub-problem in different Combinatorial Optimization problems [1].

Given a weighted graph, the objective of the classic SPP is to determine a minimum cost path from a *root* node r to a *destination* node d. A natural extension of the SPP (referred to as *Shortest Path Tree Problem*, SPT for short) consists in finding a shortest path tree, from a given root node, considering as destination all the other nodes.

While it is well known that both SPPs and SPTs are polynomially solvable,

more recent streams of research have focused on the solution of either *Dynamic SPPs* [69, 70], namely Shortest Path Problems on perturbed graphs, or *Constrained SPPs* (CSPP) [51, 68, 77], by devising both exact and heuristic techniques. Specifically, the interest of the scientific community in studying the former relies on the wide variety of applications in logistics [181] and transportation [41, 145] that often require to solve a sequence of SPPs in which two subsequent instances only slightly differ. Indeed, these differences can be related to: the perturbation of a subset of arc costs, the variation of the set of nodes/arcs, the change of the root node of the shortest path, or a combination of them.

Similarly, the significant effort made to investigate Constrained SPPs is twofold: on the one hand, by including additional constraints in the formulation of the classic SPP, it is possible to address diverse problems, like transportation planning in inter-modal networks, guaranty of reliability and robustness for transmission networks, design of optical networks. On the other hand, most of these problems are computationally intractable thus motivating the continuous development of efficient solution methods.

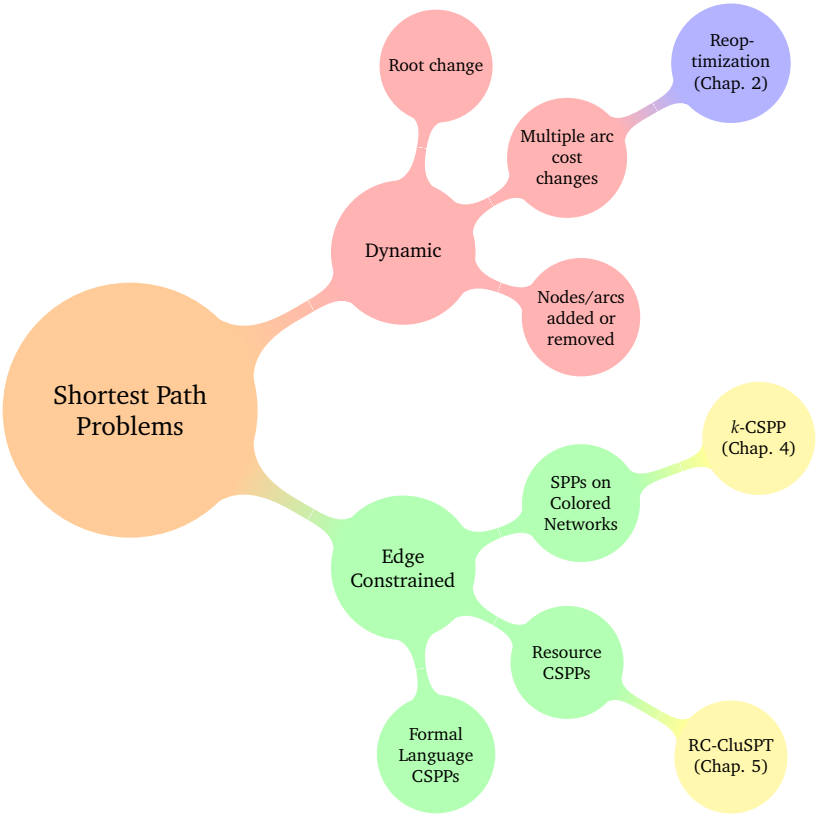


Fig. 1.1 Mind map about some of the most studied variants of SPP. Chapter numbers in parenthesis refer to those addressed in this manuscript.

Inspired by these streams of research, we have dealt with both kind of variants of SPP, addressing the *Reoptimization of Shortest Paths*, the *k-Color Shortest Path Problem* (*k*-CSPP), and the *Resource Constrained Clustered Shortest Path Tree Problem* (RC-CluSPT).

The mind map in Fig. 1.1 summarizes the main sub-classes for each variant and indicates in which Chapter are the ones addressed in this thesis.

1.2 The Reoptimization of Shortest Paths

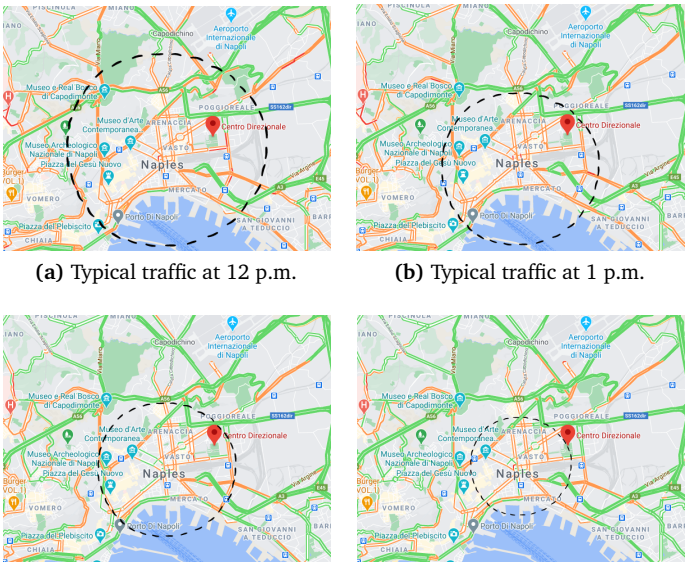
Given a sequence of slightly different SPPs, the *Reoptimization* approach is meant to reduce the computational effort required to get an optimal solution for each SPP of the sequence; hence, in contrast to solution techniques that solve each problem *ex-novo*, this method tries to solve a problem in the sequence by reusing valuable information gathered in the solution of the previous one.

From a practical perspective, such an approach can be useful, for example, in drawing up an emergency plan in case of a catastrophe [181]. Instead, in terms of SPP, the reoptimization approach may be suitable for all the situations characterized by a particular underlying graph structure. Explicitly, these kinds of graphs are referred to as *dynamic graphs* since the set of nodes and/or the set of arcs as well as the cost function may be subject to change (see Fig. 1.1).

In the scientific literature, the problem of solving a sequence of slightly different SPPs/SPTs has been addressed from two points of view: one related to the *root change* of the shortest path and the other concerning the *cost change* for a subset of arcs. Since the former has already been widely investigated, in Chapter 2 we describe the reoptimization framework focusing on the general situation of multiple cost changes.

In particular, we present the primal-dual procedure devised by Pallottino and Scutellà [144], which performs the reoptimization by considering subsets of arcs in each iteration, in contrast to the deeply studied algorithms that reoptimize with respect to one arc at a time [see 32]. In Festa et al. [83] we propose an efficient implementation of this algorithmic framework and conduct an intensive computational evaluation of it, as reported in Chapter 2. Specifically, a thorough experimentation is conducted with the aim of detecting which features make an instance more suitable for the reoptimization.

We investigated scenarios including graphs whose cost functions are affected by a local change, since, oftentimes, in real world context the perturbation of the costs affects only a small and structured portion of the network. This model has several applications, e.g. in transportation and urban traffic management: given the road network associated to a suburban area of a city and computing the shortest paths from all the points of that area to the financial district of the city in regular intervals of time, it comes noteworthy that, during the rush hour, the perturbation affects a ring-like set of arcs surrounding the district and tightening around it as time passes (Fig. 1.2) [144].



(a) Typical traffic at 12 p.m. **(b)** Typical traffic at 1 p.m.

(c) Typical traffic at 2 p.m. **(d)** Typical traffic at 3 p.m.

Fig. 1.2 `Google Maps` typical traffic map around the city centre of Naples. Colors represent the traffic flow, from *fast* (green) to *slow* (red).

1.2.1 Applications in Urban Air Mobility

In Section 1.2, it has been duly noted that the reoptimization approach might be suitable to solve problems which present an underlying structure of dynamic graphs. Indeed, this class of problems contains also those for which the information related to the graph is affected by uncertainty and/or is given as estimates [137]. In fact, a way to tackle these problems is by considering as deterministic the available information, in order to obtain a first solution which is then updated in correspondence of the new estimates. In particular, the computation of trajectories in the *Path Planning* for *Urban Air Mobility* (UAM) can be placed in this context.

Nowadays, the design of UAM models represents a flourishing field of research due to the growing awareness that the use of flying vehicles, with or without driver, could be an efficient way to overcome the barriers imposed by traffic, thus speeding up the transport service and facilitating rescue operations [170].

Actually, two relevant aspects of the UAM modelling are addressed by the Path Planning [173]: i) the representation of the obstacle-free 3D space in which the motion is going to happen, and ii) the computation of the feasible trajectories. Particularly, this latter issue requires the resolution of SPPs in which the costs are defined according to specific criteria, which take into account pollution and noise produced by the aircraft, its fuel consumption and travel time. Thus, some information related to these cost functions might be available

only as estimates. Consequently, to compute the shortest paths between different locations, a suitable approach is the *two-steps resolution*: the trajectories are obtained at first *offline*, i.e. considering the costs as deterministic, and then *online*, i.e. the estimates are updated and eventually the solution too. Indeed, the switching from the offline to the online environment makes the two-steps resolution similar to the resolution of SPPs on two slightly different graphs.

Thus, with this consideration in mind, we approach the two-steps resolution with the reoptimization algorithm [83, 144]. Specifically, in Chapter 3 we address both the issues of the Path Planning in UAM: on the one hand, we focus on the representation of the obstacle-free 3D city space, by designing three different but correlated discretization methods. Indeed, these procedures define plane sections of the 3D space and build a particular graph topology on each section, before connecting them properly. The graphical representation of the obtained trajectories highlights that randomness should be avoided both in the graph topology and in cost function.

Then, we define a problem-specific perturbation to simulate the switching from the offline to the online environment and compare the reoptimization with the resolution ex-novo. Actually, the analysis of the results underlines that the best performing approach is the latter, regardless of the percentage of network affected by the change.

1.3 Constrained Shortest Path Problems

The Constrained Shortest Path Problems (CSPPs) can be classified into subclasses according to the main features of their additional constraints [76]; among those with edge constraints, the most studied comprise: *Formal Language CSPPs*, SPPs on *Edge-Colored networks*, and *Resource CSPPs* (see Fig. 1.1).

The Formal Language CSPPs are defined on networks whose edges are given a label belonging to a *finite alphabet* Σ, and the aim is to find a shortest path such that its label belongs to a *formal language* $\mathcal{L}(\Sigma)$ built over Σ. Also this variant of CSPP is computationally intractable, even when further assumptions about $\mathcal{L}(\Sigma)$ are made [13]. Among the other scenarios, the formal language CSPPs could model the *Inter-modal Route Planning Problem*. In fact, networks that allow for several mode choices can be represented by means of an edge-labeled graph, and Formal Language CSPPs can easily define the problem of finding an optimal path subject to mode-preference constraints.

A prolific stream of research deals with SPPs on colored networks, i.e. graphs whose edges are given additional versatile labels, denoted as *colors*, which allow to model an elevated number of diverse problems. By way of example, letting the *reload cost* ρ_{bc} for colors b, c be the amount to be paid if, in a path, an edge of color c is traversed just after an edge of color b, the *Route Planning Problem* could be modeled as a *Minimum Reload Cost Problem* [5, 93], with reload costs representing tolls. Besides, among the other applications, there are connectivity and reliability problems arising in the field of telecommunication [26, 31]. In the former case, different colors could denote diverse connections between pairs

of nodes; in the latter case, instead, the same color is chosen to denote links that could be damaged by a single event happening in the network. In regards to this latter application, we addressed the problem detailed in Section 1.3.1.

Finally, in the Resource CSPP, in addition to minimizing distances travelled, solutions are subject to restrictions related to a set of resources which represent a broad variety of attributes [130, 158]. Indeed, the formulation of these constraints depends itself on the nature of the resources which could be classified as *numerical* and *cumulative* (e.g. travel time, fuel consumption), *numerical* and *non-cumulative* (e.g. road width, height), and *indexed* or *categorical* (e.g., parking restrictions, type of road) [10, 106]. It is noteworthy that even when a single resource function is defined, the corresponding Resource CSPP is computationally intractable [90]. In Section 1.3.2 we outline the characteristics of the Resource CSPP addressed in Ferone et al. [78].

1.3.1 The *k*-Color Shortest Path Problem

In the context of transmission network design, Ferone et al. [71] recently proposed the *k-Color Shortest Path Problem* (*k*-CSPP) that is related to the Minimum Color Path Problem since it aims to find a shortest path – on a colored network – that traverses at most k different colors. For example, referring to the two scenarios depicted in Fig. 1.3 and letting $k = 2$, the path from r to d in Fig. 1.3(a) is infeasible while that in Fig. 1.3(b) has to be preferred, though it could be potentially longer. In particular, also this CSPP has been proved to be computationally intractable, namely it is **NP**-hard.

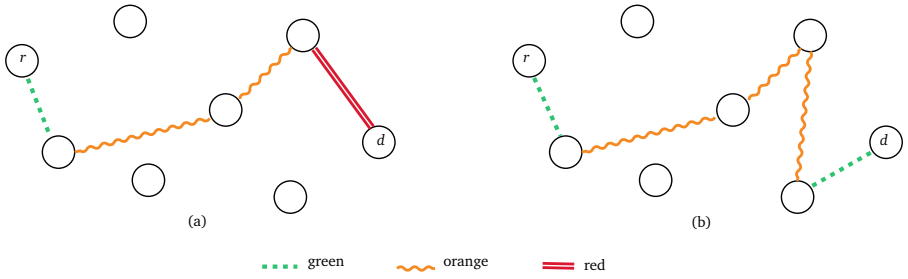


Fig. 1.3 Two edge-colored paths: (a) with four edges, and traversing three colors, (b) with five edges, and traversing two colors.

Moreover, since similar variants of CSPPs have already been tackled by means of the *Dynamic Programming* framework [51, 52], in Ferone et al. [74] we devise a Dynamic Programming algorithm based on a *path-labeling* approach and an A^*-like exploration strategy. Chapter 4 provides a further description of the problem and briefly outlines the state of the art procedure, namely a Branch & Bound method, devised to solve it [71]; additionally, our new algorithmic proposal is detailed along with the numerical comparison made to appraise its performance. Specifically, the Dynamic Programming algorithm has been compared with both the Branch & Bound technique and the solution of the

mathematical model obtained with CPLEX solver; the results highlight that the Dynamic Programming algorithm vastly outperformed the other approaches.

1.3.2 The Resource Constrained Clustered SPT

The *clustered* (or *generalized*) versions of several classic combinatorial optimization problems have received a great share of attention, due to their wide range of application [66]. At the same time, the design of realistic optimized routes in transportation or telecommunication networks generally requires finding optimal paths accounting for assigned link attributes. Consequently, several real-world problems could be modeled and solved through (Clustered) Resource CSPPs [107]. For instance, the Clustered Resource CSPP with local resource constraints could model the hop limited SPT with bandwidth constraints [177].

The optimization problem described in Chapter 5, namely the *Resource Constrained Clustered Shortest Path Tree Problem* (RC-CluSPT) [78], lies at the intersection between the Clustered SPT and the Resource CSPP. Specifically, the aim is to determine a shortest path tree respecting the local resource constraints and inducing connected subgraphs within each cluster. Since the classic Resource Constrained SPT is **NP**-hard, the same computational intractability characterizes the RC-CluSPT too. In Ferone et al. [78] we draw up a *path-based formulation* whose intrinsic structure allows for a *Dantzig-Wolfe (DW) decomposition* [46] based on Resource CSPP Decomposition. Then, for its resolution, a *Branch & Price* algorithm featuring a *Column Generation* approach with a *Multiple Pricing Scheme* is devised.

A thorough description is provided in Chapter 5, along with the detailed computational experimentation made in Ferone et al. [78] to appraise the effectiveness of the Branch & Price. Specifically, the comparison between this technique and the solution of the mathematical model performed by CPLEX solver reveals that the Branch & Price significantly improves the performance of CPLEX.

1.4 The Additive Manufacturing Machine Scheduling Problem

Nowadays, the use of *Additive Manufacturing* (AM) is spreading in the industry because of the affordable maintenance and the low processing costs of AM machines [57]. Specifically, the AM comprises several fabrication processes that exploit a wide variety of materials to create products ranging from medical implants to structural and aerospace engineering designs [9, 44, 147, 148]. Among the others, the *Selective Laser Melting* (SLM) technique, developed at the Fraunhofer Institute for Laser Technology ILT [24, 64], is widely in use since it enables the manufacturing of metal components layer by layer, according to a 3D CAD model; moreover it can handle complex geometries without part-specific tooling or pre-production costs [120].

Anyway, the optimization problems related to the planning of AM processes feature two types of operations: *nesting* and *scheduling*, where the former refers to the orientation of parts and their packing; since in the scientific literature these issues have been tackled both separately [38, 171] and jointly [95], in Alicastro et al. [3] we focus on the scheduling operations.

Specifically, Kucukkoc [119] proposed the mathematical model of an *Additive Manufacturing Machine Scheduling Problem* (AMM-SP) inspired by the SLM technique: a finite set of parts with several characteristics has to be produced by a finite set of AM machines, each of which may have different specifications, and the aim is to regroup parts and allocate them into job batches to be scheduled on the machines in order to minimize the total completion time, i.e. the *makespan*. In particular, in Alicastro et al. [3] we devise a *Reinforcement Learning Iterated Local Search* meta-heuristic, based on the implementation of a *Q-Learning Variable Neighbourhood Search*; this method has been compared with the results attained by the CPLEX solver and an *Evolutionary Algorithm* recently proposed for a similar problem, and adapted for the AMM-SP.

In Chapter 6, both the mathematical model and the proof of the **NP**-hardness of the AMM-SP are given. Furthermore, this Chapter presents a thorough analysis of the results of the extensive experimentation conducted on a rich variety of problem instances. These results evidence that the proposed meta-heuristic obtains statistically significant improvements compared with the state of the art solution approaches.

Part I

Three Variants of Shortest Path Problem

CHAPTER 2

The Reoptimization of Shortest Paths

The *Shortest Path Problem* (SPP) is among the most famous operational research problems; this Chapter is addressed to the study of the *Shortest Path Reoptimization Problem*. Given a sequence of SPP, in which two subsequent instances solely differ by a slight change in the graph structure, the goal of the reoptimization consists in solving the k^{th} SPP of the sequence by reusing valuable information gathered in the solution of the $(k-1)^{th}$ one.

In Section 2.1 we present the mathematical formulations of Shortest Path Tree Problem and its dual, while a brief overview of the reoptimization framework is drawn in Section 2.2. Then, Section 2.3 is devoted to the description of the algorithmic framework, due to Pallottino and Scutellá [144], that we implemented by exploiting efficient ad hoc data structures. Finally, in Section 2.4 we detail the extensive numerical evaluation of the performance of this algorithm [83].

2.1 Mathematical Formulation

In this Section, we describe the mathematical formulation of the Shortest Path Tree Problem and its dual, since the methodology we will present in Section 2.3 is strongly related to the primal-dual paradigm.

2.1.1 Primal Formulation of the Shortest Path Tree Problem

Let $G = (V, A, c)$ be a directed graph, where $|V| = n$, $|A| = m$, and $c\colon A \mapsto \mathbb{R}$ is a *cost function* assigning a real cost c_{ij} to each arc $(i, j) \in A$. For simplicity, we suppose that:

1. G does not have parallel arcs, i.e., for each pair of connected nodes there is exactly one arc connecting them;

2. G is strongly connected: this is not restrictive, since if there is no arc $(i, j) \in A$ for $i, j \in V$ then we can add it to A with $c_{ij} = M$ where M is big enough.

Given a root node $r \in V$, the *Shortest Path Tree problem* rooted in r (SPT_r) consists in finding a directed tree $\mathbf{T}_r^*$ such that, for each node $i \in V$, the path from r to i in the tree is a minimum cost path from r to i in G. In order to consider bounded solutions, we will assume that there are no negative cost cycles in the graph G nor can they be created.

Recalling the definitions of the *forward star* of a node $i \in V$ as the set $FS(i) = \{(i, j) \in A | j \in V\}$ and the *backward star* as the set of arcs $BS(i) = \{(j, i) \in A | j \in V\}$, we can formulate SPT_r as the following linear programming problem:

$$\text{(SPT\_r)} \min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{2.1a}$$

subject to:

$$\sum_{(i,j)\in FS(i)} x_{ij} - \sum_{(j,i)\in BS(i)} x_{ji} = n - 1, \qquad \text{if } i = r \tag{2.1b}$$

$$\sum_{(i,j)\in FS(i)} x_{ij} - \sum_{(j,i)\in BS(i)} x_{ji} = -1, \qquad \forall i \in V \backslash \{r\} \tag{2.1c}$$

$$x_{ij} \geq 0, \qquad \forall (i, j) \in A. \tag{2.1d}$$

Though the simplex method can be used to solve SPT_r, the algorithm widely in use today is the more efficient Dijkstra's label setting procedure [56].

2.1.2 Dual Formulation of the Shortest Path Tree Problem

Let $\pi = (\pi_1, \dots \pi_n)$ be the vector of dual variables, where π_i is called the *potential* of node i; then the dual formulation of SPT_r is the following:

$$\text{(Dual-SPT\_r)} \ \max \left[(n-1)\pi_r - \sum_{j \neq r} \pi_j \right] \tag{2.2a}$$

subject to:

$$\pi_i - \pi_j \leq c_{ij} \qquad \forall (i, j) \in A, \tag{2.2b}$$

where constraints (2.2b) are defined as *dual feasibility*. Indeed, since the *reduced costs* of arcs are defined as $\bar{c}_{ij} = c_{ij} + \pi_j - \pi_i$ for each $(i, j) \in A$, given an optimal solution $\mathbf{T}_r^* = (V, A_r)$ for SPT_r, the *optimality conditions of linear programming* (see. [144]) can be rewritten as

$$\bar{c}_{ij} \geq 0, \qquad \forall (i, j) \in A \tag{2.3a}$$

$$\bar{c}_{ij} = 0, \qquad \forall (i,j) \in A_r. \tag{2.3b}$$

In particular, constraints (2.3a) define the dual feasibility conditions while those (2.3b) state the *complementary slackness* (CS) conditions.

Computing the dual solution

Given an instance of SPT_r, the corresponding reduced costs could be computed by means of the dual optimal solution of Dual-SPT_r. Indeed, the *auction algorithm* [18, 19] could be employed at this purpose. This procedure builds an optimal path following a primal-dual approach, through successive phases of *path extensions*, *path contractions* and *dual price increase*.

However, on the one hand, the computational complexity of the auction algorithm is significant [29, 30]. On the other hand, there exists a link between the labels $d_r[\cdot]$ generated when solving SPT_r via Dijkstra's algorithm and the optimal dual solution π.

Lemma 2.1.1 ([19])**.** *Let π be an optimal solution for Dual-SPT_r. It holds that $\forall h \in V$ $d_r[h] = \pi_r - \pi_h$.*

Proof. Let h be a node in V. If there is an arc $(r,h) \in A$ then either (r,h) belongs to the optimal set of arcs A_r or not.

1. If $(r,h) \in A_r$, then $d_r[h] = c_{rh}$. Moreover, from (2.3b), it holds $\bar{c}_{rh} = 0$, or equivalently $\pi_r - \pi_h = c_{rh}$.

2. If $(r,h) \notin A_r$, the shortest path $P_{rh} \subset \mathbf{T}_r^*$ from the root r to h contains at least a node $i \in V \backslash \{r,h\}$, i.e., $P_{rh} = (r,i) \cup (i,j) \cup \cdots \cup (l,h)$; hence $d_r[h] = c_{ri} + c_{ij} + \cdots + c_{lh}$. Since all the arcs of P_{rh} belongs to the optimal solution, they have zero reduced costs (conditions (2.3b)); also $c_{ri} = \pi_r - \pi_i$ from 1., then $d_r[h] = \pi_r - \pi_i + \pi_i - \pi_j + \ldots \pi_l - \pi_h = \pi_r - \pi_h$.

If there is no arc $(r,h) \in A$, then the strongly connection of G ensures the existence of at least one directed path from r to h in G; thus, given the shortest path $P_{rh} \subset \mathbf{T}_r^*$, the thesis follows as in 2. □

Theorem 2.1.2 ([19])**.** *The vector $\bar{\pi} \in \mathbb{R}^n$ such that $\bar{\pi}_h = \bar{\pi}_r - d_r[h]$ $\forall h \in V \backslash \{r\}$ is an optimal solution for Dual-SPT_r.*

Proof. If P_{rh} is a shortest path in the graph from the root r to h, then for $\forall i,j$ belonging to this path, the subpath $P_{ij} \subset P_{rh}$ is a shortest path from i to j [see 42, Lemma 24.1]. As a consequence, for any arc $(i,j) \in A$, $\bar{\pi}_i - \bar{\pi}_j = d_r[j] - d_r[i] \leq c_{ij}$, thus $\bar{\pi}$ is dual admissible. Moreover, when the arc (i,j) belongs to the shortest path, $c_{ij} = d_r[j] - d_r[i]$ which proves that (2.3b) holds. □

Indeed, π_r could be chosen as the maximum label generated by Dijkstra's procedure in order to have a non-negative dual solution. In particular, $\forall (i,j) \in A$ it holds that $\bar{c}_{ij} = c_{ij} + d_r[i] - d_r[j]$ [see 19].

2.2 Reoptimization Framework

In several application contexts, it might be necessary to solve a sequence of SPPs such that the corresponding instances are graphs whose structure may be affected by some changes [41, 145]. At this purpose, the *reoptimization* approach is meant to reduce the computational effort required to get an optimal solution for each SPP of the sequence; hence, in contrast to solution techniques that solve each problem *ex-novo*, this method tries to solve a SPP starting from the solution of the one that precedes it in the sequence [69, 70]. This approach may be suitable for all the situations characterized by a particular underlying graph structure. Explicitly, this kind of graphs are referred to as *dynamic graphs* since the set of nodes, the set of arcs or both may be subject to change.

Indeed, due to the interplays among the variants of SPP on dynamic graphs [see 70, Figure 1], it is always possible to reduce the case of insertion/deletion of nodes or arcs to that of cost change [138]. Consequently, in the scientific literature, the problem of solving a sequence of slightly different SPPs/SPTs has been addressed from two points of view: the first related to the *root change* of the shortest path and the second concerning the *cost change* for a subset of arcs.

Assuming that an optimal tree $\mathbf{T}_r^*$ rooted in r has been computed, the solution exhibits some theoretical properties [88]. Specifically, when a root change occurs, i.e. $s \neq r$ becomes the new root node, the paths contained in the sub-tree of $\mathbf{T}_r^*$, rooted in s, are still optimal shortest paths from s to its descendants.

Proposition 2.2.1 ([88]). *Let $\boldsymbol{T}_r^*(h)$ denote the sub-tree of $\boldsymbol{T}_r^*$ containing node h and all its descendants and let $d(i, j)$ denote the cost of the shortest path from i to j. Then $\boldsymbol{T}_r^*(s) \subseteq \boldsymbol{T}_s^*$ and $d(s, j) = d(r, j) - d(r, s)$ for any $j \in \boldsymbol{T}_r^*(s)$.*

Thus, storing information about the old solution is an efficient strategy when reoptimizing, since a consistent part of the previously optimal tree belongs to the new optimal solution.

It is worth to note that reducing the maximum cost within the graph does not change the sets of feasible and optimal solutions, as formally stated in Proposition 2.2.2.

Proposition 2.2.2 ([88]). *Consider a vector $\pi \in \mathbb{R}^n$ such that*

$$l_{ij} = c_{ij} + \pi_i - \pi_j \geq 0 \quad \forall (i, j) \in A. \tag{2.4}$$

Then, SPT_r with arc lengths c_{ij} is equivalent to the problem of finding a shortest path tree rooted in r with arc lengths l_{ij} given by Equation (2.4).

An improvement of the classic Dial's implementation of Dijkstra's algorithm [54, 55] was derived by exploiting these theoretical results [88].

Remark 1. The reoptimization on dynamic graphs represents a more general scenario. In fact, also the case of root change is reducible to that of arc cost change [88]. Let r be the old root and s be the new one, then connect both with a *dummy node o*, as showed in Fig. 2.1. Moreover, set $c_{or} = 0$ and $c_{os} = U$

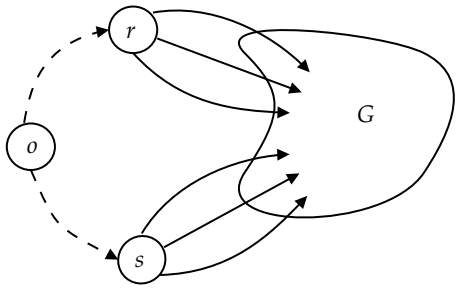


Fig. 2.1 Reduction of root change to arc cost change: r is the old root, s the new one and o is a dummy node added to G.

where U is a large positive integer, namely $U \geq max_{(i,j)\in A} c_{ij}$. Indeed, if U is large enough, a shortest path tree rooted in o differs from $\mathbf{T}^*_r$ only by arc (o,r). Then, setting $c'_{or} = 2U$ a new optimal tree rooted in o, will differ from $\mathbf{T}^*_s$ only for the arc (o,s) and $d(s,h) = d(o,h) - U$ for any $h \in V$.

Therefore, the problem of finding $\mathbf{T}^*_s$ given $\mathbf{T}^*_r$ is equivalent to the problem of finding, in the extended network, a new SPT, when the length of arc (o,r) is increased from 0 to $2M$. □

2.2.1 State of the Art

In this Section we briefly recall the main state of the art approaches proposed to address the SPT reoptimization.

Since the first algorithmic proposal [88], many efforts have been made to derive efficient techniques to handle the case of root change; at this purpose, e.g., a modified version of the dual simplex approach was devised in Florian et al. [86]. The most recent advance in this direction has been made by Festa et al. [81]: the reoptimization paradigm is implemented via a sequence of operations (called *extensions*) performed by applying a strongly polynomial auction algorithm, based on the *virtual source* concept [29, 30].

Initially, only the case of perturbations of the same type, namely increase or decrease, was taken into account. First, Ramalingam and Reps [152] tackled the reoptimization of SPP in the case of single arc cost increase with an *incremental algorithm* devised ad hoc for a version of the *grammar problem*. Later, Buriol et al. [27] empirically proved that this algorithm outperformed an optimization from scratch with Dijkstra's algorithm. Moreover, employing a technique based on to the *reverse graph representation*, it was adapted to cope also with the case of single arc cost decrease.

However, most of the algorithms, proposed to deal with the case of multiple cost changes, perform the reoptimization in subsequent phases, by considering a single arc cost change in each of them [32]. In contrast to this scheme, Pallottino and Scutellá [144] described a procedure based on the partition of the set of arcs, whose cost has been modified, in order to reoptimize the current tree

in subsequent phases, each with respect to a specific subset of arcs, and their reoptimization paradigm is based on a primal-dual approach.

2.3 Primal-dual Reoptimization Technique

For the sake of completeness and clarity, in this Section, we provide a description of the primal-dual solution approach devised in Pallottino and Scutellá [144] and for which we proposed an implementation [83].

Suppose that, after a pair $(\mathbf{T}^*_r, \pi^{(r)})$ of optimal solutions for SPT_r and Dual-SPT_r has been calculated, a new cost – either lower or higher than the old one – is assigned to a subset K of the arcs of G. Then c'_{ij} denotes the new integer cost for the arc (i, j) and $\bar{c}'_{ij} = c'_{ij} + \pi_j{}^{(r)} - \pi_i{}^{(r)}$ is its updated reduced cost.

Indeed $c'_{ij} = c_{ij}$ if $(i, j) \notin K$ hence $\bar{c}'_{ij} \geq 0 \ \forall (i, j) \in A \backslash K$ and $\bar{c}'_{ij} = 0 \ \forall (i, j) \in A_r \backslash K$. Moreover, after the arc costs change, only one of the following cases can occur for an arc $(i, j) \in K$: 1. $\bar{c}'_{ij} < 0$ or 2. $\bar{c}'_{ij} \geq 0$. The former implies that $\pi^{(r)}$ is no more dual feasible; instead, when inequality 2. strictly holds for an arc in the current solution, that is $\bar{c}'_{ij} > 0$ for some $(i, j) \in A_r$, then $\pi^{(r)}$ no longer satisfies Equation (2.3b) and $\mathbf{T}^*_r$ is no more an optimal tree.

To update the optimal solutions, the algorithm in [144] partitions K into two disjoint subsets:

$$K^+ = \{(i, j) \in K : \bar{c}'_{ij} \geq 0\} \quad \text{and} \quad K^- = \{(i, j) \in K : \bar{c}'_{ij} < 0\}.$$

Afterwards, it reoptimizes firstly with respect to the arcs in K^+ and then considering those in K^-. If $K^+ \neq \emptyset$ and $K^- = \emptyset$, the dual feasibility of $\pi^{(r)}$ holds, while when $K^- \neq \emptyset$, the feasibility has to be restored.

We have outlined in Algorithm 1 the operations performed by the above mentioned algorithm.

Algorithm 1 Reoptimization Procedure

```
1:  procedure REOPT
2:      Update the reduced costs.
3:      Partition K in K+ and K−.
4:      if K+ ≠ ∅ then
5:          Dual_Phase                          ▷ Restore c̄'_ij = 0   ∀(i, j) ∈ A_r.
6:      if K− = ∅ return.                       ▷ The current tree is optimal.
7:      else
8:          while K− ≠ ∅ do
9:              Primal_Phase                    ▷ Restore dual feasibility of π^(r).
10:             Update the set K−.
11:         end while
12: end procedure
```

Specifically, if $K^+ \neq \emptyset$ (Line 4) then only the cost changes of its arcs are applied to the graph. At this purpose, the `Dual_Phase` procedure at Line 5, described in Algorithm 2, is meant to restore CS and it is executed on the graph $G' = (N, A, \bar{c}')$ having the same sets of nodes and arcs of G and a cost function that assigns the modified reduced cost to each arc. This phase of reoptimization can be performed either via a label setting procedure (e.g. Dijkstra's algorithm) or with a *dual-hanging approach*, with the theoretical computational complexity of the two methods remaining comparable [144]. As a consequence, we chose to implement the dual phase with the former approach (Line 5, Algorithm 2), so as to remain consistent with the ongoing investigation.

While K^- is nonempty, the `Primal_Phase` procedure at Line 9, described in Algorithm 3, runs on the subgraph of G formed by all the arcs with a non-positive reduced cost, i.e $G_0^- = (N, A_0^-)$ where $A_0^- = \{(i, j) \in A : \bar{c}'_{ij} \leq 0\}$. Clearly, $A_r \subset A_0^-$ and $K^- \subset A_0^-$; in particular, this graph could be either acyclic or at most it could contain zero reduced cost cycles. In fact, by assumption no negative cost cycle is created in G after the arc cost changes; anyway each cycle in G_0^- can be unfolded by removing one arc from it.

Through the execution of the *primal phases*, the set K^- is dynamically decomposed into h disjoint subsets K_j^- $j = 1, \ldots h$ so that the current solution $\mathbf{T}_r^*$ is reoptimized – with respect to the arcs in each subset – via a single call to any implementation of a label setting procedure. Indeed, each K_j^- induces a *Dijkstrable subgraph*, i.e. a subgraph of G_0^- verifying the Definition 2.3.1. In particular, Fig. 2.2 depicts an example of Dijkstrable subgraph.

Definition 2.3.1 ([144])**.** A subgraph of G_0^- is called *Dijkstrable* if its set of arcs can be reoptimized via a single call to a Dijkstra's permanent procedure [see 88].

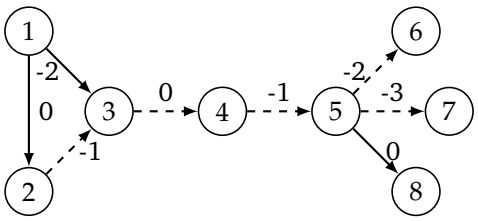


Fig. 2.2 Directed graph with eight nodes and eight arcs. Dashed arcs form a Dijkstrable subgraph.

Algorithm 3 provides the outline of a generic *primal phase* [see 144]: a Dijkstrable subgraph is determined and the label of each node is initialized to zero (Line 2-4); then, the Dijkstrable subgraph is visited in topological order so as to update the labels of its nodes to a negative value (Line 6). Finally, the reoptimization is performed via *Dijkstra's Permanent Procedure* [see 88] (Line 7), whose input is represented by the nodes with negative label.

Algorithm 4 outlines the operation of this procedure: *Bellman's optimality condition* (Line 5) is checked for each arc in the forward star of the node with

Algorithm 2 Dual Phase

```
1: procedure DUAL_PHASE
2:     if c̄'_ij = 0 ∀(i, j) ∈ A_r return          ▷ Complementary Slackness holds.
3:     else
4:         Change only the costs of the arcs in K+.
5:         Call Dijkstra on G' = (N, A, c̄').
6:         Update dual vector π^(r) and set K−.
7: end procedure
```

minimum (negative) label. The updating of the label at Line 6 propagates the negativity of the labels in the graph.

Remark 2. Anytime a reoptimization phase is executed, there occur the updates of the current optimal solution $\mathbf{T}_r^*$ and of the corresponding dual vector $\pi^{(r)}$ (Line 6, Algorithm 2 and Line 8, Algorithm 3). Indeed, also the reduced costs are adjusted: as a consequence, at the end of the `Dual_Phase` procedure, some arcs may have to be removed from K^- (Line 6, Algorithm 2) since their (modified) reduced cost is no more negative. Moreover, at the end of the j^{th} primal phase, other arcs apart those of K_j^- may have to be removed from K^- (Line 10, Algorithm 1). □

Algorithm 3 Primal Phase [144]

```
1: procedure PRIMAL_PHASE
2:     G(K_j^-) = Dijkstrable.                  ▷ G(K_j^-) = (N(K_j^-), A(K_j^-))
3:     Change the costs of the arcs in K_j^- = A(K_j^-) ∩ K−.
4:     Set d[i] = 0 ∀i ∈ N.
5:     Visit G(K_j^-) in topological order and set
6:         d[j] = min{d[i] + c̄'_ij : (i, j) ∈ BS(j) ∩ A(K_j^-)}.
7:     Dijkstra_permanent_procedure.
8:     Set π_i^(r) = π_i^(r) + d[i] ∀i ∈ N.      ▷ Update dual vector.
9: end procedure
```

Remark 3. There exists a family of subgraphs which are indeed Dijkstrable (see Definition 2.3.2) [144].

Definition 2.3.2 ([144]). *A starpath centred at i and j* is a subgraphs of G_0^- consisting of a directed path from node i to node j plus the following stars:

$$\overline{BS}(i) = BS(i) \cap K^- \quad \text{and} \quad \overline{FS}(j) = FS(j) \cap K^-.$$

Algorithm 5 reports the procedure devised in [144] to build a starpath: a depth first search (DFS) is performed on the graph G_0^- starting from r (Line 2) until a leaf node of the current optimal tree is reached and there is at least one arc belonging to K^- in the path from the root to that node. Denoting with (j, u) the last arc of K^- along this path, the centre node i of the starpath is determined (Line 6) following up the predecessors set up by the DFS. □

Algorithm 4 Dijkstra Permanent Procedure [88]

```
1: procedure DIJKSTRA_PERMANENT_PROCEDURE
2:     while Q ≠ ∅ do                          ▷ Q contains nodes with negative label.
3:         i =extract_min(Q)
4:         for (i, j) ∈ FS(i)
5:             if d[i] + c̄'_ij < d[j] then
6:                 d[j] = d[i] + c̄'_ij
7:                 pred[j] = i
8:                 Q = Q ∪ {j}
9:     end while
10: end procedure
```

Algorithm 5 Build a Dijkstrable subgraph [144]

```
1: procedure DIJKSTRABLE
2:     Depth First Search of G_0^- starting from r.
3:     if v is a leaf of T*_r then
4:         if there is at least one arc of K^- in the path from r to v then
5:             return
6:     Determine the centre node i.                ▷ See Definition 2.3.2.
7: end procedure
```

2.3.1 Computational Complexity

Algorithm 1 is strongly based on "Dijkstra-like" procedures; as a consequence, the computational complexity resembles that of Dijkstra's algorithm.

In particular, let C_d denote the perturbation due to the arcs in K^+ and let C_p be the sum of the minimum (negative) labels that can be generated during the execution of each primal phase. Formally,

$$C_d = \max\Big\{ \sum_{(i,j)\in P} \bar{c}'_{ij} | P \text{ is a path in } \mathbf{T}^*_r \Big\} \quad \text{and} \quad C_p = -\sum_{(i,j)\in K^-} \bar{c}'_{ij}.$$

When using Dial's implementation [54, 55] the computational complexity of Dijkstra's algorithm is $O(m + nCmax)$, with $Cmax = \max_{(i,j)\in A} c_{ij}$. Thus, the computational complexity of the dual phase is $O(m+C_d)$. Instead, the overall time complexity of the primal phases is equal to $O(|K^-|m + C_p)$ since the construction of a Dijkstrable subgraph (Algorithm 5) takes $O(m)$ and the number of arcs in K^- is an upper bound for the total number of primal phases.

In conclusion, the overall complexity of Algorithm 1 is $O(|K^-|m + C_p + C_d)$.

2.4 Computational Experiments

Since Pallottino and Scutellá [144] only provided a theoretical description of the primal-dual reoptimization framework, we implemented this algorithm by

devising ad hoc data structure [see 83]. Besides, we compared the performance of our proposal with Dijkstra's well-known algorithm, adopted for solving each modified problem from scratch.

In this Section, we analyze the behaviour of the reoptimization algorithm (which we will refer to as `Reopt`) comparing it with Dijkstra's label setting procedure (i.e., `Dijkstra`) used for a resolution from scratch. This comparison is performed in order to identify the features which make an instance of SPT more suitable for the reoptimization than for a classic resolution when a subset of arcs is given a new cost.

2.4.1 Implementation Details

Both `Reopt` and `Dijkstra` algorithms have been coded in C, compiled with gcc 8.3.0 and tested by using an Intel® core™ i7-5500U, 2.40 GHz, RAM 8.00 GB, under a Ubuntu 19.04 operating system.

The Multi Level Bucket (MLB) structure [39, 40], designed ad hoc for Dijkstra's algorithm, was used in our `Dijkstra`, since it has been proved to be robust while the 1-Level Bucket Structure is advisable only for instances with small arc lengths [92]. Furthermore, due to its features and with the aim of making a fair comparison among the algorithms, this structure was adapted for the implementation of both the `Dual_Phase` and the `Dijkstra_Permanent_Procedure` in `Reopt`.

The MLB is nothing more than a *priority queue* consisting of l *levels*, each containing $p = \left\lceil C^{1/l} \right\rceil$ *buckets*, where C is an integer constant related to the specific problem instance[1]. In addition, depending on the level to which a bucket belongs, the labels of the elements, the nodes in our case, in that bucket lie in a interval of integers. Finally, the elements are stored in *double linked list*, thus allowing to perform the *insertion* and *deletion* operations in constant time.

Specifically, during the execution of a label setting procedure, an *active bucket* is maintained, which consists of the first nonempty bucket at the lowest level and contains all nodes with the smallest label. Thus, at each iteration, a node is removed from the active bucket and the labels of its neighbours are updated. As a consequence, the nodes whose distance has been decreased are reallocated either in the lowest level or in a higher level bucket. When the active bucket becomes empty, the algorithm makes active the first nonempty bucket in the first nonempty level and the *expansion* of that bucket is performed in constant time, i.e. its elements are relocated to lower levels until there is a new, nonempty active bucket at the lowest level.

2.4.2 Test Problems

The instances we generated can be divided in the following five families[2]:

[1] We used the maximum arc cost in the graph instance as value for C [see 92].

[2] The full data-set can be found at `https://figshare.com/articles/INSTANCES/10115900`.

1. GTC-instances. Four different typologies of network obtained using the generator described in Festa and Pallottino [80]: `grid`, `full-grid`, `cylinder`, `torus`. They have a skeleton structure consisting of a two dimensional grid $G = (V, A)$ where the set $V = \{1, 2, \ldots, n^2\}$ is made of n levels of n nodes. Each node is connected to those adjacent; in addition, for all typologies except the `grid` one, there are supplementary bilateral connections. We briefly outline the features of these networks, as done in Napoletano [138]. The set of nodes is $V = CN \cup LN \cup IN$ where:

CN contains the *corner-nodes* $\{1, n, n^2 - n + 1, n^2\}$;

LN contains the *lateral-nodes*, i.e. $LN = \{v\colon 1 < v < n, n^2 - n + 1 < v < n^2,$ $\mod(v, n) = \{0, 1\}\} \backslash CN$;

IN contains the *inner-nodes* which are all the remaining nodes.

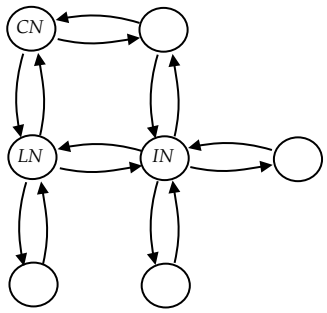
In `grid` networks: $|BS(v)| = |FS(v)| = 2\ \forall v \in CN$, $|BS(v)| = |FS(v)| = 3$ $\forall v \in LN$ and $|BS(v)| = |FS(v)| = 4.\ \forall v \in IN$.
The supplementary bilateral connections in `full-grid` are defined as follows: i) $\forall v \in CN$, the set A contains the arcs (u, v) where $v - u = 0 \mod (n+1)$ if either $v = 1$ or $v = n^2$, and $v - u = 0 \mod (n-1)$ otherwise; ii) $\forall v \in LN$, A contains the arcs (u, v) with $v - u = 0 \mod (n-1)$ and $v - u = 0 \mod (n+1)$. As a consequence: $|BS(v)| = |FS(v)| = 3, \forall v \in CN$; $|BS(v)| = |FS(v)| = 5, \forall v \in LN$; $\forall v \in IN, |BS(v)| = |FS(v)| = 8$.
In `cylinder` networks, $CN = \{\emptyset\}$ and $LN = \{v\colon 1 < v < n, n^2 - n + 1 < v < n^2\}$. Besides, the supplementary bilateral arcs are defined for each pair of nodes $u, v \in V$ such that $v = u + (n-1) \mod (u, n) = 1$.
Finally, for `torus` networks both CN and LN are empty and the supplementary bilateral arcs are defined $\forall u, v \in V$ such that $v = n^2 - n + u$ where $1 < u < n$. Hence $\forall v \in V\ |BS(v)| = |FS(v)| = 4$.
Fig. 2.3 provides an example of each topology when $n = 3$.

The arc costs are chosen independently and uniformly at random in the set $\{1, 2, \ldots Cmax\}$ where $Cmax$ is given as input. We generated networks of increasing dimensions setting $n = 100, 400, 700$ in order to represent "small", "medium" and "large" sized instances.

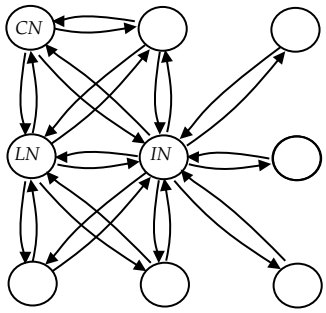
2-3. GRIDGEN-instances. Obtained with Bertsekas' generator [16, 17], which allows to create square and rectangular grids, with or without additional arcs between randomly chosen nodes, where each node is connected to four neighbours and to two special nodes, *source* and *sink*. Grids of increasing dimensions have been generated to represent "small", "medium" and "large" sized instances. They can be grouped as:

`SQUARE`. Square $n \times n$ grids with $n \in \{100, 400, 700\}$. We generated: the standard skeleton networks with $6n^2$ arcs (`square`n), the ones with 25% and 50% additional arcs (`square`n`+25`, and `square`n`+50`).

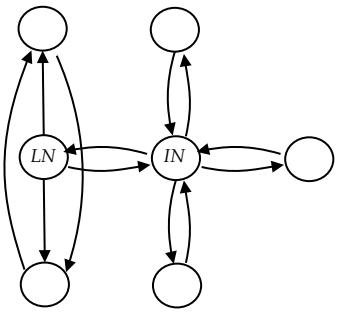
`RECT`. Rectangular grids with $n \times 2n$ nodes with $n \in \{100, 300, 500\}$. We generated: the standard skeleton networks with $6(n \times 2n)$ arcs (`rect`n), those with 25% and 50% additional arcs (`rect`n`+25` and `rect`n`+50`).



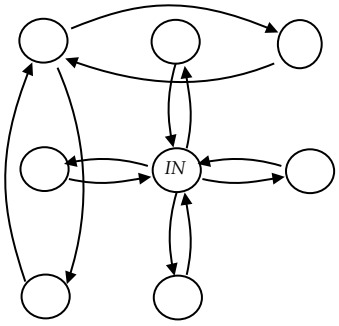
(a) Partial structure of a 3×3 `grid`.



(b) Partial structure of a 3×3 `full-grid`.



(c) Partial structure of a 3×3 `cylinder`.



(d) Partial structure of a 3×3 `torus`.

Fig. 2.3 Example of GTC topologies.

Cmax is the cost of the arcs connecting a node with its four neighbours; while the other costs are chosen at random in the set $\{1, 2, \ldots Cmax\}$.

4-5. SPACYC and SPRAND instances. They were obtained using Cherkassky's SPACYC and SPRAND generators [39]: the former creates acyclic random networks, the latter strongly connected random networks.
Since the average cardinality of the forward stars of GTC instances is 4 and that of the GRIDGEN ones is 6, we generated three typologies of SPACYC networks and three typologies of SPRAND networks : spacyc$n$-4n and sprand$n$-4n with n^2 nodes and $4n^2$ arcs, spacyc$n$-6n and sprand$n$-6n with n^2 nodes and $6n^2$ arcs, spacyc$n$-8n and sprand$n$-8n with n^2 nodes and $8n^2$ arcs. The tests were carried out setting $n = 100, 400, 700$ to have "small", "medium" and "large" sized instances; the arc costs are chosen independently and uniformly at random in the set $\{1, 2, \ldots Cmax\}$.

2.4.3 Experiments Setup and Evaluation Metrics

We devised two kinds of perturbation for the cost function in order to appraise the performance of the reoptimization approach both when only the set K^+ is nonempty and when the execution of at least one primal phase is required to update the current solution. Specifically, the following perturbations were applied separately on each instance graph:

INCREASE: for each arc $(i, j) \in K$, the modified cost is equal to $c_{ij} + D$ where D is an integer chosen at random in the set $\{1, \ldots, Cmax\}$;

DECREASE: for each arc $(i, j) \in K$, the modified cost is equal to R where R is an integer chosen at random in the set $\{1, 2, \ldots, c_{ij}\}$.

On the one hand, when the `INCREASE` is performed, only the execution of the dual phase is needed. On the other hand, the structure of the `DECREASE` modification ensures that K^- is nonempty, since its definition allows to consider a reduction in the range $[0, c_{ij} - 1]$ for the arc cost.

The changes were applied to the arcs in the forward star of k randomly chosen nodes, where $k = 1, 2, \ldots 10$. Actually, an earlier experimentation was conducted by perturbing the costs for a given number of randomly chosen arcs. However, these changes generated instances on which the reoptimization was not needed: in fact, either the cost increase occurred for arcs not belonging to the current solution or the cost decrease was not enough to affect the optimality of the current solution. Instead, the optimal tree is always involved when changing the cost of the arcs in the forward stars. Furthermore, this choice is motivated also by practical applications: in general, the changes may be localized only in small portions of the networks.

For any choice of the number of nodes we generated five instances, one for each value of $Cmax$ in the set $\{75, 150, 280, 560, 1125\}$. As a consequence, we carried out the testing on 60 `GTC` instances, 90 `GRIDGEN` instances, 45 `SPACYC` and 45 `SPRAND` instances. Table 2.1 summarizes the total number of test problems.

We considered the *time ratio* of `Reopt` (i.e., `time-ratio`) in order to give a measure of the performance of the algorithm in comparison with that obtained with Dijkstra's approach:

$$\texttt{time-ratio} = \frac{\texttt{Reopt running time}}{\texttt{Dijkstra running time}}. \tag{2.5}$$

Let us denote with I^x_{Cmax} the set of instances of typology x with maximum cost equal to $Cmax$. Hence, the value in Equation (2.6) is the `time-ratio` on an instance $l \in I^x_{Cmax}$ when $k \in \{1, \ldots 10\}$ forward stars are affected by the cost perturbation.

$$(TR^l_{Cmax})_k. \tag{2.6}$$

In addition, as k varies from 1 to 10, we calculated the Relative Standard Deviations (RSD) of the values in Equation (2.6) in order to measure the dispersion of the their probability distribution. When the RSDs confirmed that this dispersion

Table 2.1 Numbers of INCREASE and DECREASE instances for each family of test problems and total dimension of the test set.

Family	Typology	Nodes	Arcs	#INCR	#DECR	Tot.
GTC	grid	10k to 490k	40k to 1.96M	150	150	300
	full-grid		80k to 3.92M			300
	cylinder		40k to 1.96M			300
	torus		40k to 1.96M			300
SQUARE	square	10k to 490k	60k to 2.94M	150	150	300
	square+25		75k to 3.67M			300
	square+50		90k to 4.41M			300
RECT	rect	20k to 500k	120k to 3.00M	150	150	300
	rect+25		150k to 3.75M			300
	rect+50		180k to 4.50M			300
SPACYC	spacyc-4n	10k to 490 k	40k to 1.96M	150	150	300
	spacyc-6n		60k to 2.94M			300
	spacyc-8n		80k to 3.92M			300
SPRAND	sprand-4n	10k to 490 k	40k to 1.96M	150	150	300
	sprand-6n		60k to 2.94M			300
	sprand-8n		80k to 3.92M			300
						4800

is negligible, we assumed the average over k of $(TR^l_{Cmax})_k$ to be a good indicator of time-ratio on the instance $l \in I^x_{Cmax}$.

Finally, as an indicator of the performance trend on a typology x with a given $Cmax$, we used the average of the average time ratios (defined in Equation (2.7)), shortly referred to as "average time-ratio" .

$$\texttt{AvgTR}^x_{Cmax} = \frac{\displaystyle\sum_{l \in I^x_{Cmax}} \left(\frac{\sum_{k=1}^{10} (TR^l_{Cmax})_k}{10} \right)}{|I^x_{Cmax}|}. \tag{2.7}$$

2.4.4 Results

The results of the computational experiments have been divided according to the type of perturbation applied to the graph: Tables 2.2 to 2.9 refer to the INCREASE perturbation and Tables 2.10 to 2.15 refer to the DECREASE one. In particular, the "AvgTR" values have been computed according to Equation (2.7) while the field "#Wins" contains the number of instances for which Reopt wins Dijkstra, namely it presents a lower running time[3].

Appendix 2.B reports the average computation times of both the algorithms, while a graphical representation of the results is presented in Appendix 2.A.

[3]Data of the executed experiments are available at https://figshare.com/articles/dataset/RESULTS/10129223.

Table 2.2 (GTC-INCREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

Cmax	#Wins	AvgTR	#Wins	AvgTR
	grid		full-grid	
75	29/30	0.54 ± 79.0%	30/30	0.44 ± 33.8%
150	28/30	0.70 ± 33.2%	30/30	0.75 ± 5.3%
280	15/30	1.00 ± 20.9%	3/30	1.19 ± 9.6%
560	3/30	1.15 ± 16.6%	2/30	1.51 ± 29.8%
1125	1/30	1.67 ± 20.9%	0/30	2.40 ± 39.3%
	cylinder		torus	
75	30/30	0.55 ± 20.4%	30/30	0.26 ± 51.1%
150	21/30	0.87 ± 28.0%	30/30	0.38 ± 53.5%
280	2/30	1.25 ± 8.9%	28/30	0.57 ± 40.5%
560	4/30	1.20 ± 8.4%	28/30	0.71 ± 28.5%
1125	1/30	1.73 ± 21.3%	1/30	1.15 ± 7.4%

Finally, we performed a further experimentation on a subset of instances in order to appraise the performance of Dijkstra and Reopt when both the perturbations occur on the network. We will refer to that kind of change as RANDOM perturbation.

INCREASE perturbation

▷ GTC-instances. The computational results are reported in Table 2.2-Table 2.3.

On these networks, the values of time-ratio increase with that of *Cmax* i.e. the running time of Reopt becomes comparable with (or even higher than) the one of Dijkstra. Moreover, as appears clearly from the plot in Fig. 2.7, the algorithm outperforms Dijkstra on all the four topologies when either $Cmax = 75$ or $Cmax = 150$; this result holds for grid networks if *Cmax* is increased to 280 and for torus ones when $Cmax = 560$.

This inversion of the performance trend depends on the perturbation applied to the graph and on the specific characteristics of the MLB data structure: in fact, the running time of Dijkstra's algorithm with this structure is linked to *Cmax* [39]; then, since the INCREASE perturbation affects both the original costs and the reduced ones, the increase of the maximum reduced cost implies the presence of a higher number of nonempty buckets to *unroll*, namely expand, during the dual phase. These operations, make Dijkstra performing better than Reopt as *Cmax* increases. Nevertheless, we point out that a computational effort is required to check whether the CS hold for all the arcs in the current solution, after the cost changes.

Since the RSDs for the AvgTR values are relevant for most of the considered networks, (see Table 2.2), we considered also the average over k of the time-ratio (Equation (2.6)) for each instance, with the aim of analysing how

the size of the network influences the performance trend (Table 2.3). This analysis revealed that on `torus` networks, as the size increases, the greater *Cmax* is, the smaller is the effect of its increase on the running time of `Reopt`. A similar trend emerges for the other instances of the family only when $Cmax \leq 150$. We relate this somehow regular behaviour to the particular topology of `torus` networks.

▷ `GRIDGEN`-instances. The computational results are given in Tables 2.4 to 2.7. Specifically, the first column indicates the value of *Cmax* considered, the second, third and forth pair of columns summarize the results obtained.

Actually, the results in Table 2.4 and Table 2.5 and the scatter plots in Fig. 2.8a and Fig. 2.8b show a similar trend for the AvgTR values. This means that the "shape" of the grid has no influence on the performance of the two algorithms.

The comparison of the running times revealed that the addition of the 25% of arcs affects the performance of `Dijkstra` more significantly; instead, the 50% additional arcs has a major impact on those of `Reopt`. This is due to the costs distribution: in the skeleton grids, i.e. those without additional arcs, the probability of finding an arc with cost equal to *Cmax* is about 0.70. Therefore, although we perform a slight perturbation, the presence of arcs in K^+ leads to an elevated number of buckets to unroll for `Reopt`. In the case of 25% additional arcs, the reoptimization becomes advisable since for the same *Cmax*, there are many more possible values – if compared with those relative to the skeleton grids – for the labels generated during Dijkstra's procedure; therefore unrolling the buckets in `Dijkstra` takes more time. The performance of `Reopt` is affected in a similar way when there is a 50% additional arcs.

Due to the significant RSDs associated to the AvgTR for these networks, also for them we conducted an analysis by size, given in Tables 2.6 and 2.7: it simply reveals that the reoptimization is advisable on the small sized instances of the two families without additional arcs and with a 50% additional arcs.

▷ `SPACYC`-instances. The results in Table 2.8 and in Fig. 2.8c prove that when the density of the graph increases, the performance of `Reopt` becomes comparable with (or even worse than) those of `Dijkstra`. Indeed, also for this family of problems, there is a threshold value for *Cmax* indicating when the resolution from scratch is more efficient. Specifically, it decreases from 280 to 75 as the density of the networks increases from $1 \cdot 10^{-4}$ to $3 \cdot 10^{-4}$.

Although the increase in the number of arcs is supposed to influence similarly the running time of both algorithms, the performance of `Reopt` is more affected. In fact, as observed for `GTC` instances, when the `INCREASE` perturbation occurs, the number of nonempty buckets to unroll during the dual phase increases.

Finally we have not included the analysis by size, because the related results have not added any information to what has already been observed. Probably, this specific behaviour can be explained by observing that the acyclic networks are not characterized by a specific structure, unlike the networks considered so far. It is worth to mention that the RSD of the average `time-ratio` for `SPACYC` instances is on average equal to 8.7%.

Table 2.3 (GTC-INCREASE.) Computational results in terms of number of wins for Reopt, and AvgTR. Instances classified by size: block $n = 100/400/700$ refers to "small"/"medium"/"large" instances.

	grid		full-grid		cylinder		torus	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
				$n = 100$				
75	9/10	0.86 ± 18.2%	10/10	0.61 ± 9.5%	10/10	0.87 ± 9.1%	10/10	0.49 ± 17.3%
150	9/10	0.92 ± 13.0%	10/10	0.71 ± 18.1%	2/10	1.11 ± 16.0%	10/10	0.62 ± 7.7%
280	5/10	1.13 ± 36.6%	0/10	1.13 ± 8.6%	2/10	1.13 ± 20.5%	8/10	0.84 ± 3.2%
560	3/10	1.01 ± 15.7%	2/10	1.14 ± 10.5%	4/10	1.09 ± 31.3%	8/10	0.93 ± 22%
1125	1/10	1.27 ± 12.4%	0/10	1.33 ± 5.3%	1/10	1.31 ± 18.6%	0/10	1.24 ± 22.8%
				$n = 400$				
75	10/10	0.40 ± 4.9%	10/10	0.34 ± 6.6%	10/10	0.38 ± 4.8%	10/10	0.16 ± 8.5%
150	9/10	0.71 ± 50.5%	10/10	0.78 ± 11.4%	9/10	0.86 ± 11.1%	10/10	0.29 ± 12.1%
280	10/10	0.76 ± 6.7%	3/10	1.11 ± 13.6%	0/10	1.32 ± 6.2%	10/10	0.48 ± 10.3%
560	0/10	1.07 ± 6.7%	0/10	1.38 ± 4.0%	0/10	1.29 ± 6.8%	10/10	0.66 ± 10.1%
1125	0/10	1.92 ± 4.7%	0/10	2.73 ± 7.0%	0/10	1.97 ± 5.7%	0/10	1.15 ± 6.7%
				$n = 700$				
75	10/10	0.37 ± 5.4%	10/10	0.37 ± 6.4%	10/10	0.39 ± 6.1%	10/10	0.12 ± 3.2%
150	10/10	0.46 ± 12.7%	10/10	0.77 ± 12.5%	10/10	0.62 ± 12.2%	10/10	0.24 ± 10.8%
280	0/10	1.12 ± 6.8%	0/10	1.32 ± 3.6%	0/10	1.32 ± 7.8%	10/10	0.40 ± 3.1%
560	0/10	1.37 ± 12.7%	0/10	2.02 ± 2.8%	0/10	1.23 ± 5.0%	10/10	0.54 ± 3.3%
1125	0/10	1.81 ± 5.2%	0/10	3.13 ± 4.7%	0/10	1.93 ± 6.8%	1/10	1.07 ± 8.4%

Table 2.4 (SQUARE-INCREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

	square		square+25		square+50	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	30/30	0.60 ± 28.3%	30/30	0.50 ± 26.7%	30/30	0.52 ± 13.2%
150	20/30	0.96 ± 58.3%	20/30	0.80 ± 29.6%	10/30	1.30 ± 28.7%
280	11/30	1.16 ± 66.2%	29/30	0.80 ± 20.6%	10/30	1.62 ± 69.3%
560	20/30	0.95 ± 85.5%	30/30	0.67 ± 26.4%	30/30	0.52 ± 56.6%
1125	10/30	1.23 ± 50.3%	30/30	0.77 ± 11.8%	10/30	1.12 ± 29.9%

Table 2.5 (RECT-INCREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

	rect		rect+25		rect+50	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	30/30	0.73 ± 13.0%	30/30	0.48 ± 16.5%	30/30	0.45 ± 12.7%
150	10/30	1.34 ± 38.0%	29/30	0.76 ± 27.8%	9/30	1.29 ± 26.2%
280	10/30	1.33 ± 54.9%	19/30	0.99 ± 24.9%	10/30	1.75 ± 70.5%
560	30/30	0.53 ± 13.7%	30/30	0.51 ± 32.7%	30/30	0.45 ± 39.4%
1125	10/30	1.24 ± 45.1%	29/30	0.89 ± 10.9%	10/30	1.11 ± 22.1%

Table 2.6 (SQUARE-INCREASE). Computational results in terms of number of wins for Reopt, and AvgTR. Instances classified by size: block n = 100/400/700 refers to "small"/"medium"/"large" instances.

	square		square+25		square+50	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
			n = 100			
75	10/10	0.60 ± 5.1%	10/10	0.37 ± 6.8%	10/10	0.47 ± 5.0%
150	10/10	0.55 ± 2.5%	0/10	1.04 ± 1.6%	10/10	0.92 ± 4.9%
280	10/10	0.45 ± 4.2%	9/10	0.95 ± 4.2%	10/10	0.35 ± 1.8%
560	10/10	0.36 ± 6.1%	10/10	0.79 ± 7.6%	10/10	0.26 ± 4%
1125	10/10	0.52 ± 7.0%	10/10	0.84 ± 4.1%	10/10	0.74 ± 4.3%
			n = 400			
75	10/10	0.77 ± 5.9%	10/10	0.49 ± 6.7%	10/10	0.48 ± 10.0%
150	0/10	1.60 ± 6.2%	10/10	0.58 ± 5.5%	0/10	1.32 ± 4.2%
280	0/10	1.97 ± 4.2%	10/10	0.83 ± 5.0%	0/10	2.02 ± 5.9%
560	10/10	0.61 ± 2.9%	10/10	0.76 ± 10.0%	10/10	0.84 ± 5.5%
1125	0/10	1.53 ± 5.8%	10/10	0.81 ± 6.1%	0/10	1.26 ± 8.8%
			n = 700			
75	10/10	0.43 ± 3.9%	10/10	0.63 ± 4.6%	10/10	0.59 ± 5.2%
150	10/10	0.73 ± 8.3%	10/10	0.77 ± 6.4%	0/10	1.67 ± 6.9%
280	1/10	1.05 ± 3.7%	10/10	0.62 ± 4.4%	0/10	2.49 ± 3.5%
560	0/10	1.88 ± 6.1%	10/10	0.47 ± 3.3%	10/10	0.47 ± 5.1%
1125	0/10	1.63 ± 5.8%	10/10	0.67 ± 4.5%	0/10	1.37 ± 1.7%

Table 2.7 (RECT-INCREASE). Computational results in terms of number of wins for Reopt, and AvgTR. Instances classified by size: block n = 100/300/500 refers to "small"/"medium"/"large" instances.

	rect		rect+25		rect+50	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
			$n = 100$			
75	10/10	0.64 ± 2.8%	10/10	0.40 ± 6.5%	10/10	0.40 ± 6.2%
150	10/10	0.76 ± 1.2%	9/10	0.99 ± 0.6%	9/10	0.93 ± 4.7%
280	10/10	0.49 ± 0.0%	0/10	1.27 ± 3.3%	10/10	0.34 ± 9.9%
560	10/10	0.44 ± 2.4%	10/10	0.32 ± 0.0%	10/10	0.25 ± 1.3%
1125	10/10	0.59 ± 3.3%	9/10	0.98 ± 1.1%	10/10	0.83 ± 4.0%
			$n = 300$			
75	10/10	0.74 ± 13.8%	10/10	0.48 ± 9.6%	10/10	0.44 ± 9.2%
150	0/10	1.53 ± 4.4%	10/10	0.58 ± 7.2%	0/10	1.34 ± 10.8%
280	0/10	1.82 ± 3.5%	9/10	0.82 ± 13.1%	0/10	2.54 ± 6.6%
560	10/10	0.57 ± 5.5%	10/10	0.61 ± 9.8%	10/10	0.58 ± 9.8%
1125	0/10	1.54 ± 3.7%	10/10	0.80 ± 6.0%	0/10	1.22 ± 4.5%
			$n = 500$			
75	10/10	0.83 ± 5.4%	10/10	0.56 ± 4.9%	10/10	0.51 ± 3.2%
150	0/10	1.72 ± 3.5%	10/10	0.69 ± 4.0%	0/10	1.61 ± 2.9%
280	0/10	1.67 ± 3.0%	10/10	0.87 ± 7.0%	0/10	2.37 ± 1.5%
560	10/10	0.57 ± 1.9%	10/10	0.61 ± 2.2%	10/10	0.52 ± 4.2%
1125	0/10	1.58 ± 3.5%	10/10	0.87 ± 3.9%	0/10	1.28 ± 4.5%

Table 2.8 (SPACYC-INCREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

	spacyc-4n		spacyc-6n		spacyc-8n	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	20/30	0.83 ± 73.5%	13/30	0.97 ± 34.9%	20/30	0.99 ± 28.8%
150	21/30	0.74 ± 60.1%	11/30	1.14 ± 32.4%	6/30	1.29 ± 29.1%
280	19/30	0.86 ± 48.0%	20/30	0.95 ± 9.4%	3/30	1.34 ± 30.5%
560	14/30	0.99 ± 24.9%	11/30	1.18 ± 32.4%	16/30	1.18 ± 32.9%
1125	20/30	0.99 ± 49.5%	7/30	1.29 ± 23.9%	2/30	1.71 ± 30.3%

Table 2.9 (SPRAND-INCREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

	sprand-4n		sprand-6n		sprand-8n	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	30/30	0.53 ± 35.7%	30/30	0.39 ± 66.6%	30/30	0.47 ± 52.4%
150	30/30	0.47 ± 28.9%	30/30	0.58 ± 5.7%	30/30	0.58 ± 15.3%
280	30/30	0.48 ± 53.8%	29/30	0.67 ± 37.0%	30/30	0.61 ± 47.7%
560	30/30	0.33 ± 26.9%	30/30	0.47 ± 19.8%	28/30	0.60 ± 62.7%
1125	30/30	0.40 ± 29.7%	30/30	0.53 ± 3.4%	30/30	0.43 ± 9.9%

▷ SPRAND-instances. The computational results are given in Table 2.9 and Fig. 2.9a: Reopt is more efficient than Dijkstra on the whole sets of problems.

Furthermore, from Fig. 2.9a, it is evident that the time-ratio values only slightly differ. Actually this means that the increase in the number of arcs affects the performances of both the algorithms in the same way. Supposedly this is due to the lack of an intrinsic structure in the graph. Also for this family of test problems the analysis by size did not reveal any particular trend. Hence we have decided to not include the related results in this evaluation, though we underline that the average RSD for the average time-ratio is equal to 6.3%.

Summary of Results

On the basis of the computational results collected, when a a perturbation of type INCREASE is performed, the following final considerations can be drawn.

For GTC instances the AvgTR values (Equation (2.7)) increase with that of the maximum cost within the graph. Indeed, for each topology, a threshold value for *Cmax* was identified that separates the instances on which the reoptimization approach is efficient from those that should be solved from scratch.

For SQUARE and RECT networks, we found that the shape of the grid has no influence on the performance trends. Moreover, Reopt outperforms Dijkstra on all the grids with 25% additional arcs. Actually, the addition of the 25% of arcs affects the performance of Dijkstra more significantly; an opposite trend is observed in the case of 50% additional arcs.

Finally, the performance of the reoptimization procedure have been studied on random – acyclic and cyclic – networks. The threshold value for *Cmax* was detected for the former, while on the latter Reopt showed always the best behaviour. Probably, these outcomes are due to the absence of an intrinsic structure in the networks, unlike the other family of test problems.

DECREASE perturbation

When this perturbation is applied, both the sets K^+ and K^- can be nonempty; in this case, Reopt has to perform the dual phase and at least one primal phase.

Table 2.10 (torus-DECREASE). Computational results in terms of number of wins for Reopt, and AvgTR. Columns $n = 400/700$ refer to "medium"/"large" instances.

	$n = 400$		$n = 700$	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	9/10	0.63 ± 47.1%	10/10	0.56 ± 61.2%
150	9/10	0.79 ± 33.7%	10/10	0.59 ± 32.7%

Table 2.11 (SQUARE-DECREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

	square		square+25		square+50	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	24/30	0.92 ± 9.5%	30/30	0.67 ± 20.0%	30/30	0.74 ± 6.3%
150	10/30	1.26 ± 46.9%	22/30	0.78 ± 33.9%	4/30	1.37 ± 22.3%
280	12/30	1.00 ± 48.6%	28/30	0.71 ± 27.9%	10/30	1.41 ± 66.4%
560	29/30	0.80 ± 18.4%	30/30	0.81 ± 16.3%	20/30	0.93 ± 45.7%
1125	10/30	1.24 ± 45.5%	27/30	0.86 ± 13.8%	10/30	1.22 ± 26.2%

▷ GTC-instances. The AvgTR values were meaningless for these problems, since the related RSD value was on average equal to 0.46. Moreover, no regular trend was observed for the time-ratio values. There is only the evidence that Reopt outperforms Dijkstra on all the medium and large sized torus instances with $Cmax \leq 150$; Table 2.10 reports the number of wins and the AvgTR values for these networks.

Comparing the running times of the dual and primal phases, we found that in 82% of cases the execution of latter is the most time consuming. Specifically, we observed that the computational efforts of each primal phase does not depend on the graph topology, but on where the perturbation occurs: the further away from the root are the nodes incident to arcs with negative reduced costs, the higher the computational time required to build the Dijkstrable subgraph is. Indeed, the Depth First Search is the most time consuming phase of the procedure adopted to build the Dijkstrable subgraph.

▷ GRIDGEN-instances. Comparing the results in Tables 2.11 and 2.12 with the data in Tables 2.4 and 2.5 (and the relative scatter plots in Figures 2.8a, 2.8b and 2.9b, 2.9c) it is evident that the trend for the time-ratio values remains approximately the same. Hence, we analyzed the running times of the dual and primal phases: for all the considered problems, the execution of the dual phase is the most time consuming operation, as appears clearly from the average computation times of Tables 2.24 and 2.25. Therefore, all the observations made in subsection "INCREASE perturbation" apply also for the DECREASE perturbation.

Actually, the only set of problems on which the reoptimization approach

Table 2.12 (RECT-DECREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

	rect		rect+25		rect+50	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	22/30	0.94 ± 9.7%	30/30	0.64 ± 10.6%	30/30	0.62 ± 6.5%
150	10/30	1.38 ± 35.9%	25/30	0.74 ± 29.1%	8/30	1.37 ± 26.0%
280	10/30	1.11 ± 49.4%	20/30	0.86 ± 33.0%	10/30	1.49 ± 66.9%
560	28/30	0.84 ± 5.8%	29/30	0.83 ± 10.1%	20/30	0.94 ± 41.3%
1125	10/30	1.24 ± 46.5%	21/30	0.94 ± 10.4%	9/30	1.27 ± 26.4%

always behaves the best is represented by the grids with 25% additional arcs. Moreover, the structure of these networks ensures that the Depth First Search is not affected by the location in the graph of the arcs with negative reduced costs.

Also the analysis by size imitates the one relative to the INCREASE perturbation: the reoptimization approach is convenient on SQUARE and RECT small sized instances without additional arcs and with a 50% additional arcs. By way of example, we reported only the data relating to SQUARE family in Table 2.13.

Finally, the average RSD for the average time-ratio is equal to 7% for SQUARE instances and 7.4% for RECT instances.

▷ SPACYC-instances. The testing phase revealed the absence of a regular trend for the performance of Reopt on such test problems. Indeed, in Table 2.14 we reported the AvgTR values only for spacyc-6n networks, since the other ones have significant RSD.

As regards the running times of Reopt, an in-depth study revealed that those of the dual phase are determinant in terms of complexity, as emerges from the average times in Table 2.26. In particular, the execution of the primal phases is extremely rapid whatever the modification occurred on the graph. Therefore, we have concluded that the lack of an intrinsic structure for these networks implies that the computational effort needed to build a Dijkstrable subgraph is independent on the position of the arcs with negative reduced costs. Namely, that running time is not affected by the distance (in terms of number of arcs) between the root node and the nodes incident to arcs with negative reduced costs.

▷ SPRAND-instances. The results in Table 2.15 reveal that Reopt outperformes Dijkstra on the whole set of problems. Moreover, the comparison between the scatter plots in Figures 2.9a and 2.10 highlights that the AvgTR values only slightly differ, namely the performance trend is almost the same for both the perturbations.

This result was determined by the fact that, as for GRIDGEN instances, the most time consuming phase of Reopt is the dual one. In particular we retain that the lack of an intrinsic structure for the graph makes the Depth First Search extremely fast on these networks too.

Table 2.13 (SQUARE-DECREASE). Computational results in terms of number of wins for Reopt, and AvgTR. Instances classified by size: block n = 100/400/700 refers to "small"/"medium"/"large" instances.

	square		square+25		square+50	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
			$n = 100$			
75	10/10	0.82 ± 12.4%	10/10	0.59 ± 7.2%	10/10	0.74 ± 8.3%
150	10/10	0.58 ± 12.9%	2/10	1.08 ± 6.9%	4/10	1.02 ± 6.2%
280	10/10	0.51 ± 11.7%	8/10	0.93 ± 8.3%	10/10	0.36 ± 11.6%
560	10/10	0.64 ± 10.2%	10/10	0.94 ± 4.9%	10/10	0.79 ± 5.5%
1125	10/10	0.59 ± 8.1%	8/10	0.98 ± 11.8%	10/10	0.87 ± 7.8%
			$n = 400$			
75	6/10	0.96 ± 8.5%	10/10	0.60 ± 20.5%	10/10	0.70 ± 8.2%
150	0/10	1.56 ± 11.2%	10/10	0.58 ± 6.1%	0/10	1.50 ± 6.0%
280	0/10	1.48 ± 5.1%	10/10	0.68 ± 5.3%	0/10	1.70 ± 6.1%
560	10/10	0.81 ± 8.6%	10/10	0.81 ± 4.3%	0/10	1.40 ± 3.5%
1125	0/10	1.52 ± 10.7%	9/10	0.86 ± 7.7%	0/10	1.30 ± 3.6%
			$n = 700$			
75	8/10	0.97 ± 3.0%	10/10	0.83 ± 4.8%	10/10	0.79 ± 3.4%
150	0/10	1.65 ± 3.7%	10/10	0.69 ± 6.3%	0/10	1.59 ± 3.8%
280	2/10	1.01 ± 2.5%	10/10	0.54 ± 3.2%	0/10	2.16 ± 3.4%
560	9/10	0.93 ± 11.0%	10/10	0.68 ± 6.3%	10/10	0.59 ± 6.6%
1125	0/10	1.62 ± 2.9%	10/10	0.74 ± 13.3%	0/10	1.49 ± 5.7%

Table 2.14 (SPACYC-DECREASE). Computational results. The AvgTR values are given only for spacyc-6n instances.

	spacyc-4n	spacyc-6n		spacyc-8n
Cmax	**#Wins**	**#Wins**	**AvgTR**	**#Wins**
75	13/30	3/30	1.67 ± 32.6%	1/30
150	12/30	5/30	1.69 ± 46.1%	0/30
280	10/30	11/30	1.38 ± 32.9%	2/30
560	10/30	8/30	1.59 ± 36.0%	0/30
1125	16/30	7/30	1.57 ± 38.5%	0/30

Table 2.15 (SPRAND-DECREASE). Computational results in terms of number of wins for Reopt, and AvgTR.

	sprand-4n		sprand-6n		sprand-8n	
Cmax	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**	**#Wins**	**AvgTR**
75	30/30	0.47 ± 36.2%	30/30	0.47 ± 63.8%	28/30	0.52 ± 69.2%
150	30/30	0.57 ± 41.4%	30/30	0.70 ± 15.5%	30/30	0.70 ± 2.3%
280	28/30	0.48 ± 59.2%	29/30	0.65 ± 32.7%	27/30	0.61 ± 55.0%
560	30/30	0.45 ± 38.7%	30/30	0.53 ± 33.2%	23/30	0.62 ± 69.9%
1125	30/30	0.49 ± 41.9%	29/30	0.68 ± 16.5%	30/30	0.64 ± 11.0%

Summary of Results

On the basis of the computational results collected, when the DECREASE perturbation of is applied, the following final considerations can be drawn.

The computational experiments on GTC instances revealed that a resolution from scratch has to be preferred. Moreover, the construction of a Dijkstrable subgraph is the most time consuming operation for Reopt: its running time strongly depends on the location of the arcs whose cost violates dual feasibility.

The results on SQUARE and RECT problems are similar to those of the INCREASE perturbation: whatever the value of *Cmax*, the reoptimization approach is advisable on the grids with 25% additional arcs. Moreover, the shape of the grid has no influence on the performance trend.

Finally, the absence of an intrinsic structure within the random networks, ensures that the computational effort needed to build a Dijkstrable subgraph is independent on the *depth* of the perturbation. Namely, by depth we refer to the distance – in terms of number of arcs – between the root and the nodes that are incident to arcs with negative reduced costs. However, on SPACYC networks no regular trend for the performance was observed, while Reopt overcomes Dijkstra on all the SPRAND instances.

As a final remark, we underline that the analysis we conducted – by considering the cost changes of arcs in forward stars – could be equivalently and directly presented in terms of arcs affected by the perturbation. Indeed, when the AvgTR values could be used as indicators of the performance trends, it means that the performance of both Reopt and Dijkstra is independent on the number of arcs involved in the cost perturbation.

RANDOM perturbation

In order to appraise the performance trends when the arc cost change is generic, that is both cost increase and decrease may occur, we considered also the following perturbation:

RANDOM: for each arc $(i, j) \in K$, a coin toss decided whether to apply the INCREASE or the DECREASE perturbation to its cost.

Table 2.16 Comparison of results for INCREASE, DECREASE and RANDOM perturbations on the "medium" instances with $Cmax = 280$. Results presented in terms of number of wins for Reopt, and AvgTR.

Typology	INCREASE		DECREASE		RANDOM	
	#Wins	AvgTR	#Wins	AvgTR	#Wins	AvgTR
grid	10/10	0.76 ± 6.7%	0/10	4.13 ± 59.9%	0/10	3.08 ± 36.0%
full-grid	3/10	1.11 ± 13.6%	0/10	3.14 ± 37.5%	0/10	3.86 ± 46.4%
cylinder	0/10	1.32 ± 6.2%	0/10	3.63 ± 42.8%	0/10	3.70 ± 48.9%
torus	10/10	0.48 ± 10.3%	5/10	1.12 ± 39.7%	5/10	1.31 ± 44.7%
square	0/10	1.97 ± 4.2%	0/10	1.48 ± 5.1%	0/10	1.48 ± 13.5%
square+25	10/10	0.83 ± 5.0%	10/10	0.68 ± 5.3%	10/10	0.71 ± 5.1%
square+50	0/10	2.02 ± 5.9%	0/10	1.70 ± 6.1%	0/10	1.72 ± 13.4%
rect	0/10	1.82 ± 3.5%	0/10	1.42 ± 5.0%	0/10	1.44 ± 9.3%
rect+25	9/10	0.82 ± 13.1%	10/10	0.65 ± 4.8%	10/10	0.67 ± 9.5%
rect+50	0/10	2.54 ± 6.6%	0/10	2.03 ± 7.8%	0/10	2.01 ± 3.8%
spacyc-4n	0/10	1.27 ± 12.7%	0/10	1.36 ± 14.1%	0/10	1.37 ± 8.2%
spacyc-6n	10/10	0.84 ± 11.6%	10/10	0.86 ± 9.1%	10/10	0.83 ± 9.9%
spacyc-8n	0/10	1.80 ± 10.9%	0/10	1.77 ± 7.8%	0/10	1.75 ± 10.7%
sprand-4n	10/10	0.37 ± 5.4%	10/10	0.27 ± 6.5%	10/10	0.29 ± 23.3%
sprand-6n	10/10	0.46 ± 7.4%	10/10	0.44 ± 5.8%	10/10	0.45 ± 16.4%
sprand-8n	10/10	0.67 ± 9.1%	10/10	0.57 ± 9.2%	10/10	0.56 ± 9.5%

As done in the previous sections, the perturbation was applied to the arcs in the forward stars of k randomly chosen nodes, where $k = 1, 2, \ldots 10$. We selected the medium sized instances of each family for which $Cmax = 280$; this choice is motivated by two aspects that emerged in the previous experimentation: i) whenever $Cmax \geq 280$ the resolution from scratch, via Dijkstra, is always the best performing approach; ii) the performance is not particularly affected by the size of the networks, thus the medium sized graphs could be used as sample graphs. In Table 2.16 the results for RANDOM perturbation are compared with those of INCREASE and DECREASE perturbations (presented in previous subsections) so as to identify some analogies.

Summary of Results

We expected to find similarities – in terms of number of wins and performance trends – between the results relative to DECREASE and RANDOM perturbation; in fact, in both cases Reopt has to execute at least one primal phase since some arcs could have a modified negative reduce cost. Indeed, focusing on the last four columns of Table 2.16, it is easy to find that kind of analogies for the whole set of considered instances.

In particular, for GTC instances we made a comparison of the running times of the dual phase and the primal ones: in 75% of cases the primal phases are those which require the most time effort thus confirming that the Depth First

Search, necessary to build the Dijkstrable subgraph, is determinant in terms of complexity.

On GRIDGEN instances, independently on the shape of the grid, the performance trends in the case of DECREASE and RANDOM changes are similar to that of the INCREASE perturbation. Hence, we conducted an analysis of the running times of the dual and the primal phases: for all the considered problems, the most time consuming operation of Reopt is the dual phase whose running time is γ times the total running time of the primal phases, where γ is at least 125 and at most 1427. As a result, the behaviour of the two algorithms on these networks is perfectly described by that observed in the case of INCREASE perturbations.

We drawn similar conclusions for SPACYC and SPRAND networks under consideration since, also on these graphs, the analysis of the running times revealed that the dual phase of Reopt is the most time consuming operation. In particular, whatever perturbation interests the SPRAND graphs, the reoptimization approach is always preferable for the medium sized instances with $Cmax = 280$.

2.4.5 Performance Profiles

This Section provides a further analysis conducted considering the *Performance Profiles* [58] for Reopt and Dijkstra, which give a measure of how much one of the two algorithms is worse than the other on a set of test problems.

In general, given a set $\mathcal{P}$ of n_p problems and a set $\mathcal{S}$ of n_s solvers, the *performance ratio* $r_{p,s}$ for each $p \in \mathcal{P}$ and each $s \in \mathcal{S}$ is defined as the ratio between the computation time $t_{p,s}$ of solver s on problem p and the best time of any solver on p [see 58]:

$$r_{p,s} = \frac{t_{p,s}}{\min\{t_{p,s} : s \in \mathcal{S}\}}.$$

Furthermore, the Performance Profile is the *cumulative distribution function* $\rho_s(\cdot)$ for the performance ratio of a solver $s \in \mathcal{S}$; indeed, for each $\tau \in \mathbb{R}$, $\rho_s(\tau)$ measures the probability that for s a performance ratio is within a factor τ of the best possible ratio:

$$\rho_s(\tau) = \frac{\left|\{p \in \mathcal{P} : r_{p,s} \leq \tau\}\right|}{n_p}.$$

Our analysis was conducted on all the families of instances, except the SPRAND one, since, as observed in Section 2.4.4, Reopt overcomes Dijkstra on the whole set of test problems.

We selected a set of problems for each kind of perturbation, and to make them suitably representative, we used the two parameters characterizing each typology of instances: size and maximum cost. Specifically, we fixed, in turn, one of them to the average value among those possible and left the other vary. In this way we selected all the graphs with $Cmax = 280$ and all the medium ones. Finally, we applied the INCREASE and DECREASE perturbations, separately, to the arcs in the forward stars of $k = 1, 2, \ldots 10$ nodes of each selected graph.

Figures 2.4a to 2.6c depict the representation of the Performance Profiles obtained with The MathWorks, Inc.MATLAB ® R2018b; in particular, we refer to them as "PerfPro" .

INCREASE perturbation

The PerfPro in Fig. 2.4a refers to GTC instances: it shows that approximately half of the considered problems are solved faster by `Reopt`, being its Performance Profile value approximately equal to 0.5 when $\tau = 1$. Moreover, with probability 1, when `Dijkstra` is faster than `Reopt`, then the `time-ratio` (Equation (2.5)) is at most equal to 3.

As regards SQUARE and RECT networks (Fig. 2.4b and Fig. 2.4c), the PerfPros confirm that the performance of the two algorithms are comparable. In fact, `Reopt` is faster than `Dijkstra` with probability 0.5 and the τ values are approximately the same. For example, on RECT instances, when an algorithm exhibits the best running time, then τ for the other one is at most equal to 4.

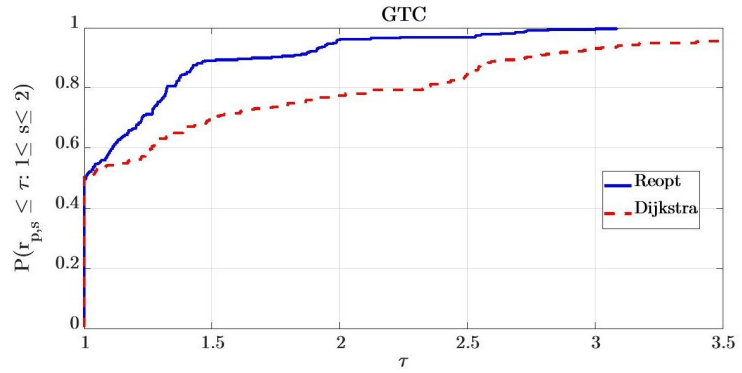
The PerfPro in Fig. 2.5a confirms that on SPACYC networks, although `Reopt` is not competitive with `Dijkstra` on many instances, the values of τ remain relatively low: the reoptimization approach is not much worse than a resolution from scratch.

DECREASE perturbation

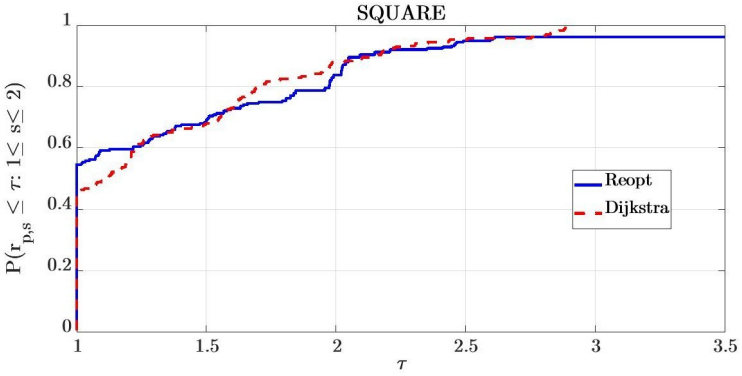
Fig. 2.5b shows that when `Reopt` performs also the primal phases, then τ grows dramatically for GTC problems: we truncated the graphic at $\tau = 10$, but the maximum value observed was $\tau = 14$, meaning that the running time of `Reopt` is up to 14 times worse than the one of `Dijkstra`.

The PerfPros in Fig. 2.6a and Fig. 2.6b give another proof of the similarity between the performance trends on SQUARE and RECT instances. Actually, just few less problems of RECT family, with respect to the SQUARE one, are solved faster by `Reopt`. Moreover, `Dijkstra` presents for both the sets at most $\tau \approx 2.5$ meaning that when `Reopt` is faster, then the minimum `time-ratio` (Equation (2.5)) is approximately $1/2.5$.

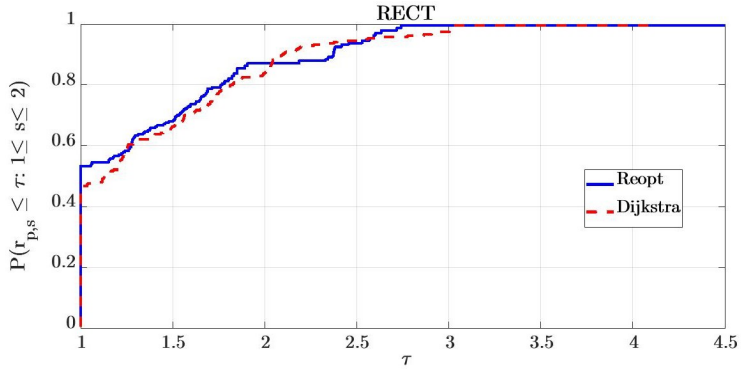
As regards the acyclic networks, the PerfPro in Fig. 2.6c underlines the resolution from scratch is the best performing approach with probability ≈ 0.78. In particular, we truncated the graphic in Fig. 2.6c at $\tau = 4.5$, but the maximum value observed is $\tau = 22$; namely there is at least an SPACYC instance on which `Reopt` running time is $1/22 \approx 0.045$ the one of `Dijkstra`.



(a) `GTC-INCREASE`.

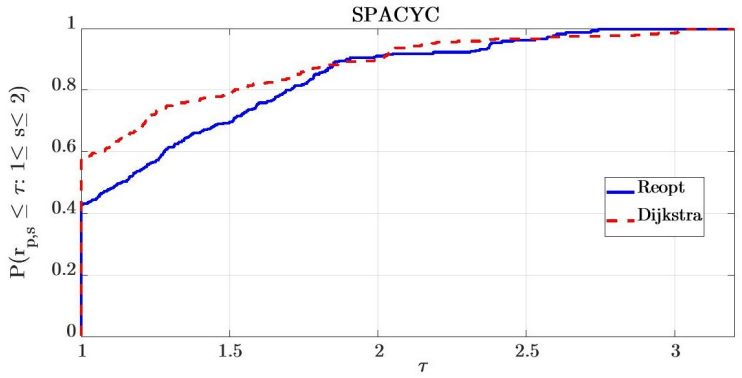


(b) `SQUARE-INCREASE`.

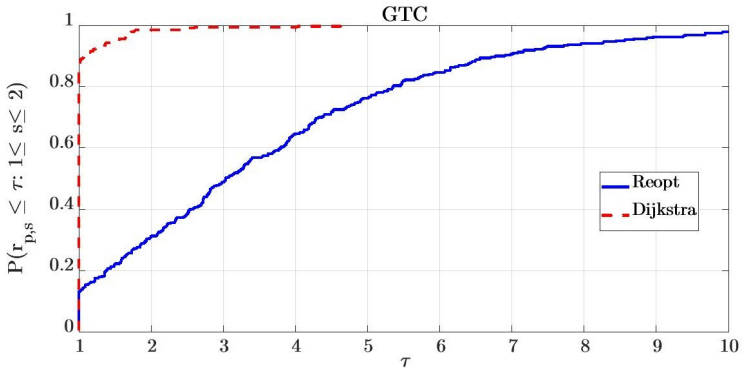


(c) `RECT-INCREASE`.

Fig. 2.4 PerfPro of `Reopt` and `Dijkstra` on `GTC`, `SQUARE`, and `RECT` "medium" instances and those with *Cmax* = 280.

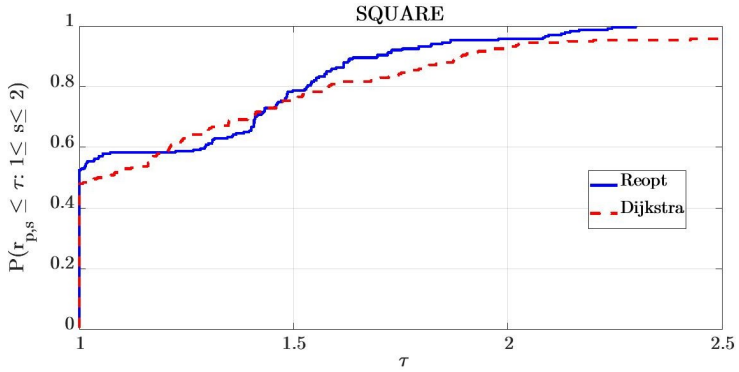


(a) SPACYC-INCREASE.

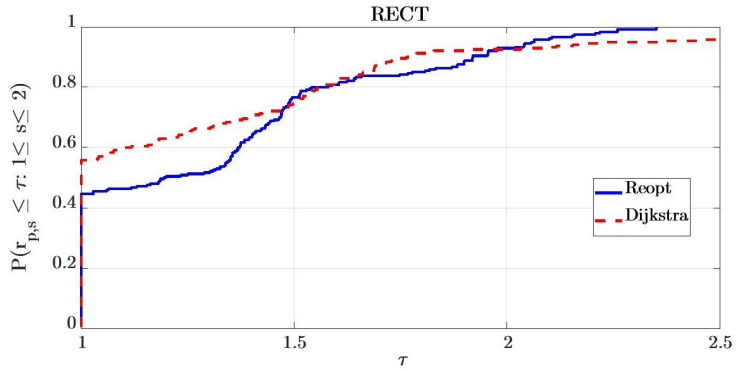


(b) GTC-DECREASE.

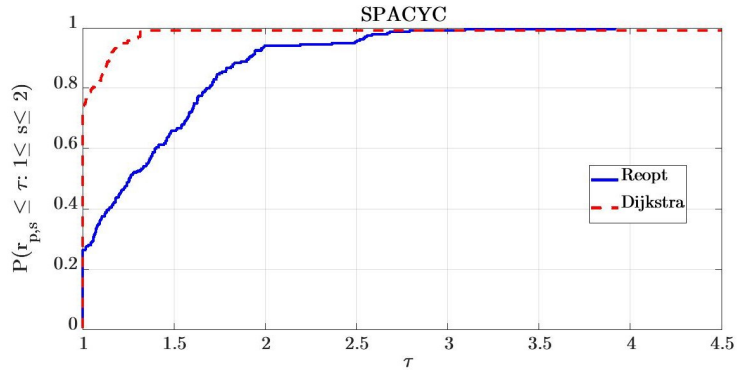
Fig. 2.5 PerfPro of `Reopt` and `Dijkstra` on SPACYC and GTC "medium" instances and those with $Cmax = 280$.



(a) SQUARE-DECREASE.



(b) RECT-DECREASE.



(c) SPACYC-DECREASE.

Fig. 2.6 PerfPro of `Reopt` and `Dijkstra` on SQUARE, RECT, and SPACYC "medium" instances and those with *Cmax* = 280.

2.5 Conclusions

This Chapter describes the framework of the reoptimization of shortest paths problem along with a methodological primal-dual approach devised ad hoc in Pallottino and Scutellá [144] to deal with the most general scenario, namely the case when any subset of arcs of the input graph is given a new cost, which can be either lower or higher than the old one.

Indeed, in Festa et al. [83] we exploited the features of the Multi Level Bucket data structure devised by Cherkassky et al. [40] to implement this approach, since to the best of our knowledge any implementation had been proposed so far. Moreover, we conducted several experiments on a heterogeneous set of instances, with the aim of detecting when the reoptimization is preferable to the classic resolution from scratch, performed with the well known Dijkstra's algorithm.

The considered instances mainly differ for their underlying graph structure, according to which they can be classified as: *a. graphs with specific topologies* (torus, cylinder and grid); *b. grids with additional random connections*; *c. fully random graphs* (acyclic and not). Furthermore, to mimic the occurrences of arc costs increase and decrease, we defined two kinds of perturbation for the cost function, namely `INCREASE` and `DECREASE`, which we performed both separately and jointly. On the test problems *a.*, in case of arc cost decrease, we determined that the resolution from scratch is always preferable. In fact, due to the intrinsic structure of these networks, the location of the arcs whose cost is decreased, strongly affects the performance of the reoptimization algorithm. On the contrary, we detected that the absence of an intrinsic structure in the network is beneficial for the reoptimization. In fact, it is to be preferred when the perturbed graph is fully random, independently from the kind of perturbation.

In conclusion, we were able to trace out a border – in terms of cost, topology and size – separating the set of instances for which the reoptimization approach is more suitable than a standard resolution.

With these results in mind, our future research is twofold. On the one hand, we would like to compare the considered algorithm with other reoptimization approaches. On the other hand, we would investigate the applicability of the reoptimization paradigm to problems arising in different fields.

Appendix

Appendix 2.A Scatter Plots

This Section presents the scatter plots related to the results of the testing phase. All the graphics are obtained with The MathWorks, Inc.MATLAB ® R2018b; moreover, the AvgTR values are given on the y-axis while the different *Cmax* values are reported on the x-axis.

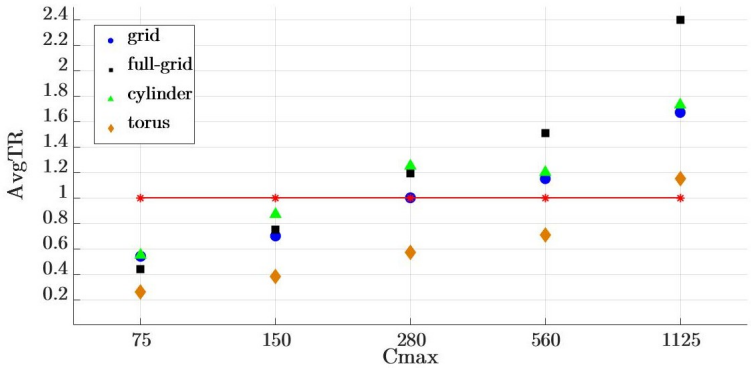
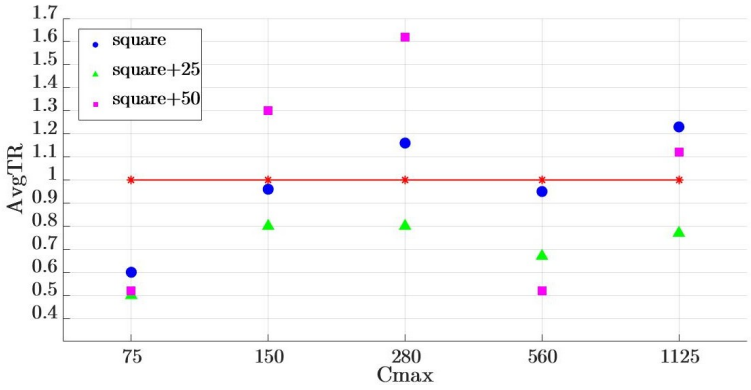
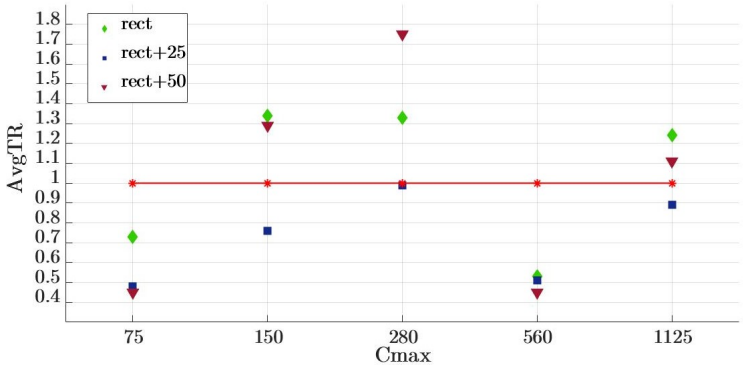


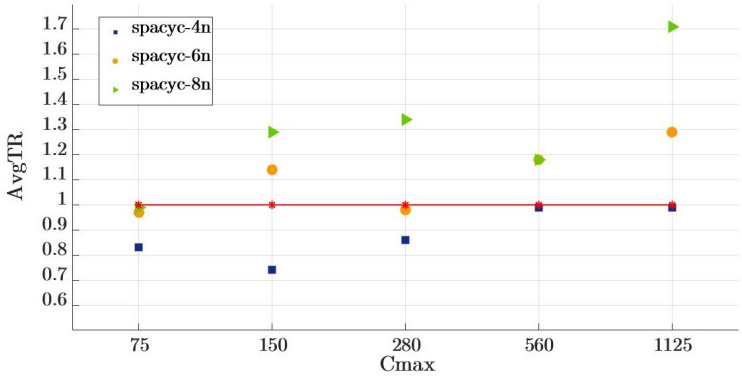
Fig. 2.7 (`GTC-INCREASE`). Average `time-ratio` for each typology of networks. Solid red line represents `Dijkstra`'s AvgTR, equal to 1.



(a) SQUARE-INCREASE.

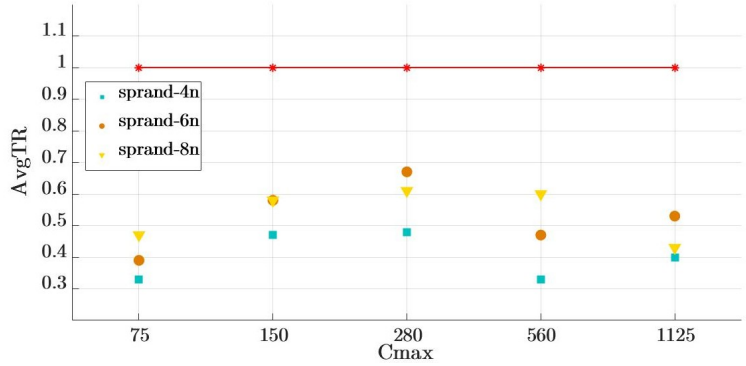


(b) RECT-INCREASE.

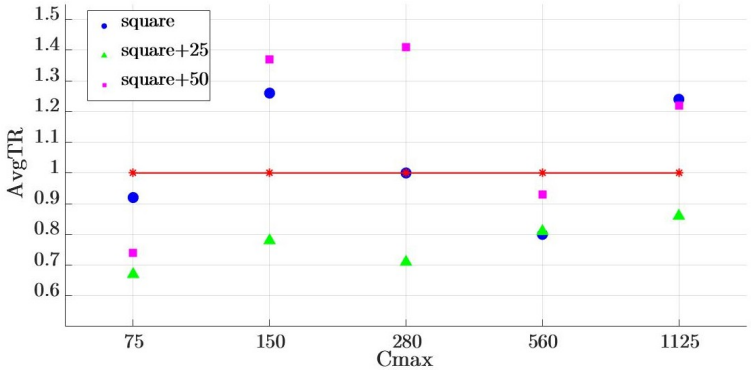


(c) SPACYC-INCREASE.

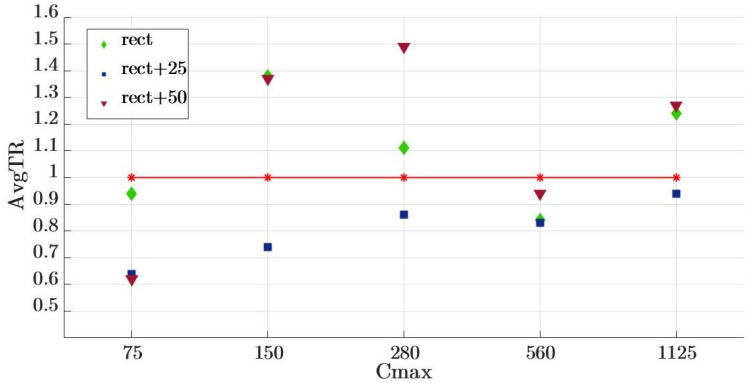
Fig. 2.8 (SQUARE, RECT, SPACYC). Average `time-ratio` for each typology of networks. Solid red line represents `Dijkstra`'s AvgTR, equal to 1.



(a) SPRAND-INCREASE.



(b) SQUARE-DECREASE.



(c) RECT-DECREASE.

Fig. 2.9 (SPRAND, SQUARE, RECT). Average `time-ratio` for each typology of networks. Solid red line represents `Dijkstra`'s AvgTR, equal to 1.

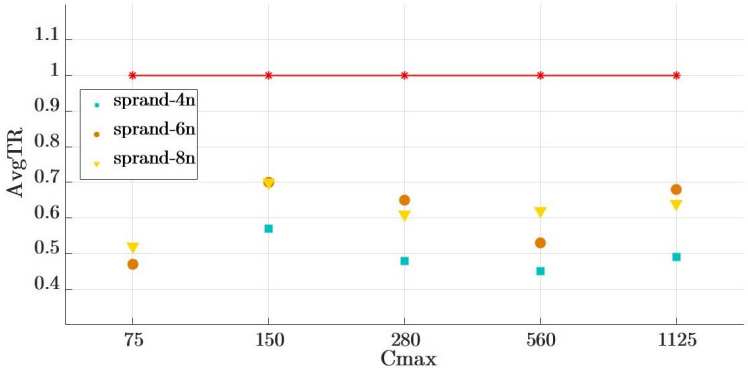


Fig. 2.10 (SPRAND-DECREASE). Average `time-ratio` for each typology of networks. Solid red line represents `Dijkstra`'s AvgTR, equal to 1.

Appendix 2.B Time Results

This Section summarizes the computation times registered during the experimentation[4]. Specifically, we computed their average over the number of forward stars affected by the cost perturbation.

Tables 2.17 to 2.21 refer to the `INCREASE` perturbation and contain the average computation times of `Reopt` and `Dijkstra`. Tables 2.22 to 2.27, instead, refer to the `DECREASE` perturbation: for each instance, the average computation times of both the dual and the primal phases along with the total (average) running time of `Reopt` is given. All the times are expressed in seconds while the lowest computation times are reported in bold.

[4]Data of the executed experiments are available at `https://figshare.com/articles/dataset/RESULTS/10129223`.

Table 2.17 (GTC-INCREASE). Computational results in terms of average times.

Cmax	Instance	Reopt	Dijkstra	Cmax	Instance	Reopt	Dijkstra
75	grid100	**0.005 ± 15.7%**	0.006 ± 16.6%	280	cylinder100	0.006 ± 25.1%	**0.006 ± 28.6%**
75	grid400	0.216 ± 8.9%	**0.014 ± 7.8%**	280	cylinder400	0.314 ± 5.5%	**0.238 ± 2.1%**
75	grid700	1.296 ± 7.7%	**0.006 ± 10.5%**	280	cylinder700	1.880 ± 8.2%	**1.456 ± 3.3%**
75	full-grid100	**0.008 ± 8.2%**	0.010 ± 7.8%	280	torus100	**0.006 ± 19.4%**	0.007 ± 11.5%
75	full-grid400	0.610 ± 6.4%	**0.546 ± 7.5%**	280	torus400	**0.250 ± 11.5%**	0.528 ± 4.9%
75	full-grid700	4.556 ± 13.1%	**1.785 ± 9.4%**	280	torus700	**1.434 ± 7.7%**	3.576 ± 4.5%
75	cylinder100	**0.005 ± 8.4%**	0.618 ± 4.6%	560	grid100	0.005 ± 13.6%	**0.005 ± 20.5%**
75	cylinder400	**0.236 ± 6.9%**	1.413 ± 8.0%	560	grid400	0.154 ± 6.4%	**0.144 ± 1.6%**
75	cylinder700	**1.520 ± 8.9%**	3.505 ± 6.9%	560	grid700	1.090 ± 11.7%	**0.799 ± 4.0%**
75	torus100	**0.005 ± 14.5%**	12.379 ± 7.8%	560	full-grid100	0.009 ± 5.5%	**0.008 ± 8.8%**
75	torus400	**0.228 ± 4.1%**	3.860 ± 8.2%	560	full-grid400	0.605 ± 7.1%	**0.437 ± 5.6%**
75	torus700	**1.188 ± 9.3%**	10.157 ± 7.6%	560	full-grid700	5.422 ± 4.3%	**2.690 ± 4.5%**
150	grid100	0.005 ± 8.5%	**0.005 ± 5.5%**	560	cylinder100	0.006 ± 19.5%	**0.005 ± 18.2%**
150	grid400	**0.297 ± 50.6%**	0.415 ± 4.3%	560	cylinder400	0.213 ± 7.5%	**0.166 ± 6.7%**
150	grid700	**1.239 ± 12.9%**	2.695 ± 4.0%	560	cylinder700	1.070 ± 4.2%	**0.869 ± 2.2%**
150	full-grid100	**0.009 ± 7.8%**	0.014 ± 15.2%	560	torus100	**0.005 ± 15.3%**	0.006 ± 14.0%
150	full-grid400	**1.082 ± 11.4%**	1.387 ± 5.4%	560	torus400	**0.203 ± 7.9%**	0.310 ± 5.1%
150	full-grid700	**7.334 ± 7.3%**	9.661 ± 12.6%	560	torus700	**1.076 ± 6.2%**	2.000 ± 3.6%
150	cylinder100	0.006 ± 16.4%	**0.005 ± 7.4%**	1125	grid100	0.005 ± 2.6%	**0.004 ± 14.0%**
150	cylinder400	**0.409 ± 12.0%**	0.473 ± 1.9%	1125	grid400	0.241 ± 16.1%	**0.126 ± 18.8%**
150	cylinder700	**1.943 ± 9.5%**	3.147 ± 13.3%	1125	grid700	1.076 ± 17.1%	**0.591 ± 13.4%**
150	torus100	**0.005 ± 8.2%**	0.009 ± 8.1%	1125	full-grid100	0.009 ± 4.1%	**0.007 ± 5.4%**
150	torus400	**0.312 ± 11.2%**	1.071 ± 7.4%	1125	full-grid400	0.853 ± 14.6%	**0.312 ± 12.1%**
150	torus700	**1.804 ± 9.1%**	7.532 ± 8.7%	1125	full-grid700	5.682 ± 13.0%	**1.823 ± 15.0%**
280	grid100	0.006 ± 28.9%	**0.006 ± 25.1%**	1125	cylinder100	0.006 ± 14.7%	**0.004 ± 18.0%**
280	grid400	**0.170 ± 7.0%**	0.223 ± 4.5%	1125	cylinder400	0.258 ± 17.6%	**0.131 ± 13.7%**
280	grid700	1.505 ± 5.1%	**1.328 ± 3.8%**	1125	cylinder700	1.225 ± 15.8%	**0.635 ± 12.6%**
280	full-grid100	0.012 ± 16.9%	**0.010 ± 22.6%**	1125	torus100	0.006 ± 23.6%	**0.005 ± 3.8%**
280	full-grid400	0.849 ± 12.9%	**0.775 ± 8.4%**	1125	torus400	0.276 ± 32.4%	**0.238 ± 26.6%**
280	full-grid700	6.267 ± 5.5%	**4.714 ± 5.7%**	1125	torus700	1.421 ± 14.7%	**1.328 ± 12.8%**

Table 2.18 (SQUARE-INCREASE). Computational results in terms of average times.

Cmax	Instance	Reopt	Dijkstra	Cmax	Instance	Reopt	Dijkstra
75	square100	**0.098 ± 30.5%**	0.164 ± 3.6%	280	square700+25	**1785.016 ± 30.7%**	2866.994 ± 5.3%
75	square400	**69.440 ± 30.6%**	90.174 ± 3.0%	280	square100+50	**0.218 ± 1.7%**	0.621 ± 0.4%
75	square700	**1112.279 ± 30.8%**	2581.711 ± 4.1%	280	square400+50	170.514 ± 2.4%	**84.793 ± 5.9%**
75	square100+25	**0.105 ± 30.7%**	0.289 ± 1.3%	280	square700+50	2321.703 ± 3.4%	**932.311 ± 1.6%**
75	square400+25	**87.247 ± 30.4%**	178.069 ± 5.8%	560	square100	**0.088 ± 30.5%**	0.244 ± 2.6%
75	square700+25	**1346.358 ± 30.6%**	2131.773 ± 3.8%	560	square400	**67.162 ± 30.2%**	109.673 ± 1.5%
75	square100+50	**0.147 ± 4.9%**	0.312 ± 1.4%	560	square700	879.010 ± 30.5%	**466.911 ± 2.1%**
75	square400+50	**112.996 ± 6.0%**	236.370 ± 12.4%	560	square100+25	**0.122 ± 31.0%**	0.156 ± 13.3%
75	square700+50	**1633.171 ± 5.4%**	2758.064 ± 9.6%	560	square400+25	**97.726 ± 30.6%**	152.029 ± 8.7%
150	square100	**0.120 ± 30.3%**	0.218 ± 2.5%	560	square700+25	**1282.328 ± 30.6%**	2733.063 ± 5.1%
150	square400	88.394 ± 30.5%	**55.436 ± 2.8%**	560	square100+50	**0.146 ± 3.8%**	0.568 ± 0.4%
150	square700	**1317.089 ± 30.7%**	1896.875 ± 4.1%	560	square400+50	**118.486 ± 2.1%**	140.853 ± 5.4%
150	square100+25	0.173 ± 30.2%	**0.166 ± 1.6%**	560	square700+50	**1587.609 ± 4.0%**	3384.309 ± 6.3%
150	square400+25	**125.723 ± 30.4%**	219.410 ± 3.8%	1125	square100	**0.144 ± 31.7%**	0.278 ± 4.0%
150	square700+25	**2241.168 ± 31.3%**	2912.161 ± 4.8%	1125	square400	109.435 ± 31.0%	**71.903 ± 8.5%**
150	square100+50	**0.272 ± 11.6%**	0.295 ± 8.6%	1125	square700	1563.078 ± 33.0%	**955.302 ± 10.2%**
150	square400+50	160.158 ± 4.2%	**121.238 ± 3.0%**	1125	square100+25	0.215 ± 31.0%	**0.098 ± 52.6%**
150	square700+50	2858.243 ± 4.5%	**1716.047 ± 6.0%**	1125	square400+25	222.478 ± 33.1%	**62.256 ± 95.9%**
280	square100	**0.084 ± 30.5%**	0.189 ± 1.7%	1125	square700+25	**2091.403 ± 30.3%**	3120.434 ± 3.6%
280	square400	77.192 ± 30.3%	**39.225 ± 2.0%**	1125	square100+50	**0.240 ± 4.7%**	0.324 ± 1.2%
280	square700	1009.582 ± 30.4%	**957.170 ± 2.0%**	1125	square400+50	183.144 ± 7.0%	**146.308 ± 10.4%**
280	square100+25	**0.129 ± 30.3%**	0.136 ± 2.3%	1125	square700+50	2476.785 ± 1.7%	**1803.502 ± 2.0%**
280	square400+25	**121.813 ± 30.5%**	147.319 ± 7.7%				

Table 2.19 (RECT-INCREASE). Computational results in terms of average times.

Cmax	Instance	Reopt	Dijkstra	Cmax	Instance	Reopt	Dijkstra
75	rect100	**0.428 ± 6.4%**	0.672 ± 5.4%	280	rect500+25	**1855.466 ± 9.2%**	2141.463 ± 4.7%
75	rect300	**103.575 ± 8.4%**	142.426 ± 16.8%	280	rect100+50	**0.802 ± 3.2%**	2.460 ± 13.1%
75	rect500	**1274.926 ± 8.1%**	1542.604 ± 7.8%	280	rect300+50	253.431 ± 6.5%	**100.312 ± 9.4%**
75	rect100+25	**0.490 ± 8.4%**	1.231 ± 7.0%	280	rect500+50	2328.798 ± 1.1%	**981.971 ± 1.3%**
75	rect300+25	**118.835 ± 7.5%**	248.390 ± 10.2%	560	rect100	**0.343 ± 1.9%**	0.773 ± 1.8%
75	rect500+25	**1533.759 ± 7.9%**	2764.646 ± 7.4%	560	rect300	**88.357 ± 4.9%**	156.109 ± 8.3%
75	rect100+50	**0.559 ± 7.7%**	1.416 ± 2.4%	560	rect500	**908.457 ± 1.5%**	1595.823 ± 1.9%
75	rect300+50	**148.807 ± 10.1%**	339.988 ± 11.6%	560	rect100+25	**0.435 ± 0.3%**	1.358 ± 0.2%
75	rect500+50	**1638.135 ± 1.3%**	3229.069 ± 2.5%	560	rect300+25	**116.691 ± 3.9%**	192.197 ± 10.2%
150	rect100	**0.504 ± 2.6%**	0.663 ± 1.7%	560	rect500+25	**1299.462 ± 1.4%**	2131.430 ± 1.9%
150	rect300	118.765 ± 2.5%	**77.563 ± 4.0%**	560	rect100+50	**0.549 ± 3.1%**	2.219 ± 3.6%
150	rect500	1466.894 ± 2.7%	**855.051 ± 2.3%**	560	rect300+50	**165.011 ± 10.3%**	285.276 ± 15.0%
150	rect100+25	**0.747 ± 2.5%**	0.754 ± 2.1%	560	rect500+50	**1588.348 ± 1.5%**	3039.907 ± 3.3%
150	rect300+25	**172.080 ± 7.2%**	295.575 ± 6.4%	1125	rect100	**0.696 ± 1.3%**	1.063 ± 29.1%
150	rect500+25	**2292.514 ± 2.1%**	3309.671 ± 2.8%	1125	rect300	142.945 ± 5.0%	**85.400 ± 24.0%**
150	rect100+50	**0.995 ± 6.9%**	1.066 ± 3.2%	1125	rect500	1527.912 ± 2.9%	**970.203 ± 2.6%**
150	rect300+50	235.779 ± 13.5%	**175.715 ± 9.7%**	1125	rect100+25	**0.978 ± 4.7%**	0.995 ± 3.7%
150	rect500+50	2763.136 ± 0.3%	**1722.489 ± 2.8%**	1125	rect300+25	**204.535 ± 10.2%**	254.682 ± 9.2%
280	rect100	**0.390 ± 0.5%**	0.797 ± 0.7%	1125	rect500+25	**2115.227 ± 2.9%**	2437.732 ± 2.0%
280	rect300	99.733 ± 3.9%	**54.864 ± 3.8%**	1125	rect100+50	**1.164 ± 3.6%**	1.411 ± 1.0%
280	rect500	1100.947 ± 2.2%	**658.990 ± 2.4%**	1125	rect300+50	240.534 ± 7.3%	**198.017 ± 6.7%**
280	rect100+25	0.628 ± 5.4%	**0.495 ± 4.6%**	1125	rect500+50	2460.484 ± 2.0%	**1927.355 ± 3.7%**
280	rect300+25	**187.601 ± 11.4%**	230.788 ± 16.0%				

Table 2.20 (SPACYC-INCREASE). Computational results in terms of average times.

Cmax	Instance	Reopt	Dijkstra	Cmax	Instance	Reopt	Dijkstra
75	spacyc-4n100	**0.019 ± 8.5%**	0.025 ± 5.5%	280	spacyc-6n700	174.987 ± 1.7%	**173.022 ± 2.8%**
75	spacyc-4n400	9.909 ± 21.3%	**6.812 ± 16.7%**	280	spacyc-8n100	0.061 ± 5.6%	**0.060 ± 4.1%**
75	spacyc-4n700	**47.775 ± 4.4%**	192.180 ± 3.6%	280	spacyc-8n400	23.565 ± 9.2%	**13.155 ± 6.7%**
75	spacyc-6n100	0.049 ± 10.9%	**0.040 ± 7.8%**	280	spacyc-8n700	401.547 ± 6.9%	**337.893 ± 11.0%**
75	spacyc-6n400	11.648 ± 7.1%	**10.631 ± 6.7%**	560	spacyc-4n100	**0.020 ± 10.5%**	0.027 ± 12.1%
75	spacyc-6n700	**232.806 ± 2.7%**	397.822 ± 4.9%	560	spacyc-4n400	8.074 ± 35.3%	**6.693 ± 35.4%**
75	spacyc-8n100	**0.048 ± 5.4%**	0.059 ± 10.9%	560	spacyc-4n700	51.698 ± 14.6%	**51.009 ± 14.9%**
75	spacyc-8n400	26.066 ± 6.8%	**19.882 ± 10.8%**	560	spacyc-6n100	0.048 ± 9.4%	**0.044 ± 7.2%**
75	spacyc-8n700	**500.684 ± 4.0%**	605.070 ± 3.2%	560	spacyc-6n400	**10.119 ± 9.5%**	12.020 ± 6.7%
150	spacyc-4n100	**0.019 ± 6.4%**	0.026 ± 6.8%	560	spacyc-6n700	343.136 ± 7.3%	**215.279 ± 8.0%**
150	spacyc-4n400	6.965 ± 13.7%	**5.931 ± 10.2%**	560	spacyc-8n100	**0.053 ± 3.4%**	0.055 ± 3.3%
150	spacyc-4n700	**42.022 ± 7.7%**	139.898 ± 6.0%	560	spacyc-8n400	**24.068 ± 5.4%**	25.238 ± 5.9%
150	spacyc-6n100	0.047 ± 4.8%	**0.032 ± 6.4%**	560	spacyc-8n700	608.625 ± 3.8%	**375.471 ± 5.1%**
150	spacyc-6n400	12.041 ± 10.5%	**10.243 ± 5.1%**	1125	spacyc-4n100	**0.019 ± 6.2%**	0.028 ± 8.6%
150	spacyc-6n700	**211.142 ± 3.6%**	279.552 ± 3.8%	1125	spacyc-4n400	8.869 ± 13.3%	**5.790 ± 7.5%**
150	spacyc-8n100	0.065 ± 8.7%	**0.054 ± 11.7%**	1125	spacyc-4n700	**72.510 ± 18.1%**	101.774 ± 11.7%
150	spacyc-8n400	32.111 ± 6.5%	**18.965 ± 7.4%**	1125	spacyc-6n100	0.054 ± 8.2%	**0.038 ± 5.1%**
150	spacyc-8n700	**510.751 ± 5.9%**	528.506 ± 5.2%	1125	spacyc-6n400	**11.184 ± 7.5%**	11.984 ± 10.9%
280	spacyc-4n100	**0.021 ± 3.8%**	0.024 ± 7.5%	1125	spacyc-6n700	363.314 ± 2.3%	**240.830 ± 4.9%**
280	spacyc-4n400	8.597 ± 16.5%	**6.844 ± 17.4%**	1125	spacyc-8n100	0.073 ± 14.1%	**0.064 ± 10.5%**
280	spacyc-4n700	**40.044 ± 4.9%**	92.706 ± 13.8%	1125	spacyc-8n400	32.790 ± 5.4%	**18.261 ± 8.3%**
280	spacyc-6n100	**0.044 ± 9.3%**	0.045 ± 12.4%	1125	spacyc-8n700	791.829 ± 2.5%	**365.369 ± 4.7%**
280	spacyc-6n400	**9.550 ± 9.4%**	11.370 ± 7.0%				

Table 2.21 (SPRAND-INCREASE). Computational results in terms of average times.

Cmax	Instance	Reopt	Dijkstra	Cmax	Instance	Reopt	Dijkstra
75	sprand-4n100	**0.040 ± 8.2%**	0.149 ± 3.3%	280	sprand-6n700	**1495.921 ± 7.1%**	1589.039 ± 6.1%
75	sprand-4n400	**28.431 ± 5.6%**	60.898 ± 3.5%	280	sprand-8n100	**0.157 ± 2.6%**	0.179 ± 3.8%
75	sprand-4n700	**478.396 ± 6.4%**	1850.752 ± 8.7%	280	sprand-8n400	**146.535 ± 7.8%**	229.755 ± 5.3%
75	sprand-6n100	**0.072 ± 4.7%**	0.274 ± 3.0%	280	sprand-8n700	**2269.372 ± 4.1%**	7481.925 ± 5.1%
75	sprand-6n400	**56.685 ± 3.6%**	81.331 ± 5.2%	560	sprand-4n100	**0.040 ± 8.3%**	0.095 ± 4.8%
75	sprand-6n700	**873.333 ± 6.1%**	3926.379 ± 5.4%	560	sprand-4n400	**30.549 ± 6.3%**	124.358 ± 4.8%
75	sprand-8n100	**0.122 ± 2.8%**	0.165 ± 4.2%	560	sprand-4n700	**511.389 ± 4.1%**	1576.355 ± 5.9%
75	sprand-8n400	**92.729 ± 3.8%**	327.888 ± 7.5%	560	sprand-6n100	**0.083 ± 4.9%**	0.156 ± 4.5%
75	sprand-8n700	**1369.751 ± 7.0%**	3700.496 ± 4.4%	560	sprand-6n400	**68.581 ± 7.7%**	189.612 ± 5.6%
150	sprand-4n100	**0.061 ± 6.2%**	0.104 ± 3.1%	560	sprand-6n700	**1045.349 ± 2.8%**	2030.718 ± 5.0%
150	sprand-4n400	**42.249 ± 5.8%**	87.088 ± 8.8%	560	sprand-8n100	**0.121 ± 4.9%**	0.124 ± 2.5%
150	sprand-4n700	**718.315 ± 7.3%**	2243.618 ± 5.7%	560	sprand-8n400	**103.811 ± 4.4%**	173.816 ± 5.1%
150	sprand-6n100	**0.109 ± 5.4%**	0.178 ± 4.1%	560	sprand-8n700	**1610.508 ± 2.5%**	7233.787 ± 2.1%
150	sprand-6n400	**85.838 ± 7.9%**	159.791 ± 9.3%	1125	sprand-4n100	**0.072 ± 10.6%**	0.135 ± 5.3%
150	sprand-6n700	**1380.799 ± 8.4%**	2403.460 ± 6.4%	1125	sprand-4n400	**48.042 ± 13.1%**	162.330 ± 14.8%
150	sprand-8n100	**0.140 ± 3.5%**	0.291 ± 2.0%	1125	sprand-4n700	**705.023 ± 13.6%**	1888.749 ± 4.8%
150	sprand-8n400	**122.914 ± 8.7%**	191.146 ± 6.7%	1125	sprand-6n100	**0.120 ± 4.6%**	0.219 ± 2.8%
150	sprand-8n700	**1850.916 ± 5.8%**	2990.540 ± 2.5%	1125	sprand-6n400	**80.727 ± 6.3%**	150.366 ± 3.7%
280	sprand-4n100	**0.055 ± 7.0%**	0.072 ± 3.7%	1125	sprand-6n700	**1313.006 ± 14.3%**	2555.907 ± 10.9%
280	sprand-4n400	**38.622 ± 3.7%**	131.068 ± 3.1%	1125	sprand-8n100	**0.148 ± 4.1%**	0.341 ± 3.7%
280	sprand-4n700	**642.400 ± 6.1%**	1748.046 ± 7.1%	1125	sprand-8n400	**104.675 ± 2.7%**	221.252 ± 4.4%
280	sprand-6n100	**0.118 ± 3.7%**	0.193 ± 4.1%	1125	sprand-8n700	**1666.484 ± 5.1%**	4285.418 ± 4.1%
280	sprand-6n400	**85.451 ± 4.3%**	187.742 ± 4.6%				

Table 2.22 (GTC-DECREASE). Computational results in terms of average times on instances with $Cmax \in \{75, 150, 280\}$. Reopt and Dijkstra average computation times in last two columns.

Cmax	Instance	Reopt			Dijkstra
		Dual	Primal	Total	
75	grid100	0.008 ± 39.9%	0.019 ± 65.8%	0.026 ± 51.4%	**0.008 ± 38.1%**
75	grid400	0.224 ± 35.8%	3.084 ± 95.9%	3.309 ± 90.3%	**0.562 ± 34.0%**
75	grid700	1.334 ± 33.7%	57.778 ± 65.0%	59.112 ± 63.9%	**3.222 ± 33.2%**
75	full-grid100	0.011 ± 39.0%	0.030 ± 74.8%	0.041 ± 59.5%	**0.018 ± 35.0%**
75	full-grid400	0.614 ± 41.4%	5.878 ± 70.2%	6.492 ± 65.1%	**1.772 ± 33.5%**
75	full-grid700	4.527 ± 34.2%	325.382 ± 101.0%	329.909 ± 99.7%	**12.376 ± 33.5%**
75	cylinder100	0.007 ± 39.6%	0.018 ± 64.5%	0.025 ± 52.6%	**0.008 ± 43.5%**
75	cylinder400	0.239 ± 36.0%	1.284 ± 69.5%	1.523 ± 61.0%	**0.593 ± 34.1%**
75	cylinder700	1.545 ± 34.3%	66.691 ± 54.9%	68.236 ± 54.0%	**3.691 ± 33.9%**
75	torus100	0.008 ± 39.4%	0.017 ± 69.6%	0.025 ± 53.2%	**0.015 ± 36.2%**
75	torus400	0.234 ± 35.7%	0.651 ± 76.0%	**0.884 ± 61.1%**	1.399 ± 34.8%
75	torus700	1.111 ± 34.8%	4.575 ± 86.9%	**5.686 ± 72.6%**	10.191 ± 33.4%
150	grid100	0.005 ± 33.6%	0.017 ± 66.7%	0.022 ± 55.4%	**0.006 ± 36.9%**
150	grid400	0.202 ± 33.3%	1.545 ± 77.1%	1.747 ± 69.9%	**0.368 ± 33.2%**
150	grid700	1.013 ± 34.7%	8.643 ± 64.5%	9.656 ± 59.3%	**2.405 ± 34.5%**
150	full-grid100	0.010 ± 33.7%	0.022 ± 78.0%	0.033 ± 59.3%	**0.013 ± 33.3%**
150	full-grid400	0.888 ± 33.6%	3.216 ± 74.2%	4.104 ± 61.1%	**1.249 ± 33.2%**
150	full-grid700	6.098 ± 34.1%	35.743 ± 83.0%	41.842 ± 72.3%	**8.942 ± 34.8%**
150	cylinder100	0.007 ± 37.8%	0.017 ± 67.7%	0.023 ± 53.3%	**0.006 ± 33.2%**
150	cylinder400	0.328 ± 33.5%	0.889 ± 70.9%	1.217 ± 57.1%	**0.424 ± 33.2%**
150	cylinder700	1.505 ± 33.6%	5.001 ± 71.8%	6.506 ± 59.2%	**2.638 ± 33.3%**
150	torus100	0.006 ± 34.2%	0.015 ± 71.4%	0.021 ± 56.9%	**0.009 ± 33.3%**
150	torus400	0.255 ± 33.4%	0.485 ± 63.0%	**0.739 ± 48.6%**	0.937 ± 33.2%
150	torus700	1.377 ± 41.2%	2.415 ± 62.4%	**3.882 ± 46.7%**	6.636 ± 33.5%
280	grid100	0.007 ± 40.9%	0.024 ± 74.3%	0.031 ± 63.0%	**0.007 ± 39.7%**
280	grid400	0.231 ± 37.2%	1.004 ± 91.6%	1.234 ± 79.2%	**0.289 ± 35.7%**
280	grid700	1.813 ± 34.1%	7.210 ± 65.6%	9.022 ± 55.8%	**1.679 ± 33.9%**
280	full-grid100	0.014 ± 38.7%	0.028 ± 85.0%	0.042 ± 63.5%	**0.012 ± 39.7%**
280	full-grid400	1.055 ± 34.0%	1.877 ± 68.7%	2.932 ± 51.7%	**0.930 ± 33.8%**
280	full-grid700	7.851 ± 33.5%	13.502 ± 73.7%	21.353 ± 53.4%	**5.949 ± 33.7%**
280	cylinder100	0.008 ± 42.5%	0.023 ± 74.7%	0.031 ± 62.3%	**0.007 ± 44.9%**
280	cylinder400	0.399 ± 34.2%	0.751 ± 73.1%	1.151 ± 54.6%	**0.320 ± 34.4%**
280	cylinder700	2.477 ± 33.5%	5.910 ± 80.7%	8.387 ± 62.3%	**1.952 ± 33.5%**
280	torus100	0.008 ± 38.2%	0.019 ± 71.2%	0.027 ± 54.4%	**0.011 ± 37.6%**
280	torus400	0.310 ± 35.6%	0.441 ± 75.0%	0.752 ± 52.7%	**0.675 ± 34.0%**
280	torus700	1.927 ± 33.6%	2.893 ± 81.4%	4.820 ± 55.8%	**4.705 ± 34.0%**

Table 2.23 (GTC-DECREASE). Computational results in terms of average times on instances with $Cmax \in \{560, 1125\}$. Reopt and Dijkstra average computation times in last two columns.

Cmax	Instance	Reopt			Dijkstra
		Dual	Primal	Total	
560	grid100	0.006 ± 45.4%	0.021 ± 58.0%	0.028 ± 49.5%	**0.006 ± 43.7%**
560	grid400	0.012 ± 40.3%	0.025 ± 62.8%	0.037 ± 48.8%	**0.010 ± 41.3%**
560	grid700	0.007 ± 46.2%	0.019 ± 61.5%	0.026 ± 51.8%	**0.007 ± 41.4%**
560	full-grid100	0.008 ± 44.0%	0.017 ± 61.4%	0.025 ± 49.2%	**0.007 ± 41.0%**
560	full-grid400	0.200 ± 36.0%	0.826 ± 65.5%	1.026 ± 55.2%	**0.202 ± 35.2%**
560	full-grid700	0.762 ± 34.2%	0.977 ± 81.7%	1.740 ± 55.6%	**0.576 ± 38.5%**
560	cylinder100	0.280 ± 36.1%	0.729 ± 69.0%	1.009 ± 55.7%	**0.236 ± 35.7%**
560	cylinder400	0.254 ± 35.0%	0.534 ± 75.8%	0.789 ± 58.5%	**0.415 ± 35.9%**
560	cylinder700	1.338 ± 40.2%	5.725 ± 87.6%	7.064 ± 74.0%	**1.077 ± 34.6%**
560	torus100	6.802 ± 34.4%	13.518 ± 64.8%	20.321 ± 50.0%	**3.582 ± 34.9%**
560	torus400	1.416 ± 33.6%	4.653 ± 85.3%	6.069 ± 68.8%	**1.212 ± 34.5%**
560	torus700	1.415 ± 33.4%	2.140 ± 66.8%	3.555 ± 47.8%	**2.573 ± 33.8%**
1125	grid100	0.007 ± 39.1%	0.019 ± 67.7%	0.026 ± 54.5%	**0.006 ± 41.5%**
1125	grid400	0.266 ± 34.4%	0.876 ± 67.4%	1.142 ± 56.7%	**0.149 ± 36.5%**
1125	grid700	1.122 ± 34.2%	6.312 ± 72.7%	7.434 ± 64.2%	**0.661 ± 35.6%**
1125	full-grid100	0.012 ± 40.2%	0.028 ± 76.2%	0.039 ± 60.3%	**0.010 ± 39.4%**
1125	full-grid400	0.931 ± 35.1%	1.335 ± 70.9%	2.267 ± 52.4%	**0.366 ± 35.2%**
1125	full-grid700	6.139 ± 34.0%	9.494 ± 71.5%	15.633 ± 52.5%	**2.095 ± 34.1%**
1125	cylinder100	0.007 ± 42.0%	0.020 ± 61.4%	0.027 ± 50.6%	**0.006 ± 40.7%**
1125	cylinder400	0.289 ± 34.3%	0.673 ± 69.8%	0.962 ± 55.2%	**0.159 ± 38.5%**
1125	cylinder700	1.384 ± 34.7%	5.340 ± 69.0%	6.724 ± 58.9%	**0.739 ± 34.1%**
1125	torus100	0.008 ± 37.5%	0.019 ± 69.9%	0.027 ± 56.5%	**0.008 ± 37.0%**
1125	torus400	0.297 ± 34.6%	0.455 ± 65.8%	0.752 ± 47.7%	**0.278 ± 36.4%**
1125	torus700	1.622 ± 33.7%	1.854 ± 64.3%	3.476 ± 44.2%	**1.550 ± 33.8%**

Table 2.24 (SQUARE-DECREASE). Computational results in terms of average times. Reopt and Dijkstra average computation times in last two columns.

Cmax	Instance	Reopt			Dijkstra
		Dual	Primal	Total	
75	square100	0.110 ± 35.2%	0.013 ± 78.8%	**0.123 ± 35.8%**	0.151 ± 33.5%
75	square400	94.336 ± 33.9%	0.316 ± 74.1%	**94.652 ± 33.9%**	98.673 ± 33.6%
75	square700	1310.012 ± 33.3%	0.979 ± 72.0%	**1310.991 ± 33.3%**	1350.614 ± 33.2%
75	square100+25	0.133 ± 33.4%	0.018 ± 81.5%	**0.151 ± 33.9%**	0.258 ± 33.2%
75	square400+25	105.815 ± 34.6%	0.352 ± 72.7%	**106.167 ± 34.7%**	179.233 ± 35.4%
75	square700+25	1685.317 ± 33.3%	1.447 ± 59.1%	**1686.764 ± 33.3%**	2040.346 ± 33.3%
75	square100+50	0.146 ± 33.3%	0.021 ± 75.3%	**0.168 ± 34.3%**	0.228 ± 33.2%
75	square400+50	126.525 ± 33.7%	0.511 ± 69.5%	**127.036 ± 33.7%**	181.520 ± 33.5%
75	square700+50	1921.924 ± 33.2%	1.897 ± 58.1%	**1923.822 ± 33.2%**	2425.638 ± 33.3%
150	square100	0.103 ± 33.5%	0.017 ± 72.5%	**0.120 ± 35.1%**	0.207 ± 34.1%
150	square400	87.280 ± 40.3%	0.288 ± 67.7%	87.568 ± 40.3%	**55.767 ± 35.4%**
150	square700	1265.065 ± 35.2%	0.937 ± 57.2%	1266.002 ± 35.2%	**766.663 ± 35.0%**
150	square100+25	0.144 ± 33.2%	0.020 ± 69.9%	0.164 ± 33.9%	**0.151 ± 33.2%**
150	square400+25	113.603 ± 33.4%	0.384 ± 77.9%	**113.987 ± 33.4%**	198.140 ± 33.5%
150	square700+25	1894.349 ± 33.5%	1.490 ± 74.0%	**1895.839 ± 33.5%**	2736.094 ± 33.4%
150	square100+50	0.188 ± 33.7%	0.023 ± 70.2%	0.211 ± 33.8%	**0.206 ± 33.2%**
150	square400+50	167.526 ± 33.5%	0.470 ± 71.3%	167.995 ± 33.5%	**112.214 ± 33.5%**
150	square700+50	2549.935 ± 34.1%	1.646 ± 69.0%	2551.581 ± 34.1%	**1601.081 ± 34.0%**
280	square100	0.067 ± 33.7%	0.019 ± 71.8%	**0.086 ± 35.7%**	0.169 ± 33.2%
280	square400	53.082 ± 33.5%	0.286 ± 60.3%	53.368 ± 33.5%	**36.086 ± 33.4%**
280	square700	819.157 ± 33.2%	0.860 ± 75.7%	820.018 ± 33.2%	**811.631 ± 33.2%**
280	square100+25	0.100 ± 33.5%	0.016 ± 68.5%	**0.116 ± 34.3%**	0.125 ± 33.2%
280	square400+25	88.232 ± 33.4%	0.413 ± 74.1%	**88.645 ± 33.4%**	130.988 ± 33.4%
280	square700+25	1356.643 ± 33.3%	1.431 ± 63.8%	**1358.074 ± 33.3%**	2535.271 ± 33.2%
280	square100+50	0.182 ± 34.1%	0.030 ± 86.8%	**0.212 ± 34.3%**	0.592 ± 34.2%
280	square400+50	133.883 ± 34.3%	0.446 ± 74.6%	134.330 ± 34.2%	**79.113 ± 34.2%**
280	square700+50	1833.194 ± 33.3%	1.680 ± 82.0%	1834.874 ± 33.3%	**849.244 ± 33.2%**
560	square100	0.123 ± 35.4%	0.019 ± 78.3%	**0.142 ± 36.3%**	0.221 ± 33.5%
560	square400	94.471 ± 34.2%	0.361 ± 83.5%	**94.832 ± 34.1%**	117.201 ± 34.1%
560	square700	1252.269 ± 35.2%	0.865 ± 68.0%	**1253.134 ± 35.3%**	1346.157 ± 34.1%
560	square100+25	0.104 ± 33.8%	0.021 ± 62.7%	**0.125 ± 33.7%**	0.132 ± 33.2%
560	square400+25	108.512 ± 33.6%	0.423 ± 71.4%	**108.935 ± 33.7%**	135.549 ± 33.3%
560	square700+25	1658.935 ± 35.5%	1.347 ± 55.5%	**1660.282 ± 35.5%**	2444.339 ± 34.3%
560	square100+50	0.132 ± 33.8%	0.021 ± 65.1%	**0.153 ± 33.6%**	0.195 ± 33.2%
560	square400+50	110.319 ± 33.3%	0.556 ± 66.7%	110.875 ± 33.3%	**79.134 ± 33.3%**
560	square700+50	1797.606 ± 33.4%	1.624 ± 73.2%	**1799.230 ± 33.4%**	3052.267 ± 33.4%
1125	square100	0.125 ± 33.8%	0.020 ± 73.1%	**0.145 ± 34.3%**	0.244 ± 33.3%
1125	square400	114.302 ± 33.7%	0.391 ± 66.9%	114.693 ± 33.7%	**76.322 ± 34.8%**
1125	square700	1333.036 ± 33.2%	1.038 ± 72.8%	1334.074 ± 33.2%	**825.741 ± 33.3%**
1125	square100+25	0.191 ± 35.5%	0.025 ± 100.2%	**0.216 ± 39.0%**	0.219 ± 33.9%
1125	square400+25	142.635 ± 35.0%	0.428 ± 73.0%	**143.063 ± 35.1%**	166.493 ± 33.4%
1125	square700+25	2188.806 ± 35.6%	1.558 ± 60.0%	**2190.364 ± 35.6%**	2965.140 ± 34.6%
1125	square100+50	0.234 ± 33.5%	0.021 ± 72.9%	**0.255 ± 34.1%**	0.293 ± 33.2%
1125	square400+50	166.484 ± 33.7%	0.464 ± 75.0%	166.948 ± 33.7%	**128.334 ± 33.5%**
1125	square700+50	2504.124 ± 33.4%	1.711 ± 64.5%	2505.835 ± 33.4%	**1680.483 ± 33.4%**

Table 2.25 (RECT-DECREASE). Computational results in terms of average times. Reopt and Dijkstra average computation times in last two columns.

Cmax	Instance	Reopt			Dijkstra
		Dual	Primal	Total	
75	rect100	0.495 ± 34.1%	0.035 ± 74.1%	**0.530 ± 33.9%**	0.626 ± 34.1%
75	rect300	118.452 ± 34.4%	0.355 ± 75.5%	**118.807 ± 34.4%**	124.939 ± 34.2%
75	rect500	1524.745 ± 36.1%	1.362 ± 66.6%	1526.107 ± 36.1%	**1474.774 ± 35.0%**
75	rect100+25	0.589 ± 33.2%	0.037 ± 75.7%	**0.626 ± 33.4%**	1.098 ± 33.2%
75	rect300+25	145.361 ± 35.8%	0.458 ± 71.1%	**145.819 ± 35.8%**	227.536 ± 34.1%
75	rect500+25	1694.583 ± 33.5%	1.484 ± 74.1%	**1696.067 ± 33.5%**	2410.733 ± 33.4%
75	rect100+50	0.704 ± 33.7%	0.043 ± 76.4%	**0.748 ± 34.1%**	1.273 ± 33.2%
75	rect300+50	192.566 ± 33.9%	0.602 ± 74.2%	**193.168 ± 33.9%**	319.656 ± 33.8%
75	rect500+50	1894.340 ± 33.5%	1.673 ± 61.4%	**1896.013 ± 33.5%**	2860.115 ± 33.4%
150	rect100	0.488 ± 36.6%	0.032 ± 75.3%	**0.520 ± 36.9%**	0.639 ± 34.3%
150	rect300	123.501 ± 35.0%	0.378 ± 70.8%	123.878 ± 35.0%	**73.088 ± 34.5%**
150	rect500	1305.529 ± 34.6%	1.076 ± 76.6%	1306.605 ± 34.6%	**804.255 ± 33.8%**
150	rect100+25	0.638 ± 33.2%	0.038 ± 64.4%	**0.676 ± 33.3%**	0.681 ± 33.2%
150	rect300+25	161.098 ± 35.3%	0.454 ± 82.0%	**161.552 ± 35.3%**	269.797 ± 34.7%
150	rect500+25	1956.525 ± 33.3%	1.371 ± 64.1%	**1957.895 ± 33.3%**	3056.855 ± 34.3%
150	rect100+50	0.889 ± 33.5%	0.047 ± 83.7%	**0.936 ± 33.6%**	0.977 ± 33.4%
150	rect300+50	259.283 ± 35.0%	0.586 ± 54.9%	259.869 ± 35.0%	**165.204 ± 35.6%**
150	rect500+50	2558.590 ± 34.2%	1.735 ± 73.0%	2560.324 ± 34.2%	**1636.379 ± 34.3%**
280	rect100	0.321 ± 35.5%	0.033 ± 82.6%	**0.354 ± 35.3%**	0.742 ± 33.3%
280	rect300	78.139 ± 33.9%	0.357 ± 67.6%	78.496 ± 34.0%	**55.322 ± 33.7%**
280	rect500	845.569 ± 33.2%	1.081 ± 65.9%	846.650 ± 33.2%	**591.691 ± 33.3%**
280	rect100+25	0.493 ± 33.5%	0.039 ± 80.2%	0.531 ± 33.4%	**0.448 ± 33.6%**
280	rect300+25	127.084 ± 36.5%	0.459 ± 74.9%	**127.543 ± 36.4%**	196.244 ± 38.0%
280	rect500+25	1432.206 ± 33.5%	1.444 ± 70.0%	**1433.650 ± 33.5%**	1912.192 ± 33.5%
280	rect100+50	0.718 ± 34.5%	0.069 ± 71.6%	**0.788 ± 34.0%**	2.328 ± 33.6%
280	rect300+50	167.615 ± 34.3%	0.507 ± 63.9%	168.122 ± 34.3%	**82.953 ± 33.3%**
280	rect500+50	1948.265 ± 33.3%	1.680 ± 63.5%	1949.945 ± 33.3%	**928.307 ± 33.3%**
560	rect100	0.550 ± 33.6%	0.046 ± 73.1%	**0.596 ± 34.2%**	0.761 ± 33.9%
560	rect300	148.208 ± 40.3%	0.321 ± 86.6%	**148.529 ± 40.2%**	170.018 ± 36.0%
560	rect500	1233.865 ± 33.6%	0.995 ± 66.3%	**1234.860 ± 33.6%**	1446.891 ± 33.3%
560	rect100+25	0.524 ± 37.2%	0.044 ± 88.2%	**0.567 ± 37.9%**	0.650 ± 39.0%
560	rect300+25	127.868 ± 34.8%	0.421 ± 54.9%	128.289 ± 34.8%	**100.050 ± 45.8%**
560	rect500+25	1724.029 ± 35.8%	1.191 ± 57.8%	**1725.220 ± 35.8%**	1964.684 ± 34.1%
560	rect100+50	0.596 ± 33.2%	0.052 ± 66.6%	**0.648 ± 33.5%**	0.851 ± 33.2%
560	rect300+50	146.031 ± 33.3%	0.542 ± 71.9%	146.573 ± 33.3%	**105.584 ± 33.3%**
560	rect500+50	2140.742 ± 35.7%	2.136 ± 74.7%	**2142.878 ± 35.7%**	3172.148 ± 33.9%
1125	rect100	0.639 ± 35.1%	0.056 ± 65.8%	**0.695 ± 35.9%**	1.207 ± 34.5%
1125	rect300	185.873 ± 37.3%	0.532 ± 68.0%	186.405 ± 37.3%	**122.081 ± 36.9%**
1125	rect500	1452.069 ± 33.7%	1.071 ± 68.0%	1453.140 ± 33.7%	**902.778 ± 33.7%**
1125	rect100+25	0.956 ± 35.0%	0.059 ± 56.8%	1.015 ± 34.3%	**0.972 ± 33.9%**
1125	rect300+25	238.028 ± 39.7%	0.548 ± 60.3%	**238.576 ± 39.7%**	280.047 ± 38.8%
1125	rect500+25	1963.700 ± 33.4%	1.237 ± 77.3%	**1964.937 ± 33.5%**	2147.710 ± 33.3%
1125	rect100+50	1.153 ± 34.3%	0.064 ± 86.1%	**1.217 ± 35.0%**	1.358 ± 33.8%
1125	rect300+50	264.949 ± 35.4%	0.548 ± 69.7%	265.498 ± 35.4%	**193.905 ± 34.7%**
1125	rect500+50	3006.409 ± 34.2%	1.641 ± 63.3%	3008.049 ± 34.2%	**1952.012 ± 34.0%**

Table 2.26 (SPACYC-DECREASE). Computational results in terms of average times. Reopt and Dijkstra average computation times in last two columns.

Cmax	Instance	Reopt			Dijkstra
		Dual	Primal	Total	
75	spacyc-4n100	0.018 ± 33.6%	0.026 ± 82.0%	0.044 ± 55.9%	**0.023 ± 33.8%**
75	spacyc-4n400	7.623 ± 44.5%	0.695 ± 104.0%	8.318 ± 45.1%	**6.540 ± 39.4%**
75	spacyc-4n700	59.192 ± 35.2%	1.230 ± 80.0%	**60.422 ± 34.8%**	80.617 ± 38.3%
75	spacyc-6n100	0.050 ± 34.7%	0.018 ± 72.8%	0.069 ± 36.1%	**0.036 ± 35.0%**
75	spacyc-6n400	8.118 ± 33.5%	0.528 ± 80.6%	8.646 ± 34.0%	**8.303 ± 33.6%**
75	spacyc-6n700	262.037 ± 37.5%	1.515 ± 71.9%	263.551 ± 37.3%	**129.927 ± 36.4%**
75	spacyc-8n100	0.063 ± 34.1%	0.032 ± 75.7%	0.095 ± 42.8%	**0.064 ± 34.9%**
75	spacyc-8n400	22.373 ± 37.5%	0.625 ± 83.3%	22.998 ± 37.7%	**15.756 ± 46.1%**
75	spacyc-8n700	486.422 ± 33.3%	1.689 ± 58.7%	488.111 ± 33.3%	**196.415 ± 33.9%**
150	spacyc-4n100	0.017 ± 33.7%	0.026 ± 55.2%	0.044 ± 42.7%	**0.024 ± 33.8%**
150	spacyc-4n400	5.214 ± 54.8%	0.542 ± 72.3%	5.757 ± 51.7%	**5.425 ± 35.9%**
150	spacyc-4n700	43.884 ± 33.5%	1.256 ± 74.0%	**45.140 ± 33.5%**	55.922 ± 33.7%
150	spacyc-6n100	0.043 ± 34.5%	0.032 ± 71.4%	0.076 ± 45.0%	**0.030 ± 35.9%**
150	spacyc-6n400	7.235 ± 33.8%	0.546 ± 71.2%	**7.781 ± 34.1%**	7.954 ± 33.6%
150	spacyc-6n700	176.223 ± 33.2%	1.506 ± 64.6%	177.730 ± 33.2%	**113.561 ± 33.5%**
150	spacyc-8n100	0.073 ± 33.6%	0.040 ± 83.2%	0.112 ± 41.9%	**0.063 ± 33.5%**
150	spacyc-8n400	18.603 ± 34.8%	0.452 ± 61.7%	19.055 ± 34.5%	**13.420 ± 35.2%**
150	spacyc-8n700	426.102 ± 33.7%	1.666 ± 65.4%	427.768 ± 33.7%	**202.384 ± 33.8%**
280	spacyc-4n100	0.017 ± 33.4%	0.026 ± 70.6%	0.043 ± 50.5%	**0.020 ± 33.3%**
280	spacyc-4n400	5.951 ± 36.1%	0.381 ± 82.2%	6.332 ± 36.6%	**4.660 ± 33.6%**
280	spacyc-4n700	42.935 ± 34.4%	1.160 ± 62.6%	**44.095 ± 34.5%**	50.703 ± 34.0%
280	spacyc-6n100	0.045 ± 33.3%	0.024 ± 58.2%	0.070 ± 37.4%	**0.044 ± 33.6%**
280	spacyc-6n400	5.958 ± 33.9%	0.506 ± 65.5%	**6.463 ± 34.5%**	7.498 ± 33.6%
280	spacyc-6n700	184.378 ± 33.5%	1.641 ± 62.3%	186.019 ± 33.4%	**109.308 ± 33.2%**
280	spacyc-8n100	0.062 ± 33.7%	0.031 ± 84.9%	0.093 ± 43.7%	**0.065 ± 33.4%**
280	spacyc-8n400	16.022 ± 34.6%	0.609 ± 74.1%	16.632 ± 34.3%	**9.383 ± 33.5%**
280	spacyc-8n700	374.677 ± 33.5%	1.383 ± 79.2%	376.060 ± 33.5%	**145.232 ± 33.2%**
560	spacyc-4n100	0.018 ± 33.4%	0.027 ± 64.9%	0.045 ± 47.2%	**0.025 ± 33.3%**
560	spacyc-4n400	6.308 ± 33.3%	0.443 ± 67.2%	6.750 ± 33.4%	**5.824 ± 33.3%**
560	spacyc-4n700	38.932 ± 50.6%	1.168 ± 79.5%	**40.101 ± 50.8%**	61.262 ± 34.1%
560	spacyc-6n100	0.049 ± 35.2%	0.019 ± 64.5%	0.069 ± 39.0%	**0.038 ± 35.8%**
560	spacyc-6n400	8.576 ± 38.1%	0.733 ± 80.2%	**9.309 ± 39.6%**	9.874 ± 39.6%
560	spacyc-6n700	242.868 ± 33.5%	1.381 ± 58.4%	244.249 ± 33.5%	**119.785 ± 33.3%**
560	spacyc-8n100	0.060 ± 33.9%	0.026 ± 56.7%	0.086 ± 35.7%	**0.050 ± 35.3%**
560	spacyc-8n400	16.226 ± 33.4%	0.485 ± 68.6%	16.711 ± 33.5%	**12.979 ± 33.9%**
560	spacyc-8n700	490.081 ± 33.2%	1.764 ± 70.6%	491.845 ± 33.2%	**214.433 ± 33.3%**
1125	spacyc-4n100	0.015 ± 49.6%	0.027 ± 69.1%	0.042 ± 57.1%	**0.024 ± 33.2%**
1125	spacyc-4n400	5.015 ± 53.3%	0.500 ± 84.1%	5.515 ± 52.6%	**5.432 ± 36.0%**
1125	spacyc-4n700	38.677 ± 34.3%	1.347 ± 70.1%	**40.024 ± 34.4%**	54.455 ± 35.0%
1125	spacyc-6n100	0.059 ± 39.6%	0.032 ± 66.5%	0.091 ± 45.1%	**0.043 ± 42.5%**
1125	spacyc-6n400	10.770 ± 37.2%	0.768 ± 68.4%	**11.538 ± 35.7%**	12.500 ± 37.4%
1125	spacyc-6n700	212.261 ± 35.7%	1.414 ± 73.4%	213.675 ± 35.6%	**130.887 ± 38.0%**
1125	spacyc-8n100	0.074 ± 33.5%	0.049 ± 76.0%	0.123 ± 43.3%	**0.070 ± 33.5%**
1125	spacyc-8n400	17.283 ± 33.5%	0.496 ± 70.4%	17.779 ± 33.6%	**11.438 ± 33.2%**
1125	spacyc-8n700	538.547 ± 33.6%	1.763 ± 87.1%	540.310 ± 33.6%	**236.424 ± 34.0%**

Table 2.27 (SPRAND-DECREASE). Computational results in terms of average times. Reopt and Dijkstra average computation times in last two columns.

Cmax	Instance	Reopt			Dijkstra
		Dual	Primal	Total	
75	sprand-4n100	0.050 ± 35.8%	0.016 ± 62.2%	**0.066 ± 36.6%**	0.145 ± 34.4%
75	sprand-4n400	40.792 ± 42.7%	0.334 ± 72.6%	**41.126 ± 42.4%**	69.538 ± 44.2%
75	sprand-4n700	468.952 ± 50.1%	1.148 ± 66.9%	**470.100 ± 50.0%**	1689.737 ± 33.3%
75	sprand-6n100	0.076 ± 34.7%	0.022 ± 64.6%	**0.098 ± 34.9%**	0.254 ± 33.5%
75	sprand-6n400	57.238 ± 35.6%	0.353 ± 97.1%	**57.591 ± 35.5%**	72.376 ± 33.6%
75	sprand-6n700	828.797 ± 50.5%	1.543 ± 73.3%	**830.341 ± 50.5%**	3809.134 ± 33.8%
75	sprand-8n100	0.094 ± 33.4%	0.024 ± 83.6%	**0.118 ± 37.1%**	0.126 ± 33.3%
75	sprand-8n400	78.212 ± 33.3%	0.479 ± 69.7%	**78.691 ± 33.3%**	319.044 ± 37.3%
75	sprand-8n700	1301.797 ± 33.5%	1.697 ± 59.0%	**1303.494 ± 33.5%**	3404.807 ± 33.4%
150	sprand-4n100	0.059 ± 33.6%	0.017 ± 62.8%	**0.075 ± 35.1%**	0.091 ± 33.4%
150	sprand-4n400	39.436 ± 33.4%	0.297 ± 81.5%	**39.734 ± 33.5%**	74.084 ± 34.0%
150	sprand-4n700	733.781 ± 33.2%	1.268 ± 60.9%	**735.049 ± 33.2%**	2060.019 ± 33.4%
150	sprand-6n100	0.115 ± 33.6%	0.017 ± 61.1%	**0.132 ± 33.8%**	0.160 ± 33.5%
150	sprand-6n400	89.647 ± 33.9%	0.360 ± 73.8%	**90.006 ± 33.9%**	140.657 ± 34.6%
150	sprand-6n700	1450.732 ± 33.8%	1.465 ± 70.3%	**1452.197 ± 33.8%**	2299.779 ± 33.4%
150	sprand-8n100	0.171 ± 40.3%	0.024 ± 71.1%	**0.195 ± 38.4%**	0.283 ± 34.0%
150	sprand-8n400	141.811 ± 34.1%	0.519 ± 67.9%	**142.330 ± 34.1%**	208.749 ± 33.9%
150	sprand-8n700	2226.395 ± 37.1%	2.216 ± 79.9%	**2228.611 ± 37.2%**	3110.290 ± 35.4%
280	sprand-4n100	0.040 ± 50.2%	0.012 ± 90.1%	**0.052 ± 50.9%**	0.066 ± 33.4%
280	sprand-4n400	31.417 ± 33.3%	0.305 ± 58.2%	**31.722 ± 33.2%**	119.886 ± 33.7%
280	sprand-4n700	599.852 ± 33.5%	1.144 ± 74.1%	**600.996 ± 33.5%**	1628.262 ± 33.5%
280	sprand-6n100	0.094 ± 33.3%	0.016 ± 64.0%	**0.110 ± 34.7%**	0.171 ± 33.5%
280	sprand-6n400	73.607 ± 33.7%	0.416 ± 68.2%	**74.023 ± 33.7%**	168.366 ± 33.9%
280	sprand-6n700	1337.212 ± 34.5%	1.634 ± 80.2%	**1338.846 ± 34.5%**	1561.865 ± 35.0%
280	sprand-8n100	0.136 ± 33.4%	0.023 ± 66.9%	**0.159 ± 34.7%**	0.166 ± 33.6%
280	sprand-8n400	126.243 ± 35.3%	0.478 ± 68.2%	**126.721 ± 35.2%**	223.062 ± 34.8%
280	sprand-8n700	2194.818 ± 33.6%	2.196 ± 71.8%	**2197.014 ± 33.6%**	7500.313 ± 33.4%
560	sprand-4n100	0.040 ± 50.0%	0.016 ± 56.3%	**0.055 ± 48.9%**	0.086 ± 33.5%
560	sprand-4n400	32.983 ± 33.9%	0.360 ± 68.1%	**33.342 ± 33.8%**	112.501 ± 33.6%
560	sprand-4n700	567.419 ± 33.8%	0.968 ± 80.8%	**568.386 ± 33.8%**	1377.740 ± 33.5%
560	sprand-6n100	0.076 ± 33.4%	0.020 ± 67.0%	**0.096 ± 35.0%**	0.135 ± 33.5%
560	sprand-6n400	61.269 ± 34.0%	0.383 ± 77.1%	**61.651 ± 34.0%**	170.567 ± 34.3%
560	sprand-6n700	977.168 ± 33.8%	1.408 ± 62.9%	**978.576 ± 33.8%**	1928.951 ± 33.4%
560	sprand-8n100	0.098 ± 33.5%	0.022 ± 77.2%	0.120 ± 36.5%	**0.111 ± 33.2%**
560	sprand-8n400	87.648 ± 33.4%	0.431 ± 81.2%	**88.079 ± 33.5%**	153.147 ± 33.7%
560	sprand-8n700	1526.437 ± 33.3%	1.947 ± 83.1%	**1528.384 ± 33.3%**	7140.586 ± 33.3%
1125	sprand-4n100	0.055 ± 33.8%	0.015 ± 72.9%	**0.071 ± 36.0%**	0.098 ± 33.4%
1125	sprand-4n400	39.822 ± 33.3%	0.361 ± 68.9%	**40.183 ± 33.3%**	124.969 ± 34.2%
1125	sprand-4n700	746.223 ± 33.3%	1.254 ± 83.1%	**747.477 ± 33.3%**	1763.083 ± 33.5%
1125	sprand-6n100	0.107 ± 33.7%	0.020 ± 90.7%	**0.127 ± 37.8%**	0.158 ± 33.4%
1125	sprand-6n400	76.868 ± 33.4%	0.478 ± 72.1%	**77.346 ± 33.4%**	132.561 ± 34.6%
1125	sprand-6n700	1484.569 ± 34.7%	1.453 ± 72.6%	**1486.022 ± 34.7%**	2264.907 ± 33.8%
1125	sprand-8n100	0.155 ± 34.3%	0.026 ± 77.8%	**0.180 ± 36.6%**	0.264 ± 34.8%
1125	sprand-8n400	134.701 ± 35.3%	0.503 ± 64.2%	**135.204 ± 35.3%**	200.561 ± 34.9%
1125	sprand-8n700	2379.289 ± 33.3%	1.656 ± 80.3%	**2380.945 ± 33.3%**	4270.195 ± 33.3%

CHAPTER 3

The Reoptimization of Shortest Paths in Urban Air Mobility

In this Chapter we address both the design of models for *Urban Air Mobility* and the resolution of the corresponding Shortest Path Problems. Indeed, in Fugaro [87] we devised three different discretization approaches to give a graph structure to the 3D space in which the motion is supposed to happen. Then, the two-steps resolution of SPPs is tackled by exploiting the Reoptimization algorithm presented in Chapter 2.

In Section 3.1 we outline the main aspects of designing and solving models in Urban Air Mobility while the three discretization approaches are presented in Section 3.2. Then, a complete presentation of the computational experiments is given in Section 3.3 and Appendix 3.B. In addition, Section 3.4 is devoted to the presentation of the trajectories related to the discretization approaches.

3.1 Urban Air Mobility: Motivations and Design of Models

"Urban Air Mobility (UAM) presents an opportunity to decongest road traffic, improve mobility, give back time to those trapped in daily traffic, and unlock the potential of cities worldwide." [1] Indeed, it could be employed also for emergency medical evacuations, rescue operations, humanitarian missions, weather monitoring [170]. Therefore, the development of UAM represents a flourishing stream of research: companies are adapting to this trend for the transportation as well as for delivery purposes, e.g. Amazon® has announced the launching of *Amazon Prime Air*, a fast delivery service with drones.

[1] https://www.uber.com/us/en/elevate/uberair/

However, the development of efficient systems, capable to exploit the benefits of UAM, is very challenging: among the others [2], one of the principal issues to address is the realization of well suited – electrical/hybrid – vehicles whose use in air traffic might have the lowest possible impact, in terms of pollution and noise [116]. Anyway, the most relevant and challenging aspect to consider when designing models for UAM, consists in the *Path Planning* [15, 173].

Specifically, this procedure consists in two phases: i) the discretization of the obstacle-free 3D space in which the aircraft is going to fly, and ii) the definition of the admissible trajectories. On the one hand, different strategies exist to detect and depict the available space from the map of the buildings in the city [11]; consequently, different environments results from different assumptions. On the other hand, though trajectories are generally computed as shortest paths according to specific cost criteria, the definition of the cost functions is rather than simple. In fact, they should take into account the energy consumption as well as penalty factors related to the pollution and noise produced by the aircraft, but also to the atmospheric conditions which may favor or hinder the motion of vehicles [114]. Additionally, due the complex nature of these cost functions, some of the information characterizing them might be available only as estimates.

In this scenario, the resolution of the shortest path problems related to the computation of the best trajectories is significantly complicated. However, a suitable approach consists in the *two-steps resolution*: at first, the shortest paths are computed *offline*, i.e. considering as deterministic the initial information, and then there is the switch to the *online* environment, updating the information and eventually the solution too. In particular, this approach shares similarities with the reoptimization framework depicted in Chapter 2: in fact, the discretization defines an underlying graph structure for the obstacle-free space; then, switching from the offline to the online environment yields to two slightly different graphs on which the same problem has to be solved.

With these considerations in mind, in Fugaro [87] we address both the issues related to the Path Planning in UAM modelling. Specifically, we design three discretization approaches which discretize the 3D space proceeding by layers, i.e. plane sections, and defining a particular graph topology, grid or random, on each layer, before connecting them appropriately (Section 3.2). Furthermore, we test whether the reoptimization algorithmic framework [83, 144] might be advisable for the computation of the best routes or the resolution from scratch in the online step has to be preferred (Section 3.3). In the remaining of this Chapter, the terms aircraft and "VTOL" i.e. *vertical take-off and landing aircraft* will be used as synonyms.

3.2 Discretization Approaches

This Section is devoted to thoroughly describe the three discretization approaches we propose to address the first phase of Path Planning in UAM [87].

In order to compute the aircraft trajectories, it is above all necessary to have

a detailed representation of the 3D space in which the aircraft can fly. As for the *motion planning*, several approaches have been employed in the scientific literature [36, 102, 140]; indeed, the 3D space representations can be classified in the following four categories [11]:

Object Oriented Maps. The obstacles are represented with polygons (in 2D) or polyhedrons (in 3D) in order to specify which regions should be avoided, due to the presence of obstacles.

Free-Space Maps. A graph is used to represent the environment and nodes are assigned to the free of obstacles regions, thus supporting the navigation from one free space node to another.

Composite-Space Maps. The space is represented through a grid with rectangular cells, each marked as "obstacle" or "non-obstacle" .

Paths Maps. The routes are computed with a set of predefined paths.

The three discretization methods proposed in Fugaro [87] are Free-Space Maps obtained by defining: i) a vertical sectioning of the 3D space, and ii) a graph structure on each plane section.

Specifically, the 3D map of the city is inserted in a three dimensional Cartesian coordinate system, with origin O and axis lines X, Y and Z; then, the buildings are enclosed in parallelepipeds whose base faces are parallel to the (X, Y) plane.

Remark 4. In order to approximate the structure of a building with a parallelepiped, the polygonal base of the building in the (X, Y) plane is considered: given its n vertices $(x_i, y_i)_{i=1,\dots n}$, those of the base of the parallelepiped in the same plane, namely $(\bar{x}_i, \bar{y}_i)_{i=1,\dots 4}$, are obtained by picking the minimum and maximum values for each vertices coordinates and combining them [87]. □

Additionally, before defining the vertical sections, a further *pre-processing* operation is performed; in fact, for safety reasons, the aircraft must maintain a specific safe distance from the side and top faces of buildings. Thus, two parallelepipeds, i.e. two bounded buildings, are merged whenever the free space between them does not allow the aircraft to fly safely. In order to obtain different configurations, the pre-processing operations are performed according to one of the following strategies [87].

1. Parallelepipeds are merged independently of the value of the difference between their heights. The height of the resulting parallelepiped is that of the highest of the two buildings.

2. Parallelepipeds are merged only if the value of the difference between their heights is less than the double sum of the aircraft height and the *vertical safety distance*.

Finally, the discretization is performed according to one of the methods described in the remaining of this Section. Specifically, the first two approaches, namely the *Discretization via Grids* (Section 3.2.1) and the *Discretization via*

Random Graphs (Section 3.2.2), differ in the type of topology chosen for the graphs built on plane sections, while the *One-Layer Discretization* (Section 3.2.3), consists in the construction of random graph on a single plane section which is then properly connected with the the source and destination nodes of the shortest paths.

3.2.1 The Discretization via Grids

This method constructs the Free-Space Map of the city by first discretizing the whole 3D space and then performing a post-processing phase in order to remove areas occupied by the buildings. In this way, the overall structure of the obstacle-free space resembles that of a layered graph.

Specifically, a *grid graph* is built on each layer, where the grid topology is analogous to the one described in Section 2.4.2. Then, nodes and arcs falling inside the plane sections of parallelepipeds are removed from the obtained environment. Finally, all the layers are connected together, both upward and downward, by defining arcs between each node and the corresponding on the upper level.

3.2.2 The Discretization via Random Graphs

This method constructs the Free-Space Map of the city similarly to the one of Section 3.2.1: indeed, a *random graph* is built on each layer from the skeleton structure of a grid, by defining the connections randomly.

Specifically, for each node the cardinality of the forward star is set to the one it would have had in the grid, and the arcs are defined by connecting the node with randomly chosen nodes. Then, apart from removing nodes and arcs within the plane sections of the parallelepipeds, a *feasibility check* of the set of arcs on each layer is executed to remove the arcs crossing these plane sections. Further details on the adopted procedure are given in Appendix 3.A. Finally, the layers are connected upward and downward.

3.2.3 The One-Layer Discretization

The idea behind this discretization method is to let the aircraft fly above the highest buildings. Consequently, the Free-Space Map of the city consists in a discretization of the plane section above the tallest of buildings.

In particular, the graph topology is defined as done for the method in Section 3.2.2 while origins and destinations of the routes – located on the roof of some buildings – are connected with the corresponding nodes on the plane section with upward vertical arcs.

3.3 Computational Experiments

As thoroughly motivated in Section 3.1, the computation of the most promising trajectories is rather than simple. In Fugaro [87], we addressed this issue with the two-steps resolution; specifically, once the shortest paths have been computed, we apply a local perturbation to the graph cost function to simulate the effects of switching from offline to online. Then, we perform a computational experimentation to determine whether the reoptimization approach (see Chapter 2) is advisable to update the current solutions or the modified problems have to be solved from scratch. Indeed, the aim of this experimentation is twofold, since it is also meant to determine which of the discretization methods presented in Section 3.2 produces the most realistic environment.

3.3.1 Test Problems

Since for the study conducted in Fugaro [87] high-level real world data were not available, the three discretization approaches of Section 3.2 have been tested on a *toy city* created ad hoc. Specifically, Fig. 3.1 shows its plan, while Fig. 3.2 shows the 3D plan obtained once that building have been enclosed in parallelepipeds (cfr. Section 3.2).

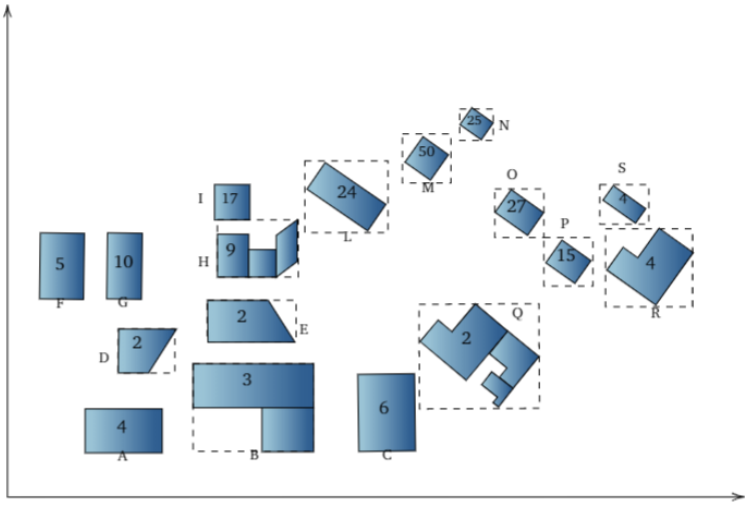


Fig. 3.1 Plan of the toy city. Numbers inside blue polygons represent floors of the corresponding building.

In order to compute the routes it is necessary to define the location of *skyports* which are the take-off and landing points. To ensure that passengers or goods are loaded on board the aircraft, these skyports are generally located on the roof of buildings. Thus, in order to depict different scenarios, we consider six couples

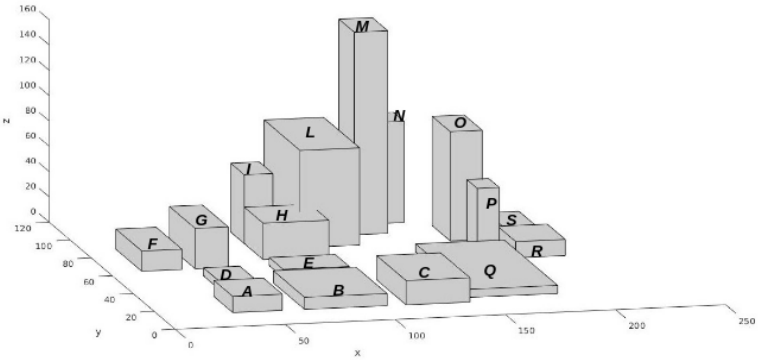


Fig. 3.2 3D plan of the toy city. Gray parallelepipeds represent the bounding boxes of buildings.

of skyports: (A,R), (B,R), (B,S), (F,S), (G,P) and (P,I).

In particular, the position of the skyports determines the height of the first and the last plane sections: in fact, the first layer is fixed in correspondence of the lowest height $\bar{h}$ between those of the skyports. Then, to define the remaining plane sections, we consider the vector $h = (h_1, h_2, \dots h_t)$ such that: a) $h_i \geq \bar{h}$ for all $i \leq t$; b) $h_i \neq h_j$ for all $i, j \leq t, i \neq j$; c) $h_i \leq h_{i+1}$ for all $i \leq t-1$ and set a layer i.e. a plane section at height $h_i + \delta$ whenever $h_i - h_{i-1} > \delta$, where δ is constant including the VTOL height and the vertical safety distance.

Finally, the pre-processing operations have been performed with both the strategies described in Section 3.2 in order to depict different configurations.

With these assumptions and applying the procedures of Section 3.2, the following four data-sets have been obtained. In particular, the different types of graph on each layer were built adapting the generator described in Festa and Pallottino [80].

1-2. `RAND-GRID` instances; `RAND`-instances. The former were generated with the approach of Section 3.2.1, the latter with the approach of Section 3.2.2. The cost function takes into account energy/fuel consumption, travel time and maximum distance covered by the VTOL. In particular, it assigns random integer cost from the set $\{1, 2, \dots Cmax\}$ to the arcs of each layer and cost equal to $Cmax$ times the difference in altitude between two successive sections to the vertical arcs, i.e. those connecting the layers. This choice aims to discourage a high variation of altitudes (in order to reduce the energy consumption).

3. `DIST`-instances. They were generated with a refinement of the Discretization via Random Graphs (Section 3.2.2): on the one hand, the density of the random graph on each layer is increased; on the other hand, a *distance-*

dependent cost function is defined. Specifically, we set cardinality $CARD = 4, 8, 20$ for the forward star of the inner nodes; the corresponding instances are indicated with prefixes `4-dist`, `8-dist` and `20-dist`. Additionally, the cost of an arc (i, j) is proportional to the product of the Euclidean distance between i and j in the (X, Y) plane.

4. `ONE-LAYER` instances. They were obtained with the approach of Section 3.2.3, by letting cardinality for the forward star of the inner nodes vary from 4 to 20. The corresponding instances are indicated with prefixes `4-ol`, `8-ol` and `20-ol`. The cost function assigns cost c_{ij} to an arc (i, j) which is the Euclidean distance between i and j in the (X, Y) plane.

Table 3.1 Details of the data-sets `RAND-GRID` and `RAND`.

RAND-GRID				RAND			
Instance	**Layers**	**Nodes**	**Arcs**	**Instance**	**Layers**	**Nodes**	**Arcs**
heights-AR	11	104313	405346	rand-heights-AR	11	104313	301833
heights-NC-AR	11	104313	404746	rand-heights-NC-AR	11	104313	301408
AR	7	66381	254982	rand-AR	7	66381	196548
NC-AR	7	66381	254622	rand-NC-AR	7	66381	196505
heights-BR	12	113796	441434	rand-heights-BR	12	113796	322308
heights-NC-BR	12	113796	440834	rand-heights-NC-BR	12	113796	321675
BR	8	75864	290731	rand-BR	8	75864	218992
NC-BR	8	75864	290371	rand-NC-BR	8	75864	218031
heights-BS	12	113796	441070	rand-heights-BS	12	113796	318102
heights-NC-BS	12	113796	440636	rand-heights-NC-BS	12	113796	317848
BS	8	75864	288845	rand-BS	8	75864	216930
NC-BS	8	75864	288605	rand-NC-BS	8	75864	216476
heights-FS	11	104313	405412	rand-heights-FS	11	104313	300311
heights-NC-FS	11	104313	404978	rand-heights-NC-FS	11	104313	300217
FS	8	75864	286076	rand-FS	8	75864	217899
NC-FS	8	75864	285912	rand-NC-FS	8	75864	217818
heights-GP	7	66381	259834	rand-heights-GP	7	66381	243892
heights-NC-GP	7	66381	259704	rand-heights-NC-GP	7	66381	243519
GP	5	47415	183650	rand-GP	5	47415	175669
NC-GP	5	47415	183572	rand-NC-GP	5	47415	175856
heights-PI	6	56898	222975	rand-heights-PI	6	56898	217247
heights-NC-PI	6	56898	222831	rand-heights-NC-PI	6	56898	217103
PI	4	37932	147136	rand-PI	4	37932	147627
NC-PI	4	37932	147064	rand-NC-PI	4	37932	147465

Indeed, a preliminary testing revealed that some of the layered graphs of data-sets 1., 2. and 3. were not connected; thus, all the data-sets have been further enriched with discretized environments in which the landing skyport is connected with all the layers above the building on which it is situated. The details about data-sets `RAND-GRID` and `RAND` are given in Table 3.1. In particular, the "-heights" instances have been obtained with the second pre-processing strategy while the "NC-" instances are those in which the destination skyport is not connected with all the remaining layers. Finally, we mention that the average construction time is equal to 0.32 seconds for the former data-set and 0.02 seconds for the latter.

3.3.2 Experiments Setup

In order to simulate the update of the cost information when switching from the offline to the online environment, in Fugaro [87] we devised two types of perturbation for the cost function.

Specifically, `PERTURBATION 1` simulates a generic change of the cost information and was applied on each instance of `RAND-GRID`, `RAND` and `DIST` data-sets.

`PERTURBATION 1:` for each chosen arc (i, j) with cost c_{ij}, a coin toss decides whether to increase or decrease its cost. In the first case, the modified cost is equal to $c_{ij} + D$ where D is an integer chosen at random in the set $\{1, \ldots, Cmax\}$; otherwise, the modified cost is equal to R where R is an integer chosen at random in the set $\{1, 2, \ldots, c_{ij}\}$.

The changes were applied to the arcs in the forward star of k randomly chosen nodes, where $k = 10, 25, 50, 75, 100$; as explained in Section 2.4.3 this choice depends on both practical and theoretical motivations. Actually, in general, the changes may be localized only in small portions of the networks and when the cost change affects random points of the network it is typically not necessary to update the current trajectories. Table 3.2 summarizes the total number of test problems.

Table 3.2 Test problems obtained applying `PERTURBATION 1`.

Data-set	Rule	#Problems	#Mod	Tot.
RAND-GRID	1	12	60	
	2	12	60	120
RAND	1	12	60	
	2	12	60	120
DIST	1	36	180	
	2	36	180	360
				600

Later on, in order to investigate a specific scenario, we left the cost perturbation model a local change in the network due to atmospheric factors, specifically the wind that affects a given area of the city. This topic is of great interest when the *STOL* vehicles i.e. "short takeoff and landing" air-crafts are employed: due to their short runway requirements for takeoff and landing, they are particularly susceptible to both crosswinds and gusting winds [114].

At this purpose, the experimentation was conducted on `ONE-LAYER` data-set: due to the random nature of the connections in these networks, `PERTURBATION 1` is not suitable to simulate the above mentioned scenario, since it could potentially affect points that are extremely far. Consequently, letting x be the centre of the skeleton grid, and $N_{2r}(x)$ be a square neighbourhood of x with side $2r$, we set $r = 2 + 5k$ with $k = 1, 2, \ldots 8$. Then, for each value of r the following cost perturbation was applied to the graph.

PERTURBATION 2: the cost c_{ij} of all the arcs (i, j) such that $i, j \in N_{2r}(x)$ are modified: c_{ij} is decreased (increased) of a value equal to $c_{ij}/2$ whenever the direction of the arc is equal (inverse) to that of the wind.

In particular, the specific change defining `PERTURBATION 2` was meant to take into account both the distance between i and j and the direction of the wind. Table 3.3 summarizes the total number of test problems obtained considering two opposite directions for the wind, namely West to East and East to West.

Table 3.3 Test problems obtained applying `PERTURBATION 2`.

Data-set	Wind	#Prob
ONE-LAYER	West-East	162
	East-West	162
		324

3.3.3 Results

In this Section we present and discuss the results of the computational experiments conducted in Fugaro [87] in order to determine whether the reoptimization approach is advisable to update the solution when switching from the offline to the online environment.

At this purpose, the reoptimization is performed with the algorithm `Reopt` thoroughly described in Chapter 2, while Dijkstra's well-known label setting procedure [56] – namely `Dijkstra`– is adopted for the resolution from scratch of the online problem. Both `Reopt` and `Dijkstra` algorithms have been coded in C, compiled with gcc 8.3.0 and tested by using an Intel® core™ i7-5500U, 2.40 GHz, RAM 8.00 GB, under a Ubuntu 19.10 operating system.

The results have been divided according to the type of perturbation applied to the graph: Tables 3.4 - 3.7 refer to `PERTURBATION 1`; instead, Tables 3.8 - 3.13 refer to the `PERTURBATION 2`. All the times are expressed in seconds while the lowest computation times are reported in bold.

PERTURBATION 1

The computational results presented in Tables 3.4-3.7 are given in terms of average computation times registered as the number of arcs affected by the cost perturbation varies. In particular, we observed that, independently from the graph dimensions, topology and density, and from the number of arcs affected by the cost change, the resolution from scratch is always the best performing approach. In fact, the average `time-ratio` (Equation (2.5)) is ≈ 5.52 for `RAND` instances, ≈ 8.38 for `RAND-GRID` instances, and ≈ 5.44 for `DIST`. However, a complete presentation of the computational results is given in Fugaro [87].

Table 3.4 (`RAND-GRID - PERTURBATION 1`). Computational results. `Reopt` and `Dijkstra` computation times in last two columns.

Instance	`Reopt`			`Dijkstra`
	Dual	**Primal**	**Total**	
heights-AR	0.204 ± 11.4%	0.811 ± 61.2%	1.016 ± 49.9%	**0.131 ± 5.4%**
heights-NC-AR	0.094 ± 8.1%	0.397 ± 81.5%	0.491 ± 67.1%	**0.065 ± 6.6%**
AR	0.208 ± 11.8%	0.827 ± 59.4%	1.036 ± 48.5%	**0.128 ± 5.1%**
NC-AR	0.095 ± 5.2%	0.421 ± 76.0%	0.516 ± 62.8%	**0.062 ± 4.2%**
heights-BR	0.223 ± 23.2%	0.830 ± 64.1%	1.053 ± 54.2%	**0.157 ± 7.3%**
heights-NC-BR	0.099 ± 9.9%	0.649 ± 77.1%	0.748 ± 67.9%	**0.084 ± 9.8%**
BR	0.244 ± 39.0%	0.960 ± 81.2%	1.204 ± 71.9%	**0.156 ± 4.3%**
NC-BR	0.101 ± 16.0%	0.627 ± 73.0%	0.728 ± 64.9%	**0.084 ± 4.4%**
heights-BS	0.213 ± 11.4%	0.750 ± 54.8%	0.963 ± 44.4%	**0.163 ± 13.1%**
heights-NC-BS	0.101 ± 13.8%	0.657 ± 63.7%	0.758 ± 55.5%	**0.074 ± 4.4%**
BS	0.221 ± 32.1%	0.811 ± 60.0%	1.032 ± 52.4%	**0.157 ± 13.0%**
NC-BS	0.096 ± 9.0%	0.620 ± 60.3%	0.716 ± 52.4%	**0.077 ± 6.2%**
heights-FS	0.193 ± 18.0%	0.747 ± 60.4%	0.940 ± 51.3%	**0.135 ± 10.1%**
heights-NC-FS	0.093 ± 6.3%	0.840 ± 60.7%	0.933 ± 54.7%	**0.077 ± 6.3%**
FS	0.192 ± 13.9%	0.816 ± 61.2%	1.008 ± 51.6%	**0.133 ± 10.2%**
NC-FS	0.091 ± 6.8%	0.795 ± 59.0%	0.885 ± 52.4%	**0.078 ± 14.0%**
heights-GP	0.075 ± 18.1%	0.561 ± 63.4%	0.636 ± 57.2%	**0.085 ± 25.8%**
heights-NC-GP	0.044 ± 11.8%	0.313 ± 73.4%	0.357 ± 65.0%	**0.043 ± 18.5%**
GP	0.076 ± 19.3%	0.521 ± 64.2%	0.597 ± 58.0%	**0.075 ± 12.7%**
NC-GP	0.040 ± 6.4%	0.307 ± 76.0%	0.348 ± 67.9%	**0.047 ± 24.2%**
heights-PI	0.066 ± 8.8%	0.443 ± 53.1%	0.509 ± 45.5%	**0.059 ± 1.8%**
heights-NC-PI	0.032 ± 7.9%	0.258 ± 68.9%	0.290 ± 61.9%	**0.032 ± 14.0%**
PI	0.060 ± 12.5%	0.445 ± 50.2%	0.505 ± 43.8%	**0.060 ± 3.6%**
NC-PI	0.034 ± 13.3%	0.278 ± 62.7%	0.312 ± 56.6%	**0.032 ± 7.0%**

As a final remark, we point out that, in general the reoptimization approach has good performance on random graphs, as observed in Section 2.4.4; however, though a certain randomness affects the considered graphs, their layered structure makes the resolution from scratch much more efficient.

Table 3.5 (`RAND-PERTURBATION 1`). Computational results. `Reopt` and `Dijkstra` computation times in last two columns.

Instance	`Reopt`			`Dijkstra`
	Dual	**Primal**	**Total**	
rand-heights-AR	2.400 ± 7.2%	0.535 ± 64.9%	2.935 ± 11.8%	**0.515 ± 6.2%**
rand-heights-NC-AR	2.158 ± 8.8%	0.534 ± 74.7%	2.692 ± 12.3%	**0.454 ± 3.9%**
rand-AR	0.814 ± 3.8%	0.340 ± 62.3%	1.154 ± 16.4%	**0.325 ± 1.6%**
rand-NC-AR	0.789 ± 8.6%	0.416 ± 76.9%	1.205 ± 24.7%	**0.332 ± 1.9%**
rand-heights-BR	3.318 ± 9.1%	0.590 ± 53.7%	3.909 ± 14.2%	**0.476 ± 3.6%**
rand-heights-NC-BR	3.521 ± 6.5%	0.501 ± 75.7%	4.022 ± 9.3%	**0.474 ± 3.6%**
rand-BR	1.486 ± 9.7%	0.425 ± 80.2%	1.910 ± 25.2%	**0.318 ± 2.4%**
rand-NC-BR	1.242 ± 4.1%	0.384 ± 60.4%	1.626 ± 15.3%	**0.336 ± 3.4%**
rand-heights-BS	1.898 ± 5.4%	0.553 ± 75.6%	2.451 ± 14.4%	**0.444 ± 5.2%**
rand-heights-NC-BS	4.130 ± 8.7%	0.729 ± 105.2%	4.858 ± 17.8%	**0.458 ± 13.9%**
rand-BS	1.123 ± 5.7%	0.353 ± 79.4%	1.476 ± 18.0%	**0.340 ± 5.0%**
rand-NC-BS	1.296 ± 6.5%	0.327 ± 72.7%	1.623 ± 13.2%	**0.289 ± 1.8%**
rand-heights-FS	3.294 ± 5.9%	0.537 ± 83.0%	3.832 ± 10.2%	**0.517 ± 4.5%**
rand-heights-NC-FS	3.097 ± 6.7%	0.602 ± 88.3%	3.699 ± 12.4%	**0.432 ± 3.3%**
rand-FS	1.262 ± 2.8%	0.382 ± 77.4%	1.644 ± 16.4%	**0.321 ± 2.5%**
rand-NC-FS	1.311 ± 3.2%	0.360 ± 67.0%	1.672 ± 16.0%	**0.335 ± 2.8%**
rand-heights-GP	1.918 ± 9.4%	0.386 ± 74.7%	2.304 ± 10.4%	**0.540 ± 5.1%**
rand-heights-NC-GP	2.612 ± 7.5%	0.401 ± 61.5%	3.013 ± 5.9%	**0.537 ± 5.2%**
rand-GP	0.988 ± 3.4%	0.288 ± 74.0%	1.276 ± 17.3%	**0.380 ± 2.2%**
rand-NC-GP	1.292 ± 2.8%	0.251 ± 66.4%	1.543 ± 10.0%	**0.368 ± 4.0%**
rand-heights-PI	1.533 ± 1.5%	0.922 ± 159.9%	2.455 ± 59.3%	**0.522 ± 3.5%**
rand-heights-NC-PI	2.021 ± 4.4%	0.591 ± 122.2%	2.612 ± 26.6%	**0.503 ± 2.9%**
rand-PI	0.819 ± 2.2%	0.302 ± 96.5%	1.121 ± 26.2%	**0.366 ± 5.0%**
rand-NC-PI	0.800 ± 2.2%	0.396 ± 114.0%	1.196 ± 37.8%	**0.329 ± 7.7%**

Table 3.6 (`DIST-PERTURBATION 1`). Computational results on `4-dist` and `8-dist` (Part 1) instances.

Instance	`Reopt`			`Dijkstra`
	Dual	**Primal**	**Total**	
4-dist-heights-AR	0.952 ± 9.6%	0.588 ± 81.3%	1.540 ± 28.4%	**0.455 ± 10.2%**
4-dist-heights-NC-AR	4.738 ± 28.0%	0.734 ± 60.1%	5.472 ± 22.0%	**0.649 ± 16.4%**
4-dist-AR	1.529 ± 42.2%	0.934 ± 111.7%	2.463 ± 53.2%	**0.454 ± 30.4%**
4-dist-NC-AR	3.610 ± 26.6%	0.995 ± 74.8%	4.606 ± 27.6%	**0.657 ± 13.2%**
4-dist-heights-BR	0.872 ± 3.1%	0.560 ± 79.6%	1.432 ± 33.0%	**0.463 ± 6.0%**
4-dist-heights-NC-BR	3.428 ± 7.6%	0.671 ± 75.6%	4.099 ± 18.1%	**0.429 ± 11.8%**
4-dist-BR	1.228 ± 14.7%	0.432 ± 62.7%	1.660 ± 22.7%	**0.569 ± 7.4%**
4-dist-NC-BR	3.945 ± 14.5%	0.697 ± 65.3%	4.641 ± 21.7%	**0.438 ± 3.5%**
4-dist-heights-BS	1.170 ± 16.0%	1.090 ± 138.1%	2.260 ± 62.4%	**0.411 ± 12.0%**
4-dist-heights-NC-BS	2.539 ± 21.9%	0.733 ± 54.5%	3.272 ± 9.5%	**0.769 ± 13.6%**
4-dist-BS	1.493 ± 27.6%	0.408 ± 53.4%	1.901 ± 11.4%	**0.262 ± 22.4%**
4-dist-NC-BS	4.166 ± 32.7%	0.682 ± 75.6%	4.848 ± 23.2%	**0.429 ± 12.8%**
4-dist-heights-FS	1.008 ± 3.4%	0.423 ± 67.8%	1.431 ± 20.5%	**0.374 ± 1.3%**
4-dist-heights-NC-FS	1.663 ± 2.5%	0.575 ± 82.0%	2.238 ± 22.0%	**0.590 ± 4.3%**
4-dist-FS	0.916 ± 3.6%	0.421 ± 66.4%	1.337 ± 21.9%	**0.388 ± 2.6%**
4-dist-NC-FS	1.921 ± 5.1%	0.628 ± 62.8%	2.549 ± 19.0%	**0.573 ± 2.7%**
4-dist-heights-GP	0.632 ± 9.4%	0.329 ± 66.3%	0.962 ± 26.2%	**0.365 ± 4.7%**
4-dist-heights-NC-GP	1.063 ± 3.0%	0.463 ± 75.9%	1.526 ± 24.4%	**0.654 ± 3.9%**
4-dist-GP	0.910 ± 19.1%	0.442 ± 81.3%	1.351 ± 38.2%	**0.515 ± 23.6%**
4-dist-NC-GP	1.555 ± 16.7%	0.515 ± 86.4%	2.069 ± 33.6%	**0.667 ± 4.1%**
4-dist-heights-PI	1.595 ± 20.5%	0.792 ± 137.8%	2.387 ± 59.1%	**0.670 ± 17.6%**
4-dist-heights-NC-PI	2.026 ± 31.7%	0.628 ± 146.4%	2.654 ± 32.7%	**0.599 ± 13.3%**
4-dist-PI	0.735 ± 9.9%	0.486 ± 116.6%	1.221 ± 51.1%	**0.506 ± 21.9%**
4-dist-NC-PI	0.777 ± 19.6%	1.078 ± 170.7%	1.856 ± 100.9%	**0.501 ± 21.4%**
8-rand-heights-AR	2.701 ± 6.0%	0.564 ± 66.2%	3.265 ± 15.1%	**1.096 ± 5.2%**
8-rand-heights-NC-AR	9.066 ± 24.3%	1.047 ± 70.4%	10.113 ± 28.3%	**1.531 ± 2.7%**
8-rand-AR	2.844 ± 10.3%	0.794 ± 107.1%	3.638 ± 29.4%	**0.601 ± 11.5%**
8-rand-NC-AR	8.652 ± 15.3%	0.969 ± 77.7%	9.621 ± 17.7%	**1.385 ± 8.1%**
8-rand-heights-BR	2.853 ± 5.3%	0.576 ± 66.5%	3.429 ± 10.3%	**0.650 ± 5.5%**
8-rand-heights-NC-BR	8.261 ± 13.4%	0.934 ± 69.8%	9.195 ± 5.9%	**0.927 ± 7.4%**
8-rand-BR	3.774 ± 15.4%	1.976 ± 176.6%	5.750 ± 56.1%	**0.983 ± 15.7%**
8-rand-NC-BR	9.282 ± 12.4%	1.840 ± 144.3%	11.122 ± 23.0%	**1.227 ± 12.9%**
8-rand-heights-BS	2.825 ± 5.1%	0.979 ± 84.0%	3.804 ± 21.4%	**0.795 ± 18.7%**
8-rand-heights-NC-BS	5.061 ± 4.7%	0.843 ± 70.2%	5.903 ± 7.8%	**0.979 ± 3.8%**
8-rand-BS	3.011 ± 3.3%	2.520 ± 127.1%	5.532 ± 56.9%	**0.889 ± 22.1%**
8-rand-NC-BS	7.134 ± 7.3%	1.046 ± 93.3%	8.180 ± 7.7%	**1.065 ± 14.3%**

Table 3.7 (`DIST-PERTURBATION 1`). Computational results on `8-dist` (Part 2) and `20-dist` instances.

Instance	Reopt			Dijkstra
	Dual	Primal	Total	
8-rand-heights-FS	2.599 ± 2.6%	0.544 ± 70.5%	3.143 ± 14.1%	**1.147 ± 2.3%**
8-rand-heights-NC-FS	4.983 ± 3.3%	1.269 ± 106.9%	6.252 ± 24.2%	**1.325 ± 6.7%**
8-rand-FS	2.326 ± 4.9%	0.560 ± 72.8%	2.886 ± 11.1%	**1.064 ± 1.4%**
8-rand-NC-FS	4.702 ± 5.5%	2.167 ± 160.4%	6.869 ± 54.4%	**1.239 ± 3.7%**
8-rand-heights-GP	1.663 ± 4.5%	0.444 ± 67.8%	2.107 ± 16.2%	**0.744 ± 3.3%**
8-rand-heights-NC-GP	1.672 ± 2.4%	0.560 ± 64.1%	2.232 ± 15.7%	**1.374 ± 3.8%**
8-rand-GP	1.541 ± 2.8%	0.459 ± 68.6%	2.000 ± 15.0%	**0.816 ± 3.2%**
8-rand-NC-GP	3.982 ± 2.8%	0.661 ± 70.4%	4.643 ± 9.8%	**1.368 ± 3.0%**
8-rand-heights-PI	2.249 ± 3.5%	0.590 ± 71.1%	2.839 ± 13.5%	**1.500 ± 4.3%**
8-rand-heights-NC-PI	2.251 ± 4.5%	0.537 ± 60.1%	2.788 ± 13.3%	**1.716 ± 1.0%**
8-rand-PI	1.308 ± 3.9%	0.348 ± 65.5%	1.655 ± 14.0%	**0.832 ± 3.3%**
8-rand-NC-PI	1.654 ± 1.7%	0.871 ± 148.1%	2.525 ± 52.1%	**0.902 ± 23.0%**
20-rand-heights-AR	4.987 ± 4.1%	2.532 ± 72.5%	7.520 ± 23.0%	**1.482 ± 23.2%**
20-rand-heights-NC-AR	17.278 ± 13.3%	1.600 ± 64.1%	18.878 ± 14.3%	**2.926 ± 3.4%**
20-rand-AR	5.393 ± 3.6%	0.840 ± 54.0%	6.234 ± 5.2%	**0.986 ± 4.5%**
20-rand-NC-AR	22.341 ± 9.7%	1.337 ± 59.7%	23.678 ± 7.3%	**2.222 ± 3.5%**
20-rand-heights-BR	3.779 ± 19.8%	10.946 ± 206.9%	14.725 ± 152.9%	**1.915 ± 7.1%**
20-rand-heights-NC-BR	18.264 ± 12.0%	1.584 ± 79.0%	19.848 ± 5.8%	**2.580 ± 14.5%**
20-rand-BR	6.916 ± 13.5%	0.873 ± 53.1%	7.789 ± 7.3%	**1.296 ± 9.3%**
20-rand-NC-BR	21.831 ± 17.1%	2.552 ± 132.2%	24.383 ± 16.3%	**2.496 ± 27.0%**
20-rand-heights-BS	5.626 ± 6.6%	16.151 ± 189.9%	21.777 ± 140.2%	**1.194 ± 17.1%**
20-rand-heights-NC-BS	13.285 ± 6.3%	1.181 ± 73.9%	14.466 ± 10.9%	**1.731 ± 3.7%**
20-rand-BS	5.849 ± 4.3%	5.271 ± 126.4%	11.120 ± 60.8%	**1.154 ± 3.0%**
20-rand-NC-BS	21.805 ± 5.5%	3.644 ± 163.2%	25.448 ± 21.8%	**2.392 ± 2.7%**
20-rand-heights-FS	4.066 ± 2.5%	0.870 ± 61.2%	4.935 ± 11.4%	**1.825 ± 2.4%**
20-rand-heights-NC-FS	16.388 ± 4.5%	1.171 ± 58.7%	17.559 ± 7.4%	**2.015 ± 5.0%**
20-rand-FS	4.912 ± 3.0%	2.154 ± 127.1%	7.065 ± 38.6%	**1.874 ± 3.5%**
20-rand-NC-FS	14.469 ± 4.8%	1.229 ± 60.2%	15.698 ± 6.7%	**1.985 ± 3.3%**
20-rand-heights-GP	2.951 ± 6.8%	0.777 ± 62.5%	3.728 ± 18.3%	**1.655 ± 2.7%**
20-rand-heights-NC-GP	5.750 ± 1.7%	0.917 ± 59.3%	6.667 ± 9.1%	**2.271 ± 0.5%**
20-rand-GP	3.369 ± 3.0%	1.284 ± 110.0%	4.652 ± 31.1%	**0.986 ± 3.9%**
20-rand-NC-GP	9.630 ± 8.0%	1.095 ± 66.3%	10.725 ± 12.7%	**2.123 ± 3.6%**
20-rand-heights-PI	7.238 ± 35.9%	0.948 ± 55.3%	8.186 ± 34.5%	**1.447 ± 7.3%**
20-rand-heights-NC-PI	6.521 ± 25.5%	5.060 ± 192.5%	11.581 ± 87.1%	**2.237 ± 7.0%**
20-rand-PI	1.913 ± 7.6%	2.320 ± 170.0%	4.234 ± 94.9%	**1.545 ± 6.7%**
20-rand-NC-PI	2.977 ± 21.7%	0.720 ± 67.6%	3.697 ± 30.3%	**1.532 ± 10.1%**

PERTURBATION 2

As explained in Section 3.3.2, this perturbation was applied by simulating the effects of the wind blowing in two different directions. Thus, Tables 3.8 - 3.10 report the results of the computational experiments relative to the wind from East to West; instead, the data in Tables 3.11 - 3.13 refer to the wind from West to East. These Tables are presented in Appendix 3.B.

Analyzing the computational times of the algorithms, it is noteworthy that, independently from the direction of the wind:

- on `4-ol` problems, `Reopt` is competitive with `Dijkstra` until $r \leq 17$, i.e. when at most 1000 nodes are affected by the perturbation.
- on `8-ol` problems, the change in the performance trends occurs when $r \geq 12$ though `Reopt` is still advisable when $r = 27, 32, 37$ on `8-ol-PI` network. A similar trend emerges also for `20-ol` instances.

In particular, the inversion in the performance trends depends on the increase of the number of arcs with negative reduced cost, namely those arcs whose cost has been decreased. In fact, when this event occurs, a greater number of primal phases needs to be executed. This explains also the behaviour of the algorithm on `8-ol-PI` and `20-ol-PI` instances: the number of primal phases executed by `Reopt` on them is lower than that relative to the other networks [87].

Additionally, the observed trends are also influenced by the density of the network: as the density of the network increases, a greater number of arcs is affected by the cost perturbation in correspondence to the same value of r. For example, for an inner node at most: 799 cost changes occurs in `4-ol` instances, 973 in the `8-ol` ones and 1512 in the `20-ol` ones.

3.4 Trajectories

In this Section we propose a graphical representation of the trajectories obtained in Section 3.3 with the aim of determining which discretization method produces the most realistic scenarios.

It is worth to mention that the trajectories obtained from the resolution of the SPPs are however subject to a post-processing that serves to make them smoother. Therefore, the main purpose of this representation is to understand which approach allows to define reasonable routes i.e. without zig-zags or sudden changes in altitude that would make the flight uncomfortable and more expensive in terms of fuel/energy consumption.

The graphics are obtained with The MathWorks, Inc.MATLAB ® R2018b; by way of example, we report only the trajectories relative to the couple (G,P) of skyports.[2]. Specifically, Fig. 3.3-3.4 refer to `RAND-GRID` and `RAND` instances, respectively; instead, Fig. 3.5 depicts the trajectories on `4-ol-GP` instance as

[2]The whole set of figures is available at `https://figshare.com/articles/figure/TRAJECTORIES/13333799/2`

r varies, and with wind blowing from East to West. All the Figures show the 3D plan of the toy city where the red boxes denote the parallelepipeds i.e. the buildings on which the skyports are situated.

As we could expect, the most realistic trajectories are obtained when the Free-Space Map is given by the Discretization via Grids (Section 3.2.1). In particular, when both the connections in the graph and the cost function are characterized by randomness, the trajectories of the VTOL get odd (see Fig. 3.4). Finally, as regards `ONE-LAYER` instances, we can observe that none of the trajectories is affected by the perturbation and that in general, though the arcs are still defined at random, the oddness of the trajectories is smoothed by using the Euclidean distance function to define costs.

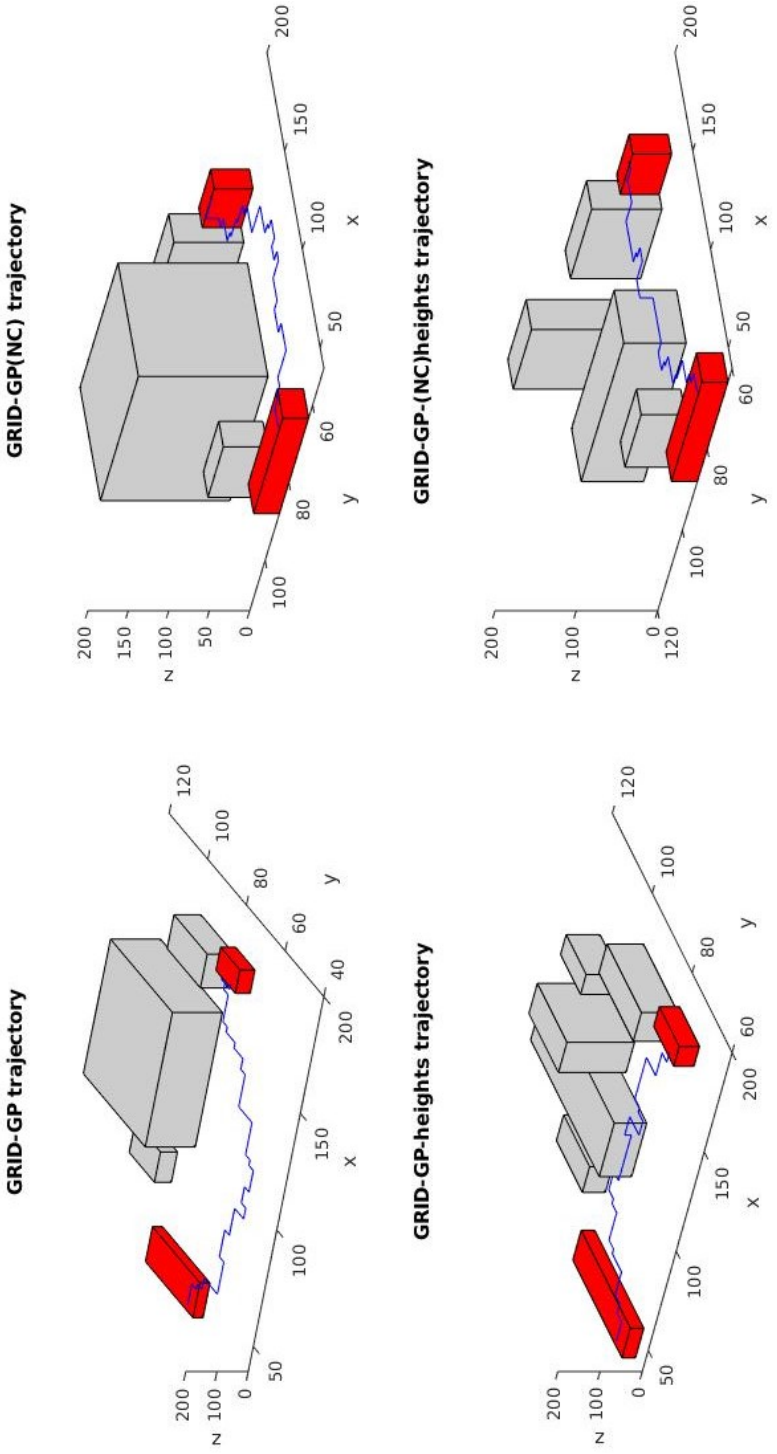


Fig. 3.3 (RAND-GRID) . Trajectories of the VTOL (blue solid line). Skyports are located on buildings G and P.

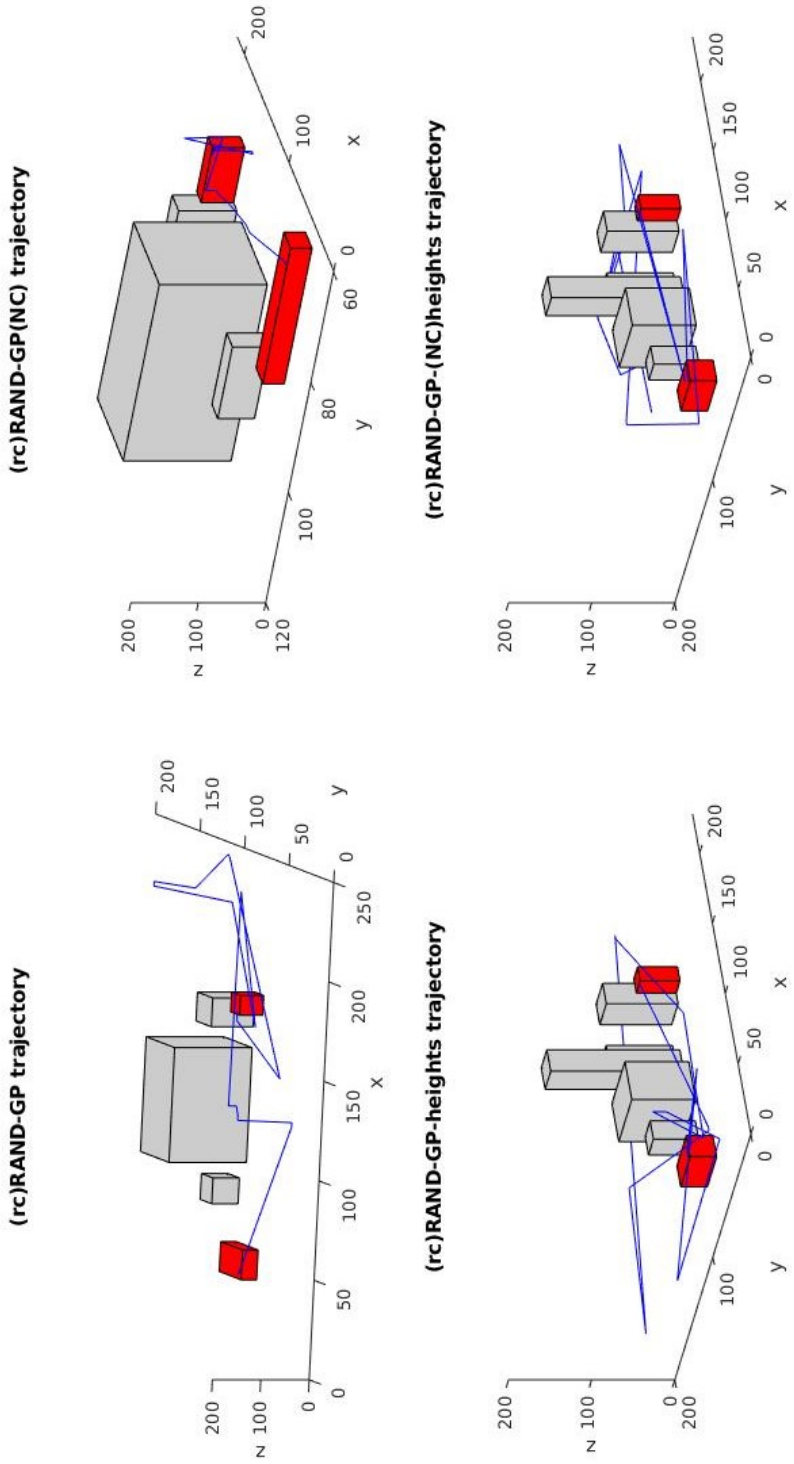


Fig. 3.4 (RAND) . Trajectories of the VTOL (blue solid line). Skyports are located on buildings G and P.

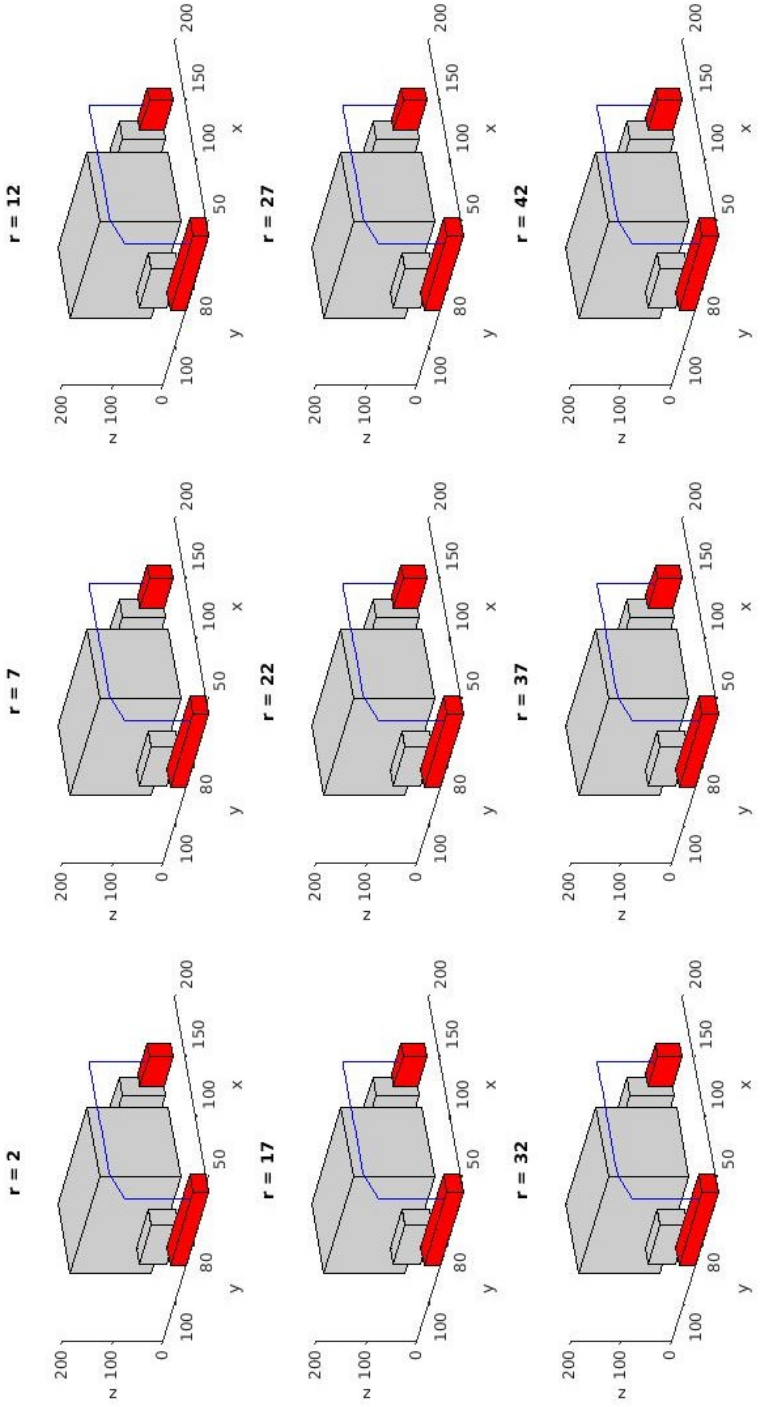


Fig. 3.5 (ONE-LAYER) . Trajectories of the VTOL (blue solid line) on 4-o1 instances. Skyports are on buildings G and P.

3.5 Conclusions

The topics addressed in this Chapter are located in the framework of Path Planning in Urban Air Mobility. Specifically, in Fugaro [87] two issues have been considered: i) the representation of the 3D space designated for the motion and ii) the computation of the trajectories.

On the one hand, three different but correlated discretization approaches are proposed, which allow to define *Free-Space Maps*; in particular, the analysis of the aircraft trajectories obtained in these different environments highlights that total randomness should be avoided. Thus, to depict more realistic scenarios, the graph representations should contain a structured topology and the definition of the cost function should account for the Euclidean distances.

On the other hand, the computation of the trajectories is addressed with the *two-steps resolution*: the shortest paths are computed firstly by considering as deterministic the initial cost information, and then updating this information along with the corresponding solution. Indeed, this scenario resembles the resolution of two slightly different shortest path problems described in Chapter 2; consequently, it could be addressed with the Reoptimization algorithm studied in Festa et al. [83]. At this purpose, a thorough experimentation is conducted in Fugaro [87] to determine whether reoptimizing is advisable or not; the obtained results underline that, except for some cases, a resolution from scratch should be preferred regardless of the percentage of the network affected by the change and the extent of the change itself.

Appendix

Appendix 3.A Intersection of Segments

The purpose of this Section is to describe a theoretical geometry result that was used in the post-processing operations needed by the One-Layer Discretization (Section 3.2.3). In fact, when this method is adopted, it is necessary to perform the feasibility check of the set of arcs on each layer to remove arcs crossing the buildings and the following property allows to find out if two segments in the plane intersect.

Given a two dimensional Cartesian coordinate system, with origin O and axis lines X and Y, we consider two segments, $\overline{12}$ and $\overline{34}$, whose endpoints have coordinates $(x_i, y_i)_{i=1,2}$ and $(x_i, y_i)_{i=3,4}$, respectively. Suppose that these two segments intersect in a point $P = (\bar{x}, \bar{y})$, and draw appropriate segments, parallel to the X axis from the endpoints $1, 3$ and from P and parallel to the Y axis from the endpoints $2, 4$ and from P, as depicted in Fig. 3.6.

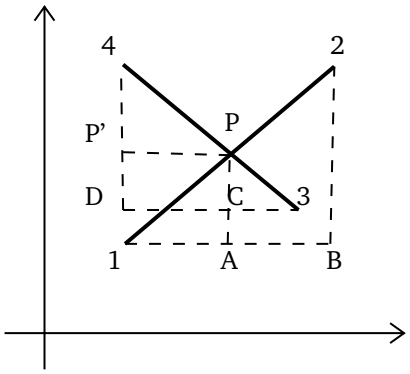


Fig. 3.6 Intersection between two segments.

Then, the rectangular triangles $1AP$ and $12B$ are similar; thus their sides are proportional according to a positive factor $k_A \leq 1$. Similarly, the rectangular triangles $3CP$ and $34D$ are similar too; thus their sides are proportional according to a positive factor $k_B \leq 1$.

Consequently, the coordinates of the intersection point P can be rewritten as Equations (3.1a)-(3.1d).

$$\bar{x} = x_1 + |\overline{1A}| = x_1 + k_A|\overline{1B}| = x_1 + k_A(x_2 - x_1), \tag{3.1a}$$

$$\bar{x} = x_3 + |\overline{3C}| = x_3 + k_B|\overline{3D}| = x_3 + k_B(x_4 - x_3), \tag{3.1b}$$

$$\bar{y} = y_1 + |\overline{AP}| = y_1 + k_A|\overline{2B}| = y_1 + k_A(y_2 - y_1), \tag{3.1c}$$

$$\bar{y} = y_3 + |\overline{CP}| = y_3 + k_B|\overline{4D}| = y_3 + k_B(y_4 - y_3). \tag{3.1d}$$

In particular, by comparing (3.1a) with (3.1c), and (3.1b) with (3.1d) we obtain the linear system (3.2).

$$\begin{cases} x_1 + k_A(x_2 - x_1) = x_3 + k_B(x_4 - x_3) \\ y_1 + k_A(y_2 - y_1) = y_3 + k_B(y_4 - y_3) \end{cases} \tag{3.2}$$

The solutions of this system allow to express the constants k_A and k_B as a function of the coordinates of the endpoints of $\overline{12}$ and $\overline{34}$, as shown in (3.3).

$$\begin{aligned} k_A &= \frac{(y_1 - y_3)(x_4 - x_3) - (y_4 - y_3)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}, \\ k_B &= \frac{(y_1 - y_3)(x_2 - x_1) - (y_2 - y_1)(x_1 - x_3)}{(y_4 - y_3)(x_2 - x_1) - (x_4 - x_3)(y_2 - y_1)}. \end{aligned} \tag{3.3}$$

Proposition 3.A.1. *In a two dimensional Cartesian coordinate system, consider the segments* $\overline{12}$ *and* $\overline{34}$*, whose endpoints have coordinates* $(x_i, y_i)_{i=1,2}$ *and* $(x_i, y_i)_{i=3,4}$*, respectively. Then, these segments do not intersect if the denominators in Equation* (3.3) *is null. Otherwise, there exists an intersection point if and only if both the numerators in Equation* (3.3) *belong to* $[0, 1]$.

Appendix 3.B Results on the ONE-LAYER data-set

In this Section we report the data relative to the experiments conducted on the ONE-LAYER data-set (see Section 3.3). Specifically, Tables 3.8-3.10 refer to experimentation conducted simulating an East to West wind, while Tables 3.11-3.13 to that relative to a West to East wind. All the computational times are given in seconds while the lower times are reported in bold.

Table 3.8 (ONE-LAYER - PERTURBATION 2). Computational results on 4-ol instances. Wind from East to West.

r	Instance	Reopt			Dijkstra	r	Instance	Reopt			Dijkstra
		Dual	Primal	Total				Dual	Primal	Total	
2	4-ol-AR	0.047	0.000	**0.047**	0.086	22	4-ol-FS	0.048	0.218	0.266	**0.092**
2	4-ol-BR	0.044	0.000	**0.044**	0.091	22	4-ol-GP	0.046	0.266	0.312	**0.070**
2	4-ol-BS	0.053	0.000	**0.053**	0.078	22	4-ol-PI	0.048	0.364	0.412	**0.071**
2	4-ol-FS	0.051	0.000	**0.051**	0.091	27	4-ol-AR	0.052	0.363	0.415	**0.098**
2	4-ol-GP	0.049	0.000	**0.049**	0.061	27	4-ol-BR	0.055	0.471	0.525	**0.116**
2	4-ol-PI	0.054	0.000	**0.054**	0.073	27	4-ol-BS	0.057	0.478	0.535	**0.084**
7	4-ol-AR	0.042	0.007	**0.048**	0.132	27	4-ol-FS	0.061	0.353	0.414	**0.120**
7	4-ol-BR	0.045	0.003	**0.048**	0.112	27	4-ol-GP	0.052	0.393	0.445	**0.070**
7	4-ol-BS	0.047	0.007	**0.054**	0.101	27	4-ol-PI	0.063	0.279	0.342	**0.060**
7	4-ol-FS	0.047	0.005	**0.051**	0.091	32	4-ol-AR	0.052	0.469	0.521	**0.121**
7	4-ol-GP	0.043	0.007	**0.050**	0.091	32	4-ol-BR	0.055	0.682	0.737	**0.108**
7	4-ol-PI	0.063	0.010	**0.073**	0.081	32	4-ol-BS	0.047	0.542	0.589	**0.094**
12	4-ol-AR	0.041	0.016	**0.058**	0.094	32	4-ol-FS	0.055	0.475	0.529	**0.125**
12	4-ol-BR	0.041	0.024	**0.066**	0.113	32	4-ol-GP	0.056	0.461	0.517	**0.071**
12	4-ol-BS	0.047	0.069	0.115	**0.088**	32	4-ol-PI	0.067	0.337	0.404	**0.061**
12	4-ol-FS	0.047	0.039	**0.086**	0.092	37	4-ol-AR	0.053	0.341	0.395	**0.130**
12	4-ol-GP	0.042	0.045	0.087	**0.069**	37	4-ol-BR	0.052	0.226	0.278	**0.151**
12	4-ol-PI	0.053	0.053	0.107	**0.073**	37	4-ol-BS	0.055	0.410	0.465	**0.118**
17	4-ol-AR	0.042	0.081	0.123	**0.097**	37	4-ol-FS	0.059	0.509	0.568	**0.129**
17	4-ol-BR	0.049	0.113	0.162	**0.101**	37	4-ol-GP	0.060	0.386	0.447	**0.088**
17	4-ol-BS	0.044	0.135	0.179	**0.085**	37	4-ol-PI	0.081	0.439	0.521	**0.057**
17	4-ol-FS	0.052	0.089	0.141	**0.120**	42	4-ol-AR	0.062	0.550	0.612	**0.158**
17	4-ol-GP	0.046	0.120	0.165	**0.066**	42	4-ol-BR	0.077	0.398	0.475	**0.188**
17	4-ol-PI	0.048	0.112	0.160	**0.071**	42	4-ol-BS	0.075	0.461	0.536	**0.131**
22	4-ol-AR	0.044	0.251	0.295	**0.100**	42	4-ol-FS	0.052	0.547	0.598	**0.165**
22	4-ol-BR	0.042	0.265	0.306	**0.110**	42	4-ol-GP	0.055	0.404	0.459	**0.096**
22	4-ol-BS	0.047	0.279	0.326	**0.096**	42	4-ol-PI	0.076	0.205	0.280	**0.054**

Table 3.9 (ONE-LAYER - PERTURBATION 2). Computational results on 8-ol instances. Wind from East to West.

r	Instance	Reopt			Dijkstra	r	Instance	Reopt			Dijkstra
		Dual	Primal	Total				Dual	Primal	Total	
2	8-ol-AR	0.095	0.000	**0.095**	0.133	22	8-ol-FS	0.094	0.270	0.364	**0.122**
2	8-ol-BR	0.099	0.000	**0.099**	0.147	22	8-ol-GP	0.110	0.359	0.468	**0.106**
2	8-ol-BS	0.107	0.000	**0.107**	0.137	22	8-ol-PI	0.116	0.501	0.616	**0.202**
2	8-ol-FS	0.087	0.000	**0.087**	0.124	27	8-ol-AR	0.108	0.852	0.960	**0.142**
2	8-ol-GP	0.085	0.000	**0.085**	0.106	27	8-ol-BR	0.106	1.087	1.193	**0.167**
2	8-ol-PI	0.118	0.000	**0.118**	0.191	27	8-ol-BS	0.106	0.941	1.047	**0.134**
7	8-ol-AR	0.094	0.008	**0.102**	0.128	27	8-ol-F	0.113	0.512	0.624	**0.124**
7	8-ol-BR	0.098	0.017	**0.115**	0.140	27	8-ol-GP	0.117	0.592	0.709	**0.102**
7	8-ol-BS	0.099	0.017	**0.115**	0.142	27	8-ol-PI	0.117	0.085	**0.202**	0.343
7	8-ol-FS	0.087	0.000	**0.087**	0.131	32	8-ol-A	0.106	0.894	0.999	**0.130**
7	8-ol-GP	0.090	0.019	**0.109**	0.111	32	8-ol-BR	0.103	1.252	1.355	**0.144**
7	8-ol-PI	0.113	0.032	**0.144**	0.185	32	8-ol-BS	0.103	1.264	1.367	**0.133**
12	8-ol-AR	0.094	0.049	0.143	**0.124**	32	8-ol-FS	0.122	0.692	0.814	**0.123**
12	8-ol-BR	0.104	0.073	0.177	**0.138**	32	8-ol-GP	0.117	0.671	0.788	**0.097**
12	8-ol-BS	0.095	0.071	0.166	**0.148**	32	8-ol-PI	0.129	0.145	**0.274**	0.382
12	8-ol-FS	0.089	0.035	**0.124**	0.138	37	8-ol-AR	0.107	0.566	0.673	**0.126**
12	8-ol-GP	0.097	0.104	0.201	**0.112**	37	8-ol-BR	0.108	0.259	0.367	**0.136**
12	8-ol-PI	0.122	0.168	0.290	**0.187**	37	8-ol-BS	0.106	0.252	0.359	**0.125**
17	8-ol-AR	0.099	0.184	0.282	**0.119**	37	8-ol-FS	0.128	0.906	1.033	**0.133**
17	8-ol-BR	0.097	0.249	0.347	**0.144**	37	8-ol-GP	0.111	0.341	0.453	**0.095**
17	8-ol-BS	0.097	0.285	0.382	**0.147**	37	8-ol-PI	0.127	0.213	**0.341**	0.383
17	8-ol-FS	0.102	0.129	0.231	**0.128**	42	8-ol-A	0.103	0.486	0.590	**0.144**
17	8-ol-GP	0.108	0.273	0.382	**0.100**	42	8-ol-BR	0.109	0.374	0.484	**0.129**
17	8-ol-PI	0.108	0.340	0.448	**0.183**	42	8-ol-BS	0.115	0.362	0.477	**0.142**
22	8-ol-AR	0.091	0.545	0.637	**0.149**	42	8-ol-FS	0.123	0.532	0.655	**0.144**
22	8-ol-BR	0.104	0.621	0.724	**0.176**	42	8-ol-GP	0.125	0.461	0.585	**0.102**
22	8-ol-BS	0.094	0.573	0.667	**0.146**	42	8-ol-PI	0.141	0.310	0.451	**0.383**

Table 3.10 `(ONE-LAYER - PERTURBATION 2).` Computational results on `20-ol` instances. Wind from East to West.

r	Instance	Reopt			Dijkstra	r	Instance	Reopt			Dijkstra
		Dual	Primal	Total				Dual	Primal	Total	
2	20-ol-AR	0.155	0.000	**0.155**	0.194	22	20-ol-FS	0.161	0.535	0.696	**0.178**
2	20-ol-BR	0.190	0.000	**0.190**	0.211	22	20-ol-GP	0.199	0.999	1.199	**0.502**
2	20-ol-BS	0.153	0.000	**0.153**	0.200	22	20-ol-PI	0.272	0.644	**0.916**	1.269
2	20-ol-FS	0.154	0.000	**0.154**	0.184	27	20-ol-AR	0.190	1.894	2.084	**0.184**
2	20-ol-GP	0.145	0.000	**0.145**	0.525	27	20-ol-BR	0.228	2.717	2.944	**0.194**
2	20-ol-PI	0.226	0.000	**0.226**	0.930	27	20-ol-BS	0.181	2.352	2.533	**0.191**
7	20-ol-AR	0.157	0.055	0.212	**0.185**	27	20-ol-FS	0.192	1.232	1.424	**0.171**
7	20-ol-BR	0.194	0.080	0.274	**0.186**	27	20-ol-GP	0.245	1.423	1.668	**0.504**
7	20-ol-BS	0.147	0.055	0.202	**0.199**	27	20-ol-PI	0.251	0.194	**0.445**	1.173
7	20-ol-FS	0.138	0.000	**0.138**	0.191	32	20-ol-AR	0.208	2.297	2.504	**0.169**
7	20-ol-GP	0.142	0.000	**0.142**	0.529	32	20-ol-BR	0.235	2.777	3.012	**0.195**
7	20-ol-PI	0.240	0.069	**0.309**	0.944	32	20-ol-BS	0.197	2.124	2.322	**0.193**
12	20-ol-AR	0.159	0.188	0.347	**0.185**	32	20-ol-FS	0.220	1.495	1.715	**0.167**
12	20-ol-BR	0.189	0.316	0.504	**0.197**	32	20-ol-GP	0.280	1.210	1.490	**0.607**
12	20-ol-BS	0.151	0.294	0.444	**0.197**	32	20-ol-PI	0.247	0.394	**0.642**	1.095
12	20-ol-FS	0.155	0.019	**0.174**	0.192	37	20-ol-AR	0.252	1.855	2.107	**0.166**
12	20-ol-GP	0.163	0.090	**0.253**	0.521	37	20-ol-BR	0.242	1.468	1.711	**0.185**
12	20-ol-PI	0.220	0.485	**0.705**	0.993	37	20-ol-BS	0.249	1.391	1.639	**0.202**
17	20-ol-AR	0.161	0.607	0.768	**0.184**	37	20-ol-FS	0.248	1.923	2.171	**0.160**
17	20-ol-BR	0.194	0.983	1.177	**0.197**	37	20-ol-GP	0.286	0.628	0.913	**0.599**
17	20-ol-BS	0.144	0.783	0.927	**0.197**	37	20-ol-PI	0.283	0.567	**0.850**	0.969
17	20-ol-FS	0.148	0.168	0.316	**0.177**	42	20-ol-AR	0.250	1.538	1.788	**0.159**
17	20-ol-GP	0.167	0.397	0.565	**0.501**	42	20-ol-BR	0.244	1.449	1.693	**0.179**
17	20-ol-PI	0.235	0.918	1.153	**1.102**	42	20-ol-BS	0.254	1.254	1.508	**0.208**
22	20-ol-AR	0.172	1.316	1.488	**0.172**	42	20-ol-FS	0.274	1.773	2.047	**0.179**
22	20-ol-BR	0.209	1.396	1.605	**0.192**	42	20-ol-GP	0.274	0.708	0.982	**0.585**
22	20-ol-BS	0.159	1.644	1.803	**0.194**	42	20-ol-PI	0.307	0.773	1.081	**0.978**

Table 3.11 (ONE-LAYER - PERTURBATION 2). Computational results on 4-ol instances. Wind from West to East.

r	Instance	Reopt			Dijkstra	r	Instance	Reopt			Dijkstra
		Dual	Primal	Total				Dual	Primal	Total	
2	4-ol-AR	0.052	0.000	**0.052**	0.088	22	4-ol-FS	0.046	0.218	0.265	**0.095**
2	4-ol-BR	0.048	0.000	**0.048**	0.097	22	4-ol-GP	0.046	0.237	0.283	**0.064**
2	4-ol-BS	0.049	0.000	**0.049**	0.080	22	4-ol-PI	0.048	0.298	0.345	**0.068**
2	4-ol-FS	0.048	0.000	**0.048**	0.087	27	4-ol-AR	0.051	0.332	0.382	**0.094**
2	4-ol-GP	0.049	0.000	**0.049**	0.061	27	4-ol-BR	0.051	0.440	0.491	**0.109**
2	4-ol-PI	0.060	0.000	**0.060**	0.065	27	4-ol-BS	0.043	0.403	0.446	**0.080**
7	4-ol-AR	0.041	0.006	**0.047**	0.095	27	4-ol-FS	0.052	0.327	0.379	**0.110**
7	4-ol-BR	0.040	0.002	**0.042**	0.103	27	4-ol-GP	0.048	0.313	0.360	**0.070**
7	4-ol-BS	0.050	0.007	**0.058**	0.087	27	4-ol-PI	0.056	0.241	0.297	**0.059**
7	4-ol-FS	0.041	0.005	**0.046**	0.089	32	4-ol-AR	0.053	0.409	0.463	**0.114**
7	4-ol-GP	0.042	0.007	**0.048**	0.066	32	4-ol-BR	0.049	0.600	0.649	**0.099**
7	4-ol-PI	0.055	0.009	**0.064**	0.064	32	4-ol-BS	0.047	0.503	0.550	**0.093**
12	4-ol-AR	0.042	0.019	**0.061**	0.096	32	4-ol-FS	0.054	0.433	0.487	**0.121**
12	4-ol-BR	0.042	0.021	**0.063**	0.103	32	4-ol-GP	0.053	0.352	0.406	**0.072**
12	4-ol-BS	0.052	0.056	0.108	**0.089**	32	4-ol-PI	0.069	0.276	0.344	**0.059**
12	4-ol-FS	0.042	0.033	**0.075**	0.089	37	4-ol-AR	0.051	0.331	0.382	**0.130**
12	4-ol-GP	0.043	0.045	0.087	**0.068**	37	4-ol-BR	0.052	0.194	0.245	**0.138**
12	4-ol-PI	0.049	0.057	0.106	**0.071**	37	4-ol-BS	0.052	0.349	0.401	**0.104**
17	4-ol-AR	0.042	0.082	0.124	**0.097**	37	4-ol-FS	0.057	0.481	0.538	**0.123**
17	4-ol-BR	0.041	0.118	0.159	**0.102**	37	4-ol-GP	0.059	0.375	0.434	**0.083**
17	4-ol-BS	0.044	0.123	0.167	**0.080**	37	4-ol-PI	0.071	0.330	0.401	**0.055**
17	4-ol-FS	0.045	0.094	0.139	**0.102**	42	4-ol-AR	0.056	0.424	0.480	**0.154**
17	4-ol-GP	0.045	0.105	0.149	**0.064**	42	4-ol-BR	0.057	0.259	0.316	**0.165**
17	4-ol-PI	0.048	0.114	0.162	**0.070**	42	4-ol-BS	0.056	0.412	0.469	**0.128**
22	4-ol-AR	0.044	0.215	0.259	**0.101**	42	4-ol-FS	0.052	0.501	0.553	**0.167**
22	4-ol-BR	0.050	0.254	0.303	**0.104**	42	4-ol-GP	0.054	0.393	0.447	**0.099**
22	4-ol-BS	0.047	0.258	0.306	**0.087**	42	4-ol-PI	0.075	0.185	0.259	**0.054**

Table 3.12 (ONE-LAYER - PERTURBATION 2). Computational results on 8-ol instances. Wind from West to East.

r	Instance	Reopt			Dijkstra	r	Instance	Reopt			Dijkstra
		Dual	**Primal**	**Total**				**Dual**	**Primal**	**Total**	
2	8-ol-AR	0.086	0.000	**0.086**	0.124	22	8-ol-FS	0.094	0.272	0.365	**0.115**
2	8-ol-BR	0.102	0.000	**0.102**	0.136	22	8-ol-GP	0.107	0.370	0.477	**0.102**
2	8-ol-BS	0.109	0.000	**0.109**	0.134	22	8-ol-PI	0.109	0.404	0.513	**0.199**
2	8-ol-FS	0.088	0.000	**0.088**	0.117	27	8-ol-AR	0.110	0.715	0.825	**0.127**
2	8-ol-GP	0.100	0.000	**0.100**	0.101	27	8-ol-BR	0.102	0.922	1.024	**0.135**
2	8-ol-PI	0.108	0.000	**0.108**	0.187	27	8-ol-BS	0.107	0.929	1.036	**0.136**
7	8-ol-AR	0.086	0.008	**0.094**	0.121	27	8-ol-FS	0.115	0.505	0.621	**0.120**
7	8-ol-BR	0.098	0.018	**0.116**	0.133	27	8-ol-GP	0.117	0.612	0.729	**0.104**
7	8-ol-BS	0.096	0.017	**0.113**	0.140	27	8-ol-PI	0.119	0.088	**0.207**	0.342
7	8-ol-FS	0.084	0.000	**0.084**	0.126	32	8-ol-AR	0.112	0.924	1.035	**0.132**
7	8-ol-GP	0.086	0.014	**0.100**	0.102	32	8-ol-BR	0.106	1.253	1.358	**0.145**
7	8-ol-PI	0.104	0.032	**0.135**	0.181	32	8-ol-BS	0.103	1.297	1.400	**0.135**
12	8-ol-AR	0.089	0.046	0.135	**0.122**	32	8-ol-FS	0.121	0.700	0.821	**0.120**
12	8-ol-BR	0.101	0.066	0.167	**0.136**	32	8-ol-GP	0.118	0.678	0.796	**0.098**
12	8-ol-BS	0.096	0.079	0.175	**0.139**	32	8-ol-PI	0.128	0.148	**0.276**	0.392
12	8-ol-FS	0.087	0.032	**0.119**	0.123	37	8-ol-AR	0.105	0.570	0.675	**0.127**
12	8-ol-GP	0.097	0.093	0.190	**0.104**	37	8-ol-BR	0.106	0.256	0.362	**0.136**
12	8-ol-PI	0.107	0.150	0.257	**0.188**	37	8-ol-BS	0.106	0.267	0.373	**0.128**
17	8-ol-AR	0.092	0.157	0.250	**0.120**	37	8-ol-FS	0.129	0.945	1.074	**0.133**
17	8-ol-BR	0.102	0.248	0.350	**0.143**	37	8-ol-GP	0.118	0.351	0.469	**0.097**
17	8-ol-BS	0.105	0.247	0.352	**0.133**	37	8-ol-PI	0.125	0.216	**0.341**	0.382
17	8-ol-FS	0.096	0.129	0.225	**0.125**	42	8-ol-AR	0.108	0.503	0.612	**0.147**
17	8-ol-GP	0.103	0.221	0.323	**0.098**	42	8-ol-BR	0.115	0.372	0.487	**0.136**
17	8-ol-PI	0.109	0.352	0.461	**0.184**	42	8-ol-BS	0.110	0.369	0.479	**0.143**
22	8-ol-AR	0.089	0.435	0.524	**0.118**	42	8-ol-FS	0.124	0.545	0.669	**0.142**
22	8-ol-BR	0.090	0.555	0.645	**0.149**	42	8-ol-GP	0.116	0.437	0.553	**0.102**
22	8-ol-BS	0.090	0.551	0.641	**0.143**	42	8-ol-PI	0.138	0.321	0.458	**0.378**

Table 3.13 `(ONE-LAYER - PERTURBATION 2).` Computational results on `20-ol` instances. Wind from West to East.

r	Instance	Reopt			Dijkstra	r	Instance	Reopt			Dijkstra
		Dual	Primal	Total				Dual	Primal	Total	
2	20-ol-AR	0.168	0.000	**0.168**	0.187	22	20-ol-FS	0.163	0.543	0.706	**0.179**
2	20-ol-BR	0.190	0.000	**0.190**	0.206	22	20-ol-GP	0.202	0.988	1.190	**0.504**
2	20-ol-BS	0.151	0.000	**0.151**	0.197	22	20-ol-PI	0.270	0.629	**0.900**	1.273
2	20-ol-FS	0.153	0.000	**0.153**	0.185	27	20-ol-AR	0.188	1.704	1.892	**0.168**
2	20-ol-GP	0.149	0.000	**0.149**	0.526	27	20-ol-BR	0.222	2.198	2.420	**0.189**
2	20-ol-PI	0.230	0.000	**0.230**	0.924	27	20-ol-BS	0.174	2.403	2.577	**0.190**
7	20-ol-AR	0.161	0.051	0.212	**0.185**	27	20-ol-FS	0.183	1.212	1.395	**0.167**
7	20-ol-BR	0.191	0.082	0.273	**0.193**	27	20-ol-GP	0.237	1.428	1.665	**0.500**
7	20-ol-BS	0.141	0.048	**0.189**	0.200	27	20-ol-PI	0.247	0.207	**0.454**	1.184
7	20-ol-FS	0.136	0.000	**0.136**	0.189	32	20-ol-AR	0.205	2.267	2.472	**0.167**
7	20-ol-GP	0.140	0.000	**0.140**	0.527	32	20-ol-BR	0.231	2.761	2.992	**0.195**
7	20-ol-PI	0.241	0.073	**0.314**	0.946	32	20-ol-BS	0.201	2.080	2.281	**0.199**
12	20-ol-AR	0.159	0.184	0.343	**0.185**	32	20-ol-FS	0.210	1.530	1.740	**0.160**
12	20-ol-BR	0.187	0.322	0.510	**0.198**	32	20-ol-GP	0.283	1.188	1.471	**0.614**
12	20-ol-BS	0.149	0.302	0.451	**0.201**	32	20-ol-PI	0.255	0.387	**0.643**	1.098
12	20-ol-FS	0.157	0.021	**0.178**	0.182	37	20-ol-AR	0.250	1.802	2.053	**0.166**
12	20-ol-GP	0.153	0.094	**0.247**	0.518	37	20-ol-BR	0.244	1.177	1.422	**0.178**
12	20-ol-PI	0.218	0.478	**0.696**	0.991	37	20-ol-BS	0.243	1.209	1.452	**0.202**
17	20-ol-AR	0.161	0.616	0.778	**0.179**	37	20-ol-FS	0.254	1.859	2.114	**0.166**
17	20-ol-BR	0.196	0.967	1.163	**0.199**	37	20-ol-GP	0.285	0.626	0.911	**0.607**
17	20-ol-BS	0.141	0.769	0.911	**0.193**	37	20-ol-PI	0.277	0.578	**0.855**	0.989
17	20-ol-FS	0.141	0.177	0.318	**0.176**	42	20-ol-AR	0.255	1.538	1.793	**0.160**
17	20-ol-GP	0.161	0.406	0.567	**0.499**	42	20-ol-BR	0.237	1.493	1.731	**0.178**
17	20-ol-PI	0.238	0.900	1.138	**1.105**	42	20-ol-BS	0.262	1.251	1.513	**0.200**
22	20-ol-AR	0.166	1.335	1.502	**0.171**	42	20-ol-FS	0.273	1.792	2.065	**0.181**
22	20-ol-BR	0.207	1.422	1.629	**0.194**	42	20-ol-GP	0.271	0.708	0.979	**0.584**
22	20-ol-BS	0.160	1.590	1.751	**0.193**	42	20-ol-PI	0.300	0.798	1.097	**0.981**

CHAPTER 4

The *k*-Color Shortest Path Problem

The *k-Color Shortest Path Problem* (*k*-CSPP) is an **NP**-hard problem recently proposed by Ferone et al. [71]: given a weighted, edge-colored, undirected graph, it aims at finding a shortest path from source to destination traversing at most k colors. Since only a Branch & Bound procedure has been proposed for *k*-CSPP [71], we devised an ad hoc Dynamic Programming algorithm for it in Ferone et al. [74].

In Section 4.1 we outline the motivations that led us to further investigate this problem, pointing out the novelty with respect to other Shortest Path Problems on edge-colored graphs. The mathematical formulation of *k*-CSPP is presented in Section 4.2. Then, in Section 4.3 we briefly describe the existing solution approach proposed in [71]; moreover, we detail our algorithmic proposal [74]. Finally, the remaining of the Chapter is devoted to the presentation of the results of the computational study conducted to appraise the performance of our proposal (Section 4.4).

4.1 Background and Motivations

The aim of this Section is to depict the background in which *k*-CSPP is located; moreover, we thoroughly detail its broad applicability in the design of transmission networks in order to point out the reasons that motivated our study.

Edge-colored networks received a fair share of attention in the scientific literature, given their aptness for the depiction of complex and diverse relations among nodes. This feature proved to be beneficial in a wide variety of application fields, such as computational biology [60], telecommunications [179], as well as in the analysis of transportation networks [5], and conflicts resolution [175].

In the study of edge-colored graphs, many works – of both theoretical and experimental interest – are concerned with the investigation of specific *properly-colored* edge structures, where a coloring is said to be *proper* whenever any two adjacent edges differ in color. These structures include for example: paths, trails, trees and cycles [see e.g. 94]. On the other hand, some classic optimization problems – such as the *Minimum Spanning Tree* (MST), the *Traveling Salesman Problem* (TSP), and the *Longest Path Problem* (LPP) – have all been extended to the case of edge-colored graphs, taking labels (i.e. colors) into account either in the objective function or in their constraints.

At this purpose, e.g. the *Minimum Label Spanning Tree* was defined in Chang and Shing-Jiuan [34] as a variant of the classic MST in which the cost of the spanning tree is given by the number of different edge-colors used; moreover, a strictly related generalization, namely the *k-Labeled Spanning Forest Problem*, was studied by Cerulli et al. [31]. Then, Jozefowiez et al. [113] conducted an in-depth analysis of the *Minimum Label Hamiltonian Cycle Problem* (MLHCP) consisting in determining a Hamiltonian cycle that presents the minimum total number of different edge-labels used. Moreover, they also introduced the MLHCP with length constraints and the TSP with label constraints (LCTSP). Actually, the aim of LCTSP is to minimize the length of the tour – as in the classic TSP – while constraining the maximum number of different colors that can be used. Finally, Carrabs et al. [28] recently studied a special case of LPP on edge-colored graphs, the *Orderly Colored Longest Path Problem*.

The main ground of interest for the k-CSPP arises in the field of telecommunications. While designing transmission networks, reliability is a crucial matter to ensure good performances and prevent data loss. Actually, the robustness of a path-routed communication network can be achieved by means of *path protection schemes*, which make use of *backup paths* to ensure reachability in the case of single link failures [178, 179]. The backup path and the primary path are link-disjoint, and share the same source and destination. To prevent traffic loss, the backup path is activated whenever the primary path fails.

On the other hand, often a single happening can cause the simultaneous failure of several links in the network. For instance, in WDM networks it is customary to bundle multiple fiber links in the same conduit. Consequently, even if these links are disjoint in the network layer, a damage to the conduit will cause the failure of all the links there bundled. The fibers sharing a common risk factor are said to be in the same *Shared Risk Link Group*, and are modeled with arcs of the same color (Fig. 4.1).

Yuan et al. [179] addressed the *Failure Minimization Problem* as a *Minimum-Color Path Problem*, in which each path is associated with one or more colors and each color is related to a given failure event. Then, minimizing the number k of different colors traversed in the path, could consequently minimize its probability of failure. Indeed, assuming that the color failures events are mutually independent and equiprobable – with probability $p \in [0,1]$ –, then the reliability of the path can be computed as $(1-p)^k$. Consequently, an upper bound on the number of different colors allows to have a probabilistic estimate on the reliability of the network. A similar argument can be repeated in the case of

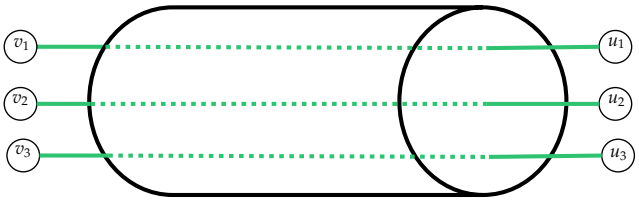


Fig. 4.1 Multiple fibers bundled in the same conduit modelled as arcs sharing the same color.

independent failure events with different probabilities.

However, while the arguments proposed in Yuan et al. [179] handled different edge-labels, they did not include lengths in the comparison of different paths. With in mind a similar network reliability scenario, the *k*-CSPP handles risk adversity as a strict requirement, while optimizing path length. Hence, the mathematical model introduced in Section 4.2 includes distances in the objective function, while encompassing the use of few colors in a problem constraint.

4.1.1 *k*-CSPP vs SPP on Edge-colored Graphs

In this Subsection we provide a brief description of two variants of Shortest Path Problem that include edge constraints, with the aim of pointing out their differences with the *k*-CSPP.

Notably, the Resource Constrained Shortest Path Problems (Resource CSPPs) [14, 51] represent one of the most broad classes of constrained SPPs. Specifically, each instance of Resource CSPP is characterized by a (un-)directed weighted graph along with a L-dimensional vector of resources R. Essentially, each resource is related to relevant link attributes that need to be accounted in the planning of the path. Accordingly, a path is optimal whenever it is minimal with respect to the weight function and satisfies the restrictions enforced on the resources associated to each arc (edge).

The fundamental difference between the *k*-CSPP and Resource CSPPs lies in the fact that the *k*-CSPP does not restricts the sets of colors *a priori*; additionally, there is a strong interdependence among edges. In our scenario, indeed, the cost of a color as a resource is not constant during the exploration of the solution space: once that an edge with a certain color is traversed, all other edges sharing the same colors turn free and thus can be inserted in the solution without placing additional burden on the color constraint.

Another variant of Shortest Path on edge-colored graphs comprises the use of *reload costs* which are defined, for each couple of colors b and c, as the amount to be paid if in the path an arc (edge) of color c is traversed after an arc of color b. At this purpose, Gourvès et al. [93] studied the *Minimum Reload Cost Problem* on walks, trails and paths, deriving the resulting computational complexities. On the other hand, Amaldi et al. [5] considered a general form of objective function that includes both distances and reload costs.

However, aside from their presence in the objective function rather than in the constraints, the main difference between reload costs and the modeling paradigm of the k-CSPP is that reload costs are fixed, and have to be taken into account any given time there is a change from a color to another. On the contrary, bounding the maximum number of different colors, as required in the k-CSPP, means to count just once a transition to a specific color, regardless of the preceding color (if any).

4.2 Mathematical Formulation

In this Section, we provide the mathematical formulation of k-CSPP, as described by Ferone et al. [71]. At this purpose, we consider an undirected weighted and colored graph $G = (V, E, w, C)$, where:

1. $w : E \longrightarrow \mathbb{R}_0^+$ is a function that assigns a non-negative distance w_{ij} to each edge $[i, j] \in E$,
2. $C : E \longrightarrow \mathbb{N}$ is a labeling function that assigns a color to each $[i, j] \in E$.

In the following, we will refer to G as an *edge-colored graph*. Additionally, let $C(\hat{E})$ be the set of different colors appearing in any subset $\hat{E} \subseteq E$, and $c(\hat{E}) = \left|C(\hat{E})\right|$; thus $c(E)$ is the total number of colors used to label the edges of G. Moreover, letting E_h, $\forall h \in \{1, \dots C(E)\}$, be the set of all the edges labeled with the color h, the set of edges E can be partitioned as $\bigcup_{h=1}^{C(E)} E_h$.

Given a source node s and a target node t, $s, t \in V$, $s \neq t$, the *k-Color Shortest Path Problem* (k-CSPP) aims at finding a minimum distance path P^* from s to t, consisting of edges of at most k different colors.

A solution for this problem can be modelled through the introduction of a Boolean decision variable x_{ij} for each edge $[i, j] \in E$ such that

$$x_{ij} = \begin{cases} 1, & \text{if } [i, j] \text{ belongs to } P^*; \\ 0, & \text{otherwise.} \end{cases}$$

Then, for each possible color h, a Boolean decision variable y_h is necessary, with

$$y_h = \begin{cases} 1, & \text{if color } h \text{ is traversed in } P^*; \\ 0, & \text{otherwise.} \end{cases}$$

Then, as done in [71], the k-CSPP is formally described as the following integer linear program:

$$(k\text{-CSPP}) \ \min \sum_{[i,j] \in E} w_{ij} x_{ij} \tag{4.1a}$$

subject to:

$$\sum_{\{j \,:\, [i,j] \in E\}} x_{ji} - \sum_{\{j \,:\, [i,j] \in E\}} x_{ij} = b_i, \qquad \forall i \in V \tag{4.1b}$$

$$x_{ij} \leq y_h, \qquad \forall [i,j] \in E_h, h \leq C(E) \tag{4.1c}$$

$$\sum_{h=1}^{C(E)} y_h \leq k \tag{4.1d}$$

$$x_{ij} \in \{0,1\}, \qquad [i,j] \in E \tag{4.1e}$$

$$y_h \in \{0,1\}, \qquad \forall h = 1, \ldots, C(E) \tag{4.1f}$$

with $b_i = -1$ for $i = s$, $b_i = 1$ for $i = t$, and $b_i = 0$ otherwise.

The objective function (4.1a) minimizes the total distance of the path. Constraints (4.1b) are classic flow-balancing restrictions; those (4.1c) connect edge traversal and color selection, while constraints (4.1d) limit the maximum number of different colors that can be used in the solution. Finally, the Boolean nature of the decision variables is expressed by (4.1e) and (4.1f).

4.2.1 Computational Complexity

In order to state the intractability of k-CSPP, we recall that in Broersma et al. [25], it is proved that finding a simple path $\bar{P}$ from s to t in an edge-colored graph, such that $c(\bar{P}) \leq k$ with a given k, is an **NP**-complete problem.

As a consequence of this computational complexity result, we have the following claim:

Lemma 4.2.1. *There does not exist a polynomial-time algorithm $\mathcal{A}$ to find a feasible solution for an arbitrary instance $\mathcal{I}$ of the k-CSPP, unless* **P** = **NP**.

Since approximation algorithms find feasible solutions with provable guarantees on solution quality, a polynomial-time approximation algorithm would find – if existing – a feasible solution in polynomial time. Consequently, Corollary 4.2.2 follows from Lemma 4.2.1.

Corollary 4.2.2. *There does not exist a polynomial-time approximation algorithm for the k-CSPP, unless* **P** = **NP**.

Remark 5. In Broersma et al. [25], it is proved that to find a path from a source node s to a destination node t with maximum k colors is an **NP**-complete problem, by reduction from the 3-SAT problem. We observe how any instance of the decision problem of finding an $s-t$ path with at most k colors can be represented as an instance of k-CSPP, where each edge has null cost. Therefore, a polynomial algorithm for the k-CSPP would efficiently solve the decision problem described in Broersma et al. [25]. Consequently, k-CSPP is **NP**-hard. □

4.3 Solution Approaches

The aim of this Section is twofold: on the one hand, we briefly describe the first approach proposed in literature for k-CSPP [71], i.e. a *Branch & Bound* based

algorithm; on the other hand, we thoroughly detail the *Dynamic Programming* method we devised in Ferone et al. [74] to tackle this problem.

The Branch & Bound is an algorithmic paradigm characterised by a procedure that implicitly enumerates all possible solutions to the problem under consideration. At this purpose, the partial solutions i.e. the *sub-problems*, are stored in a *tree data structure* whose exploration is carried out through two operations: *branching* and *bounding*. The former is meant to partition the solution space in smaller regions while the latter is used to prune off regions that are provably sub-optimal. The procedure terminates when the whole tree has been explored and the best found solution is given in output [135]. The Branch & Bound presented in Ferone et al. [71] for k-CSPP– namely, B&B – is briefly described in Section 4.3.1.

On the other hand, like the Branch & Bound, the *Dynamic Programming* is an exact algorithmic paradigm which constructs an optimal solution for the problem at hand by combining those of its sub-problems. Anyway, in contrast with the Branch & Bound, it divides – rather than partitioning – the original problem in *not-independent* sub-problems and solves them recursively. Indeed, this approach is suitable for all the problems $\mathcal{P}$ featuring the following characteristics [42]:

1. $\mathcal{P}$ has an *optimal sub-structure*;
2. $\mathcal{P}$ can be partitioned in not-independent sub-problems.

Indeed, 1. allows to define an optimal solution for $\mathcal{P}$ as a combination of the optimal solutions of its *elementary* sub-problems; 2., instead, speeds up the solution procedure allowing to solve each sub-problem exactly once. A thorough description of the Dynamic Programming approach we proposed in Ferone et al. [74] is given in Section 4.3.2. In particular, the design of this Dynamic Programming algorithm (DP) has been encouraged by the successful results that this family of solution framework gathered in the field of Constrained Shortest Path Problems [52, 150, 156].

4.3.1 Branch & Bound

The Branch & Bound algorithm proposed in Ferone et al. [71] is based on the observation that relaxing the color constraints (4.1d), the problem can be solved very efficiently by a classic shortest path algorithm.

Consequently, at each node of the branching tree, a Shortest Path Problem is solved on a given edge-colored graph $G' = (V, E')$ obtained with the branching operation. If $\hat{P}$ is the incumbent solution, $P^*_{G'}$ denotes the optimal solution obtained, $d(P^*_{G'})$ and $c(P^*_{G'})$ the total distance of the path and the number of different colors traversed by $P^*_{G'}$, respectively, then only four cases can occur:

1. $d(P^*_{G'}) = +\infty$: there is no path from s to t, the feasible region is empty, and the branching node becomes a leaf;
2. $d(P^*_{G'}) \geq d(\hat{P})$: the branching node is fathomed due to the bounding criterion;

3. $c(P^*_{G'}) \leq k$: the solution is feasible for the original problem, and the incumbent is updated if necessary;

4. $c(P^*_{G'}) = l > k$: the solution is not feasible for the original problem.

Indeed, the branching operation is performed only when case 4. occurs; at this purpose, letting $C(P^*_{G'}) = \{c_1, c_2, \dots c_l\}$ be the colors used by $P^*_{G'}$, for each $i = 1, \dots, l$ a new branching node is generated on the graph $G'' = (V, E'')$, where $E'' = E' \setminus \left\{[v, w] \in E' : C([v, w]) = c_i\right\}$.

Remark 6. For the seek of completeness, we report that the branching tree is explored with a *Depth First* strategy while colors are excluded according to their absolute frequencies in $P^*_{G'}$ i.e. the lesser used the color, the earlier it is excluded from G'. Ferone et al. [71] pointed out that these choices were made aiming at obtaining a feasible solution as quick as possible, in order exploit the bounding operation as much as possible. □

4.3.2 Dynamic Programming

The optimal solutions of the sub-problems solved by the Dynamic Programming algorithm (DP) devised in Ferone et al. [74] are subpaths, to which a certain label is assigned. Specifically, let P_{si} be a path in G connecting the source s with a generic node i, and let

$$L_i = (d_i, C_i, P_{si}) \tag{4.2}$$

be the label associated to P_{si}, where d_i and C_i are the total distance of the path and the set of different colors traversed by P_{si}, respectively. The source node s is given an initial label $L_s = (0, \emptyset, \langle s \rangle)$.

Following the general scheme of a *Label Correcting* technique, DP explores the solution space to extend the paths under construction by analyzing the set of their labels. Given a path P_{si}, the result of *path extension* operation is a path P_{sh} obtained concatenating P_{si} and $[i, h] \in E \setminus P_{si}$, and denoted as $\langle P_{si}, [i, h] \rangle$. Then, starting from a generic label $L_i = (d_i, C_i, P_{si})$, for each node $j \in V$ such that $[i, j] \in E$, new labels L_j are generated. In particular, the distance of the path $P_{sj} = \langle P_{si}, [i, j] \rangle$ is defined as $d_j = d_i + w_{ij}$, and the set of colors traversed is $C_j = C_i \cup \{C([i, j])\}$. Then, their evaluation is mainly based upon two fundamental concepts: *feasibility* and *dominance*.

Definition 4.3.1 (Feasibility)**.** A generated label L_j is feasible if $\left|C_j\right| \leq k$.

Definition 4.3.2 (Dominance)**.** Given two labels L_i and $\hat{L}_i$ associated with the same node $i \in V$, L_i is said to *dominate* $\hat{L}_i$ if the following conditions hold:

$$d_i \leq \hat{d}_i;$$
$$C_i \subseteq \hat{C}_i,$$

and at least one of such conditions is strict.

Thus, the generated labels – and the corresponding paths – are discarded if they are either associated with infeasible paths or dominated by other labels.

Theorem 4.3.3. *Let L_i and $\hat{L}_i$ be the labels associated to the paths P_{si} and $\hat{P}_{si}$, respectively. If L_i dominates $\hat{L}_i$, then, for any feasible extension $\left\langle \hat{P}_{si}, [i,h] \right\rangle$, there exists at least a feasible path $\mathcal{P}$ from s to h such that:*

1. *$\mathcal{P}$ is not longer than $\left\langle \hat{P}_{si}, [i,h] \right\rangle$;*
2. *$C(\mathcal{P}) \subseteq C\left(\left\langle \hat{P}_{si}, [i,h] \right\rangle\right)$.*

Proof. Let $\hat{\mathcal{P}} = \left\langle \hat{P}_{si}, [i,h] \right\rangle$ be a feasible extension of the path $\hat{P}_{si}$ and let $\mathcal{P}' = \langle P_{si}, [i,h] \rangle$ be the extension obtained by substituting path $\hat{P}_{si}$ with path P_{si}.

For path $\mathcal{P}'$ two cases can occur: either it is a feasible extension too or it is an infeasible extension due to the presence of a cycle. In the former case (Fig. 4.2), given the dominance of L_i over $\hat{L}_i$, the extension $\mathcal{P}'$ verifies conditions 1. and 2. In fact, according to Definition 4.3.2, path feasibility is preserved (since $C(\mathcal{P}') = C_i \cup \{C([i,h])\} \subseteq \hat{C}_i \cup \{C([i,h])\} = C(\hat{\mathcal{P}})$) while, the distance of P_{si} being not greater than that of $\hat{P}_{si}$ – i.e. $d_i \leq \hat{d}_i$ – ensures that distance-wise $\mathcal{P}'$ is not less favorable with respect to $\hat{\mathcal{P}}$. Hence, $\mathcal{P} = \mathcal{P}'$.

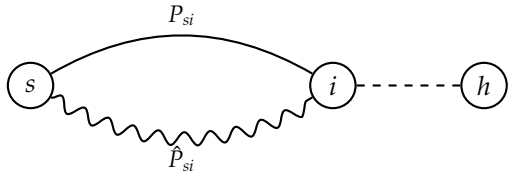


Fig. 4.2 Feasible concatenation of a "dominant" path (curved) with the extension (dashed) of a "dominated" path (normal).

In the latter case, a cycle in the extension $\mathcal{P}'$ occurs when the node h is visited by the dominant path P_{si}, as depicted in Fig. 4.3.

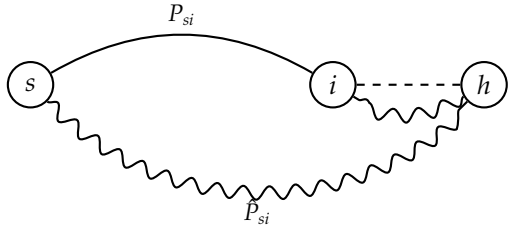


Fig. 4.3 Concatenation of a "dominant" path (curved) with the extension (dashed) of a "dominated" path (normal) containing a cycle.

Thus, given that $P_{sh} \subset P_{si}$ and $\hat{P}_{si} \subset \hat{\mathcal{P}}$, the dominance of L_i over $\hat{L}_i$ ensures that $C(P_{sh}) \subseteq C(P_{si}) \subseteq C(\hat{P}_{si}) \subseteq C(\hat{\mathcal{P}})$ while the distance of P_{sh} being not greater

than the distance of $\hat{\mathcal{P}}$. As a consequence, the subpath P_{sh} is a path from s to h which verifies conditions 1 and 2, i.e. $\mathcal{P} = P_{sh}$.
In conclusion, extending a dominated path yields to the construction of a dominated path. □

With the aim of pruning the space of solutions from those unfavourable, the Dynamic Programming algorithm collects only feasible and non-dominated labels in sets $D(i)$, which are the sets of all the labels L_i associated with the different paths connecting s to i, $\forall\, i \in V$.

Remark 7. It is duly noted how this approach preserves optimality, since there exists at least one optimal solution associated to a feasible and non-dominated label. In fact, if P^* is an optimal path associated to a dominated label, then there exists another feasible s-t path P^{**} such that $C(P^{**}) \subseteq C(P^*)$, and P^{**} is not longer than P^*. Since P^* is optimal, P^{**} and P^* have the same length, in terms of distance, and thus P^{**} is an optimal solution. □

The `DP` algorithm is summarized in Algorithm 6. Let Λ be the cost of the current incumbent. Lines from 2 to 4 initialize the first label, the list of labels L, the lists $D(j)$, $\forall\, j \in V$ and the cost Λ. While L is nonempty, the algorithm extracts a non-dominated feasible label L_i from L (Line 6). Then, if the total distance related to L_i is less than Λ, the incumbent is updated whenever $i = t$. Otherwise, the labels of each node $j \in V$ such that $[i, j] \in E$ are generated and added to the list L by means of the procedure `AddLabel` at Line 14 (see Algorithm 7). In particular, given a certain label L_j, this label-adding procedure includes L_j in L and $D(j)$ if it is feasible and non-dominated.

Algorithm 6 Dynamic Programming algorithm.

```
1:  procedure DP
2:      L_s = (0, ∅, ⟨s⟩)
3:      L = {L_s}; D(s) = {L_s}; D(j) = ∅, ∀j ∈ V, j ≠ s.
4:      best = Nil; Λ = +∞.
5:      while j ∈ V, j ≠ s do
6:          L_i = Extract(L); L ← L ∖ {L_i}.
7:          if d_i < Λ then
8:              if i = t then
9:                  Λ = d_i
10:                 best = L_i
11:             else
12:                 for j ∈ V : [i, j] ∈ E
13:                     L_j = (d_i + w_ij, C_i ∪ {C([i, j])}, ⟨P_si, [i, j]⟩)
14:                     AddLabel.
15:     end while
16:     return best
17: end procedure
```

Finally, `DP` returns the best found solution corresponding to the optimal one.

Algorithm 7 Label Adding procedure.

```
1: procedure AddLabel
2:     if L_j is feasible then
3:         if L_j is not dominated by any label L'_j ∈ D(j) then
4:             remove from D(j) and L all labels L'_j dominated by L_j
5:             L = L ∪ {L_j}
6:             D(j) = D(j) ∪ {L_j}
7: end procedure
```

Label Extraction Policy

In a Dynamic Programming technique, the choice of a well-performing *label extraction policy* (Algorithm 6, Line 6) can be a determining factor in favoring the fast convergence to good quality solutions, that combined with the pruning of dominated labels can increase the performance of the algorithm.

The following list briefly introduces some of the best-known strategies that can guide the label-extraction operations, in the case of `DP`.

Dijkstra-like Rule (DR): the label with the smallest distance is extracted;

First-In First-Out (FIFO): the label that has been in the queue for the longest time is extracted;

Last-In First-Out (LIFO): the last inserted label is the first extracted;

Small-Label-First (SLF) [20]: the extraction is performed from the top of the list L. Moreover, when a new label L_j has to be added to L, if its total distance is less than the distance of the currently top label of L, label L_j is entered at the top of L; otherwise L_j is entered at the bottom of L;

A*: the label that presents the smallest value of the sum between the distance d_i and the distance of the simple shortest path from the current node i to the target node t is extracted.

Remark 8. Interestingly enough, we observe that the function used in the A* strategy, to evaluate the length of the path from the current node i to the target node t, gives a lower bound on the cost of the optimal path from i to t. In fact, that minimum cost path is computed without taking into account constraints (4.1d).

As reported in Hart et al. [98], A* finds an optimal solution whenever the length of the path from the current node to the target one is estimated with a lower bound. As a consequence, the first feasible solution found by the A* strategy is indeed optimal, thus allowing to prune all the remaining labels.

Remark 9. As a last observation, we note how the computational effort related to the execution of a Dynamic Programming algorithm is related to the number of explored labels. Without assuming the use of a specific extraction policy, in

the worst possible case the number of explored labels is equal to the number of feasible k-colored paths from s to any node $i \in V$.

Assuming the use of A^* as extraction policy, let $N(k)$ be the total number of combinations without repetitions of l elements of $C(E)$, with $l = 1, \ldots, k$, i.e.

$$N(k) = \sum_{l=1}^{k} \binom{C(E)}{l}. \tag{4.3}$$

The number of labels extracted for each node $v \in V \setminus \{s, t\}$ is bounded by $N(k)$, since for each feasible combination of colors, the A^* strategy extracts only the most favorable path that dominates others characterized by the same set of colors. As a consequence of this, the number of iterations of the `DP` algorithm is bounded by $(|V| - 2) \cdot N(k)$. Each one of the operations executed in a single iteration can be carried out in $O(1)$, except for the `AddLabel` operation (line 14), whose complexity is linear in the size of $D(j)$. The size of $D(j)$ can be estimated by $N(k)$, since for each feasible combination of colors, only the dominating path is kept in memory. Therefore, an upper bound for the complexity of `DP` with the A^* strategy is $O((|V| - 2) \cdot N(k)^2)$. □

4.4 Computational Experiments

In this Section, we describe the computational experiments designed in Ferone et al. [74] to appraise the performances of `DP` when compared with two other different solution approaches, namely the Branch & Bound (B&B) algorithm [71] described in Section 4.3.1, and the direct solution of the mathematical model (Section 4.2) obtained by means of the ILOG CPLEX solver.

4.4.1 Test Problems

The experimentation has been conducted on two data-sets[1] – namely $\mathcal{A}$ and $\mathcal{B}$ – whose characteristics are summarized in Tables 4.1 and 4.2.

The set $\mathcal{A}$ consists of the networks described in Ferone et al. [71]. These instances were obtained by adapting the generator presented in Festa and Pallottino [80]; specifically, they can be divided in two classes:

Fully random graphs. The number n of nodes for these graphs varies in the set $\{75000, 100000, 125000\}$, while the number m of edges is set equal to $10 \cdot n, 15 \cdot n$ and $20 \cdot n$. Then, the total number of colors for each instance is $p \cdot m$ with $p \in \{0.15, 0.20\}$.

Grid graphs. Their underlying structure is analogous to that of `grid` instances described in Section 2.4.2. Indeed, we generated both square and rectangular grids with size $n \times n$ and $n \times 2n$, where $n = 100, 250, 500$. The total

[1] The full data-set is available at `https://figshare.com/articles/Instances_for_the_k-color_shortest_path_problem/11762163`.

number of colors for each instance is $p \cdot m$ with $p \in \{0.15, 0.20\}$, where m denotes the number of edges.

Finally, in order to avoid instance-triviality, the value for k has been determined solving a Shortest Path problem on each instance G. In particular, if P^* is a shortest path connecting s and t in G, and C^* is the number of different colors traversed by P^*, then k is selected as $C^* - 2$.

Letting $\{R1, \ldots, R9, G1, \ldots, G6\}$ be the set of possible combinations of graph size and number of colors, we have that each of them characterizes a collection of similar instances. Indeed, each collection contains ten different instances of the same type thus resulting in 300 instances. The characteristics of the data-set are summarized in Table 4.1.

Table 4.1 Instance parameters for data-set $\mathcal{A}$.

Fully random graphs					Grid graphs			
Combo	p	**Nodes**	**Arcs**	**Colors**	**Combo**	p	**Size**	**Colors**
R1	0.15	75000	750000	112500	G1	0.15	100×100	5940
R1	0.20	75000	750000	150000	G1	0.20	100×100	7920
R2	0.15	75000	112500	168750	G2	0.15	100×200	11910
R2	0.20	75000	112500	225000	G2	0.20	100×200	15880
R3	0.15	75000	150000	225000	G3	0.15	250×250	37350
R3	0.20	75000	150000	300000	G3	0.20	250×250	49800
R4	0.15	100000	1000000	150000	G4	0.15	250×500	74775
R4	0.20	100000	1000000	200000	G4	0.20	250×500	99700
R5	0.15	100000	1500000	225000	G5	0.15	500×500	149700
R5	0.20	100000	1500000	300000	G5	0.20	500×500	199600
R6	0.15	100000	2000000	300000	G6	0.15	500×1000	299550
R6	0.20	100000	2000000	400000	G6	0.20	500×1000	399400
R7	0.15	125000	1250000	187500				
R7	0.20	125000	1250000	250000				
R8	0.15	125000	1875000	281250				
R8	0.20	125000	1875000	375000				
R9	0.15	125000	2500000	375000				
R9	0.20	125000	2500000	500000				

Moreover, in order to better analyze how the performances of the algorithms are affected by a variation in the number of colors, we generated a second set of instances, namely $\mathcal{B}$. This data-set replicates the graph sizes of set $\mathcal{A}$, but p ranges in $\{0.01, 0.02\}$ meaning that the total number of colors in the networks has been reduced. The characteristics are summarized in Table 4.2.

4.4.2 Implementation Details

All the compared algorithms have been coded in C++ using the flags `-std=c++17 -O3` and compiled with `g++ 8.2`. The experiments were run on a INTEL i5-6400@2.70 GHz processor with 8GB of RAM. Finally, version 12.9 of ILOG CPLEX has been used. A time limit of 10 minutes has been used for each solution method.

Table 4.2 Instance parameters of data-set $\mathcal{B}$.

Fully random graphs					Grid graphs			
Combo	p	**Nodes**	**Arcs**	**Colors**	**Combo**	p	**Size**	**Colors**
R1	0.01	75000	750000	7500	G1	0.01	100×100	396
R1	0.02	75000	750000	15000	G1	0.02	100×100	792
R2	0.01	75000	112500	11250	G2	0.01	100×200	794
R2	0.02	75000	112500	22500	G2	0.02	100×200	1588
R3	0.01	75000	150000	15000	G3	0.01	250×250	2490
R3	0.02	75000	150000	30000	G3	0.02	250×250	4980
R4	0.01	100000	1000000	10000	G4	0.01	250×500	4985
R4	0.02	100000	1000000	20000	G4	0.02	250×500	9970
R5	0.01	100000	1500000	15000	G5	0.01	500×500	9980
R5	0.02	100000	1500000	30000	G5	0.02	500×500	19960
R6	0.01	100000	2000000	20000	G6	0.01	500×1000	19970
R6	0.02	100000	2000000	40000	G6	0.02	500×1000	39940
R7	0.01	125000	1250000	1250				
R7	0.02	125000	1250000	25000				
R8	0.01	125000	1875000	18750				
R8	0.02	125000	1875000	37500				
R9	0.01	125000	2500000	25000				
R9	0.02	125000	2500000	50000				

In particular, in this Section we presented the results of a preliminary study performed to detect the best performing label extraction policy for DP and the best branching rule for B&B [see 74]. In fact, as pointed out in Section 4.3.2, the choice of the label extraction policy is crucial when designing a Dynamic Programming algorithm, since it can strongly affect its performance – in terms of speed of convergence and quality of the solutions –. In the same way, the performance of a Branch & Bound method – in terms of computation time and memory used – strongly depends on the adopted *branching strategy* [135].

Finally, the best versions of both the algorithms are compared in Section 4.4.3.

Analysis of the Extraction Policies

This analysis aims to appraise how the computational performance of DP is affected by the different label selection policies. At this purpose, we randomly selected 2 out of 10 instances of each type from data-set $\mathcal{A}$ and left DP – with one of the extraction policies, in turn – run with a time limit of 10 minutes.

The results are given in Tables 4.8 and 4.9 in Appendix 4.A. They point out that the best performing strategy is A*, both in terms of number of optimal solutions found and running times. This behaviour depends on the criterion used for the selection of the label L_i to be extracted: the value of the path from node i to the target t is an estimation (i.e. a lower bound) of the final cost that can be obtained starting from the path associated with L_i.

As a consequence, the convergence to a feasible – indeed optimal – solution is more rapid with respect to the other strategies and a significant number of opened labels can be pruned when the optimum is found (see Remark 8).

Analysis of the Branching Strategies

The analysis presented in this Section is meant to determine which strategy of exploration of the branching tree is advisable for the Branch & Bound algorithm in Ferone et al. [71]. At this purpose, we compared the following two widely used strategies.

Best-First (BF). The branching node is selected as the node corresponding to the sub-problem whose relaxed solution has the highest cost.

Depth-First (DF). The branching node is chosen as the most recently generated sub-problem i.e. a node from the deepest level is used as branching node.

The results are collected in Tables 4.10 and 4.11 in Appendix 4.B. Indeed, though none of the two strategies outperforms the other one, we observe that BF presents slightly lower computational times and is able to find an extra optimal solution (compared with DF) for the grid graphs.

4.4.3 Comparison of Algorithms

The analyses conducted in Section 4.4.2 highlight that A* is the best performing extraction policy for the proposed solution approach while the Branch & Bound method in Ferone et al. [71] has slightly better performance when the best first branching strategy is adopted. Thus, the last experimentation compares DP with A* extraction strategy, B&B with BF branching strategy, and the solution of the mathematical model performed with CPLEX.

The results relative to the instances of data-set $\mathcal{A}$ are collected in Tables 4.3 and 4.4, while those relative to the instances of data-set $\mathcal{B}$ are given in Tables 4.5 and 4.6. For each instance type, the average running time ("avg(time)") in seconds, the number "O" of optimal solutions found within the limit and the number "F" of feasible – i.e. not optimal – solutions found within the time limit, are presented. Lowest running times and the corresponding O and F numbers are reported in bold.

Results on data-set $\mathcal{A}$

The analysis of the performance achieved by the algorithms on the fully random instances (Table 4.3), highlights that DP presents sensibly smaller average computational times, being at least an order of magnitude lower with respect to those achieved by B&B and CPLEX. Additionally, the number of optimal solutions detected – within the time limit – by our novel method, i.e. $^{176}/_{180}$ = 97.78%, almost doubles the number of optimal solutions found by the second best algorithm, i.e., $^{93}/_{180}$ = 51.67% optima found by B&B.

As regards the resolution of k-CSPP on grid graphs, the obtained results (Table 4.4) underline that, in general, they are more challenging for both B&B and DP. In fact, in contrast with the CPLEX solver, both the algorithms require on average higher computational times compared with those reported in Table 4.3.

Table 4.3 (Fully random graphs - data-set $\mathcal{A}$). Computational results in terms of average time, and number of optimal (O) and feasible (F) solutions found by each method within the 10 minutes time limit.

		B&B BF		CPLEX		DP	
Instance	p	**avg(time)**	**O+F**	**avg(time)**	**O+F**	**avg(time)**	**O+F**
R1	0.15	301.44	5 + 2	92.43	10 + 0	**1.56**	**10** + 0
R1	0.20	301.98	5 + 4	89.12	9 + 0	**2.35**	**9** + 0
R2	0.15	182.08	7 + 2	158.54	9 + 0	**60.24**	**9** + 0
R2	0.20	182.09	7 + 2	163.05	9 + 0	**60.24**	**9** + 0
R3	0.15	304.27	5 + 5	521.92	2 + 0	**0.47**	**10** + 0
R3	0.20	304.21	5 + 5	513.47	2 + 0	**0.49**	**10** + 0
R4	0.15	300.18	5 + 3	123.57	10 + 0	**0.65**	**10** + 0
R4	0.20	258.91	6 + 1	218.52	8 + 0	**3.95**	**9** + 0
R5	0.15	360.18	4 + 6	600	0 + 0	**0.47**	**10** + 0
R5	0.20	360.18	4 + 6	600	0 + 0	**0.50**	**10** + 0
R6	0.15	120.36	8 + 2	600	0 + 0	**0.49**	**10** + 0
R6	0.20	120.36	8 + 2	600	0 + 0	**0.50**	**10** + 0
R7	0.15	379.21	4 + 5	600	0 + 0	**0.43**	**10** + 0
R7	0.20	379.44	4 + 5	512.84	2 + 0	**0.43**	**10** + 0
R8	0.15	420.31	3 + 5	600	0 + 0	**0.63**	**10** + 0
R8	0.20	420.26	3 + 5	600	0 + 0	**0.64**	**10** + 0
R9	0.15	305.46	5 + 5	600	0 + 0	**0.71**	**10** + 0
R9	0.20	304.75	5 + 5	600	0 + 0	**0.72**	**10** + 0
Average		294.76		432.97		**7.53**	
Sum			93 + 70		61 + 0		**176** + 0

Table 4.4 (Grid graphs - data-set $\mathcal{A}$). Computational results in terms of average time, and number of optimal (O) and feasible (F) solutions found by each method within the 10 minutes time limit.

		B&B BF		CPLEX		DP	
Instance	p	**avg(time)**	**O+F**	**avg(time)**	**O+F**	**avg(time)**	**O+F**
G1	0.15	422.02	3 + 6	41.05	10 + 0	**0.92**	**10** + 0
G1	0.20	420.95	3 + 6	**42.16**	**10** + 0	60.81	9 + 0
G2	0.15	483.70	2 + 6	200.71	10 + 0	**3.81**	**10** + 0
G2	0.20	498.76	2 + 6	225.43	9 + 0	**12.69**	**10** + 0
G3	0.15	575.96	1 + 9	596.76	1 + 7	**130.83**	**8** + 0
G3	0.20	544.83	1 + 9	587.55	2 + 7	**130.01**	**8** + 0
G4	0.15	434.49	4 + 6	602.28	0 + 10	**64.99**	**9** + 0
G4	0.20	421.90	4 + 6	602.36	0 + 10	**62.00**	**9** + 0
G5	0.15	540.04	1 + 9	607.63	0 + 9	**367.50**	**4** + 0
G5	0.20	486.18	2 + 8	607.07	0 + 10	**362.37**	**4** + 0
G6	0.15	540.11	1 + 9	600.00	0 + 0	**232.51**	**7** + 0
G6	0.20	540.10	1 + 9	600.00	0 + 0	**275.19**	**6** + 0
Average		492.42		442.75		141.97	
Sum			25 + 89		42 + 53		94 + 0

Moreover, from this comparison it is possible to note a sensible decrease in the number of optimally solved instances, for all the solution techniques here considered. This behaviour can be attributed to the specific topology characterizing grids. In fact, such graphs are much sparser than the random networks of R1-R9, and the search for a shortest path coupled with a color restriction represents a harder task. Nonetheless, both the number of optimal solutions and the average computational times exhibited by DP outperform its two competitors.

Finally, we observe that on both the typology of networks, DP either converges to the optimal solution in the time-limit or is not able to find a feasible solution. Such an outcome is strongly dependent on the considered extraction policy (recall Remark 8). On the contrary, B&B is able to encounter feasible solutions in almost the totality of the tackled instances ($^{277}/_{300} = 92.33\%$).

Indeed, it is properly noted how the average times alone are not sufficient to grasp the performances achieved by the algorithms. This is clear, for example, when considering the distribution of computational times spent by the algorithms on G1 instances with $p = 0.20$, reported in the bar-chart of Fig. 4.4.

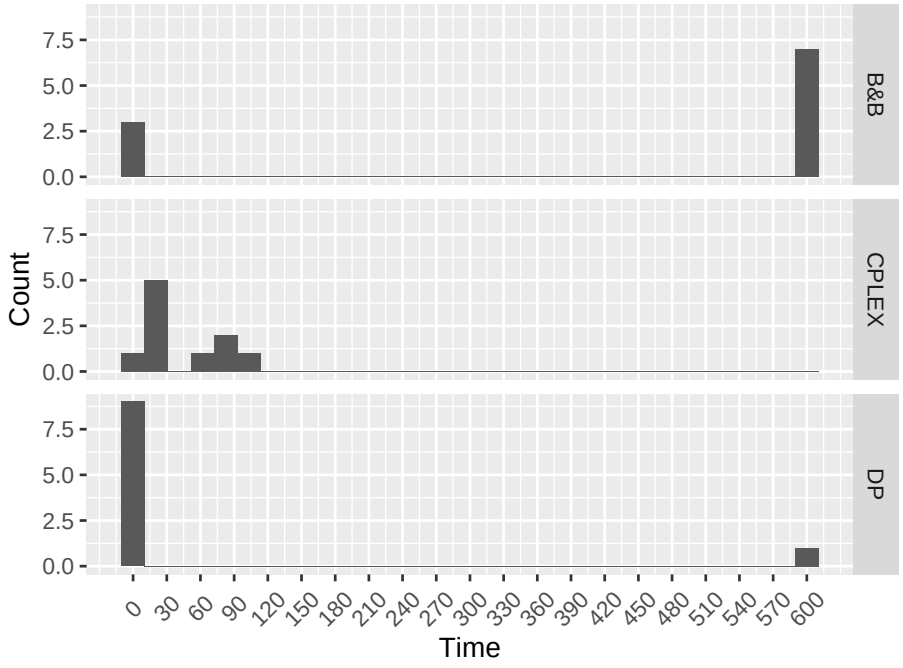


Fig. 4.4 Distribution of computational times on G1 instances with $p = 0.20$.

Among the three methods, it is evident how DP reaches the optimal solution in less than 1 second on 9 graphs, while on a single instance it is not able to find a feasible solution within the given time limit (600 seconds). The resulting average time of 60 seconds (Table 4.11) can not be clearly compared with the

one achieved by CPLEX, equal to 42 seconds, resulting from generally higher times with smaller variance.

As a consequence, we conducted a study of the distribution of computational times on the whole data-set (Fig. 4.5). It shows that DP is often extremely fast in the construction of an optimal solution, and in few cases struggles in the pursuit of a feasible solution. On the contrary, for B&B and CPLEX, the number of instances requiring the whole allotted time limit, or in general times greater than few seconds, is sensibly higher.

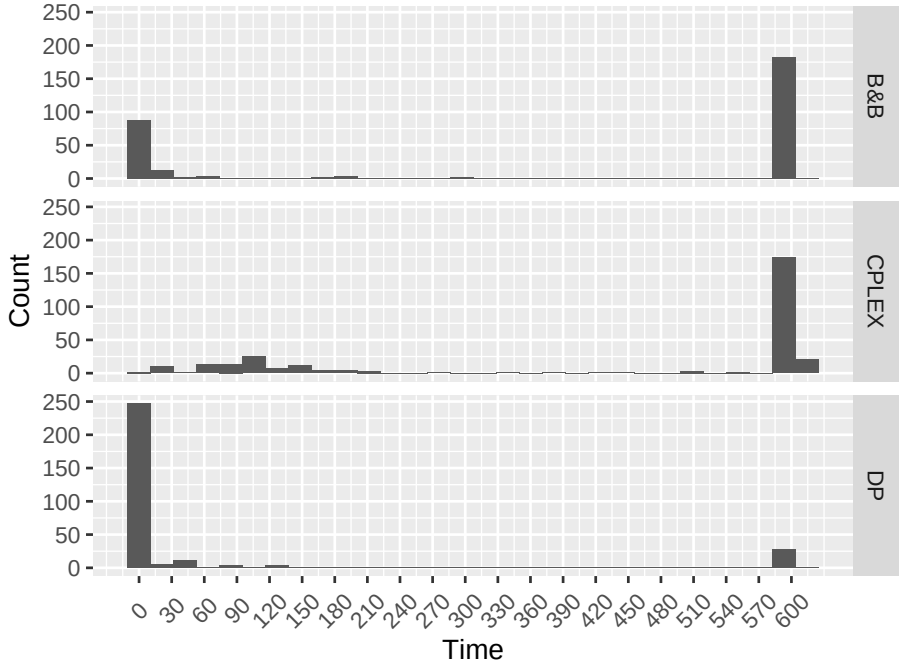


Fig. 4.5 Distribution of computational times on the whole data-set $\mathcal{A}$.

Results on data-set $\mathcal{B}$

In order to study the performance while varying the number of colors, we executed the three algorithms on the instances of data-set $\mathcal{B}$. The results are given in Table 4.5 and Table 4.6, respectively.

When analysing these data, it is immediately evident how well DP performs in comparison with the other two methods on the whole data-set. In fact, it finds the optima for all the instances and the average computational times are between one and two orders of magnitude lower than those of B&B and CPLEX.

The results on data-sets $\mathcal{A}$ and $\mathcal{B}$ are compared to understand how the overall behaviours are affected by the number of colors. Table 4.7 reports this summarized comparison.

Table 4.5 (Fully random graphs - data-set $\mathcal{B}$). Computational results in terms of average time, and number of optimal (O) and feasible (F) solutions found by each method within the 10 minutes time limit.

		B&B BF		CPLEX		DP	
Instance	p	**avg(time)**	**O+F**	**avg(time)**	**O+F**	**avg(time)**	**O+F**
R1	0.01	361.88	4 + 4	151.50	10 + 0	**0.28**	**10** + 0
R1	0.02	361.89	4 + 4	98.44	10 + 0	**0.29**	**10** + 0
R2	0.01	313.24	5 + 5	200.60	10 + 0	**0.29**	**10** + 0
R2	0.02	312.99	5 + 5	142.71	10 + 0	**0.29**	**10** + 0
R3	0.01	370.50	4 + 4	600.00	0 + 0	**0.91**	**10** + 0
R3	0.02	370.58	4 + 4	560.25	1 + 0	**1.01**	**10** + 0
R4	0.01	260.80	6 + 2	393.65	5 + 0	**0.32**	**10** + 0
R4	0.02	199.88	7 + 1	363.72	5 + 0	**0.31**	**10** + 0
R5	0.01	360.19	4 + 6	600.00	0 + 0	**0.40**	**10** + 0
R5	0.02	360.19	4 + 6	600.00	0 + 0	**0.40**	**10** + 0
R6	0.01	360.21	4 + 5	600.00	0 + 0	**0.80**	**10** + 0
R6	0.02	360.21	4 + 5	600.00	0 + 0	**0.84**	**10** + 0
R7	0.01	378.94	4 + 5	429.33	5 + 0	**0.45**	**10** + 0
R7	0.02	378.89	4 + 5	391.06	5 + 0	**0.47**	**10** + 0
R8	0.01	420.26	3 + 6	600.00	0 + 0	**1.01**	**10** + 0
R8	0.02	420.25	3 + 6	600.00	0 + 0	**1.16**	**10** + 0
R9	0.01	446.22	3 + 5	600.00	0 + 0	**0.81**	**10** + 0
R9	0.02	446.08	3 + 5	600.00	0 + 0	**0.82**	**10** + 0
Average		360.18		451.74		**0.60**	
Sum			75 + 83		61 + 0		**180** + 0

Table 4.6 (Grid graphs - data-set $\mathcal{B}$). Computational results in terms of average time, and number of optimal (O) and feasible (F) solutions found by each method within the 10 minutes time limit.

		B&B BF		CPLEX		DP	
Instance	p	**avg(time)**	**O+F**	**avg(time)**	**O+F**	**avg(time)**	**O+F**
G1	0.01	228.44	7 + 3	25.36	10 + 0	**0.03**	**10** + 0
G1	0.02	365.88	4 + 6	34.44	10 + 0	**0.07**	**10** + 0
G2	0.02	360.74	4 + 5	110.25	10 + 0	**0.11**	**10** + 0
G2	0.01	296.94	6 + 4	122.44	10 + 0	**0.09**	**10** + 0
G3	0.01	445.39	3 + 7	600.62	0 + 0	**0.92**	**10** + 0
G3	0.02	426.04	3 + 7	559.21	4 + 5	**0.79**	**10** + 0
G4	0.01	414.45	4 + 6	601.25	0 + 0	**1.01**	**10** + 0
G4	0.02	432.01	3 + 7	601.22	0 + 10	**2.41**	**10** + 0
G5	0.01	369.38	4 + 6	600.00	0 + 0	**50.67**	**10** + 0
G5	0.02	371.41	4 + 6	600.00	0 + 0	**18.37**	**10** + 0
G6	0.01	425.70	3 + 7	600.00	0 + 0	**46.33**	**10** + 0
G6	0.02	481.60	2 + 8	600.00	0 + 0	**69.49**	**10** + 0
Average		384.83		421.23		**15.86**	
Sum			47 + 72		44 + 15		**120** + 0

Table 4.7 Comparison for data-sets $\mathcal{A}$ and $\mathcal{B}$. Results are in terms of average time, and number of optimal (O) and feasible (F) solutions found by each method within the 10 minutes time limit.

		B&B BF		CPLEX		DP	
data-set	**Topology**	**avg(time)**	**O+F**	**avg(time)**	**O+F**	**avg(time)**	**O+F**
$\mathcal{A}$	Random	294.76	93 + 70	432.97	61 + 0	7.53	176 + 0
$\mathcal{A}$	Grid	492.42	25 + 89	442.75	42 + 53	141.97	94 + 0
$\mathcal{B}$	Random	360.18	75 + 83	451.74	61 + 0	0.60	180 + 0
$\mathcal{B}$	Grid	384.83	47 + 72	421.23	44 + 15	15.86	120 + 0

The results highlight that the efficiency of DP is strongly affected by the number of colors. In fact, for DP the ratio between the average computational time on $\mathcal{B}$ over computational time on $\mathcal{A}$ is equal to 0.07 and 0.11 for fully random graphs, and for grid graphs, respectively. This behaviour was expected, since the lower the number of colors, the lower the number of feasible non-dominated labels (see Remark 9).

On the contrary, it is not possible to identify a clear trend in the performance of B&B and CPLEX at varying the number of colors. On the one hand, on grid graphs, the computational times tend to decrease when lowering the number of colors. On the contrary, the trend appears to be inverse on fully random graphs.

4.5 Conclusions

The topics addressed in this Chapter are located in the framework of Constrained Shortest Path Problems (CSPP). Specifically, the Chapter is devoted to the discussion of a recent variant of CSPP proposed in Ferone et al. [71], i.e. the *k-Colored Shortest Path* (*k*-CSPP). At this purpose, both the differences with similar CSPPs on colored graphs and the existing solution approach have been briefly outlined.

Additionally, we thoroughly described a novel solution framework devised in Ferone et al. [74] which consists of a Dynamic Programming (DP) algorithm based on a *path-labeling approach* and an *A*-like exploration strategy*.

With the aim of validating this approach, we compared the performances of DP with those achieved by two alternative methods: the direct solution of the mathematical model obtained with CPLEX solver, and the Branch & Bound method described in Ferone et al. [71]. Moreover, these three techniques have been tested on two data-sets comprising two different graph topologies: *fully random* and *grid*. The experimental results show that DP outperforms its two competitors both in terms of computational times and number of optimal solutions found within the given time limit. Specifically, due to the features of the label extraction policy, either it converges to an optimum or it is not able to determine a feasible solution within the time limit. Moreover, our algorithm shows average computational times between one and two orders of magnitude lower than those of the other two approaches. Finally, we were also able to observe an improvement of the performance of DP related to a lower number of possible colors.

Due to the computational intractability of the problem and considering the obtained results, the future research streams will be mainly focused on the design of efficient heuristic approaches with the aim of solving large size instances in short computational time. Moreover, in order to properly model the complexity of real-world networks, different failure probabilities p_c, for each color $c \in C(E)$, could be considered; then the resulting bi-objective problem could be solved through a sim-heuristic approach [72, 82].

Appendix

Appendix 4.A Comparison of Label Extraction Policies

This Section provides the results of the analysis conducted in Ferone et al. [74] in order to detect the best performing extraction policy for DP, among those widely used for a Dynamic Programming algorithm (see Section 4.3.2).

Specifically, Tables 4.8 and 4.9 report, for each instance type and each extraction policy, the average running time ("avg(time)") in seconds and the number "O" of optimal solutions found by DP within the time limit. Lowest average times and its corresponding "O" number are reported in bold.

Table 4.8 (Fully random graphs). Comparison of Dijkstra-like, First-In First-Out, Last-In First-Out, Small-Label-First and A* extraction policies.

		DR		FIFO		LIFO		SLF		A*	
Instance	p	**avg(time)**	**O**	**avg(time)**	**O**	**avg(time)**	**O**	**avg(time)**	**O**	**avg(time)**	**O**
R1	0.15	300.23	1	1.70	2	3.68	2	0.86	2	**0.30**	**2**
R1	0.20	301.37	1	1.04	2	10.99	2	1.97	2	**0.32**	**2**
R2	0.15	300.75	1	300.22	1	305.07	1	300.45	1	**300.20**	**1**
R2	0.20	9.20	2	1.75	2	5.05	2	2.77	2	**0.36**	**2**
R3	0.15	307.33	1	5.20	2	11.10	2	2.85	2	**0.81**	**2**
R3	0.20	600.00	0	302.06	1	2.92	2	14.08	2	**0.53**	**2**
R4	0.15	1.49	2	0.58	2	6.41	2	2.04	2	**0.34**	**2**
R4	0.20	301.41	1	3.28	2	17.60	2	6.64	2	**0.36**	**2**
R5	0.15	600.00	0	4.20	2	16.43	2	302.31	1	**1.10**	**2**
R5	0.20	600.00	0	302.01	1	25.24	2	5.61	2	**0.71**	**2**
R6	0.15	4.36	2	3.17	2	304.80	1	9.49	2	**0.51**	**2**
R6	0.20	301.12	1	302.83	1	326.87	1	303.83	1	**0.51**	**2**
R7	0.15	301.49	1	300.79	1	306.41	1	1.23	2	**0.50**	**2**
R7	0.20	6.57	2	300.32	1	12.95	2	4.69	2	**0.52**	**2**
R8	0.15	13.84	2	2.24	2	17.78	2	3.56	2	**0.59**	**2**
R8	0.20	301.24	1	11.94	2	14.14	2	7.79	2	**0.93**	**2**
R9	0.15	303.51	1	300.75	1	32.45	2	301.13	1	**0.82**	**2**
R9	0.20	600.00	0	301.07	1	331.79	1	302.31	1	**0.83**	**2**
Average		286.33		135.84		97.32		87.42		**17.24**	
Sum			19		28		31		31		**35**

Table 4.9 (Grid graphs). Comparison of Dijkstra-like, First-In First-Out, Last-In First-Out, Small-Label-First and A* extraction policies.

		DR		FIFO		LIFO		SLF		A*	
Instance	p	**avg(time)**	**O**	**avg(time)**	**O**	**avg(time)**	**O**	**avg(time)**	**O**	**avg(time)**	**O**
G1	0.15	600	0	600	0	600	0	600	0	**0.20**	**2**
G1	0.20	600	0	600	0	600	0	600	0	**0.19**	**2**
G2	0.15	600	0	600	0	600	0	600	0	**12.33**	**2**
G2	0.20	600	0	600	0	600	0	600	0	**0.10**	**2**
G3	0.15	600	0	600	0	600	0	600	0	**0.70**	**2**
G3	0.20	600	0	600	0	600	0	600	0	**3.59**	**2**
G4	0.15	600	0	600	0	600	0	600	0	**3.33**	**2**
G4	0.20	600	0	600	0	600	0	600	0	**1.62**	**2**
G5	0.15	600	0	600	0	600	0	600	0	600.00	0
G5	0.20	600	0	600	0	600	0	600	0	**9.38**	**2**
G6	0.15	600	0	600	0	600	0	600	0	**120.21**	**2**
G6	0.20	600	0	600	0	600	0	600	0	**101.83**	**2**
Average		600		600		600		600		**71.12**	
Sum			0		0		0		0		**22**

Appendix 4.B Comparison of Branching Strategies

This Section provides the results of the analysis conducted in Ferone et al. [74] to detect the best performing branching strategy (best first vs depth first) for the Branch & Bound described in Ferone et al. [71].

Specifically, the results are collected in Tables 4.10 and 4.11 which report: the average time for the resolution of instances of the reference type ("avg(time)"), the number "O" of optimal solutions found within the time limit and the number "F" of feasible (not optimal) solutions found within the time limit.

Table 4.10 (Fully random graphs). Comparison of best first (BF) and depth first (DF) branching strategy.

		BF		DF	
Instance	p	**avg.time**	**O + F**	**avg. time**	**O + F**
R1	0.15	0.19	2 + 0	1.17	2 + 0
R1	0.20	9.52	2 + 0	40.55	2 + 0
R2	0.15	300.39	1 + 0	302.18	1 + 0
R2	0.20	300.13	1 + 1	300.74	1 + 1
R3	0.15	600.00	0 + 2	600.01	0 + 2
R3	0.20	314.10	1 + 1	378.88	1 + 1
R4	0.15	0.30	2 + 0	1.06	2 + 0
R4	0.20	0.28	2 + 0	1.09	2 + 0
R5	0.15	600.00	0 + 2	600.00	0 + 2
R5	0.20	600.00	0 + 2	600.00	0 + 2
R6	0.15	0.41	2 + 0	1.18	2 + 0
R6	0.20	0.41	2 + 0	2.07	2 + 0
R7	0.15	300.49	1 + 1	300.61	1 + 1
R7	0.20	394.25	1 + 1	420.85	1 + 1
R8	0.15	300.27	1 + 1	300.32	1 + 1
R8	0.20	600.00	0 + 2	600.00	0 + 2
R9	0.15	309.33	1 + 1	311.26	1 + 1
R9	0.20	313.93	1 + 1	316.92	1 + 1
Average		274.67		282.16	
Sum			20 + 15		20 + 15

Table 4.11 (Grid graphs). Comparison of best first (BF) and depth first (DF) branching strategy.

		BF		DF	
Instance	p	**avg.time**	**O + F**	**avg. time**	**O + F**
G1	0.15	600.00	0 + 2	600.00	0 + 2
G1	0.20	300.00	1 + 1	300.01	1 + 1
G2	0.15	600.00	0 + 1	600.82	0 + 1
G2	0.20	362.58	1 + 0	600.02	0 + 2
G3	0.15	462.39	1 + 1	493.50	1 + 1
G3	0.20	600.00	0 + 2	600.00	0 + 2
G4	0.15	116.79	2 + 0	589.58	1 + 1
G4	0.20	600.00	0 + 2	379.91	1 + 1
G5	0.15	600.00	0 + 2	600.28	0 + 2
G5	0.20	300.21	1 + 1	300.77	1 + 1
G6	0.15	300.49	1 + 1	301.51	1 + 1
G6	0.20	300.48	1 + 1	302.12	1 + 1
Average		428.58		472.38	
Sum			8 + 14		7 + 16

CHAPTER 5

The Resource Constrained Clustered Shortest Path Tree Problem

The *Resource Constrained Shortest Path Tree Problem* (RC-CluSPT) is an **NP**-hard problem that we recently proposed [78]: given a weighted, simple, undirected graph supplied with a resource function, and supposing that the set of nodes is partitioned in clusters, it aims at finding a shortest path tree respecting some resource consumption constraints and inducing a connected subgraph within each cluster. In particular, in Ferone et al. [78] we propose a mathematical model for this problem and devise a Branch & Price for its resolution.

In Section 5.1 we outline the motivations that led us to investigate the class of clustered problem along with a brief literature review (Section 5.2). The mathematical formulation of RC-CluSPT is presented in Section 5.3. Then, in Section 5.4 we thoroughly describe our algorithmic proposal [78]. Finally, the remaining of the Chapter is devoted to presenting results of the computational study conducted to appraise the performance of our proposal (Section 5.5).

5.1 Background and Motivations

The purpose of this Section is to provide a detailed representation of the context in which the RC-CluSPT is located and the reasoning for our study. The scientific community is devoting great efforts to the study of *clustered* – or *generalized* – versions of several classic optimization problems since they allow to address a wide variety of real-world issues [66]. In particular, in a network problem, the switch from the classic to the generalized version is obtained by partitioning the set of nodes of the underlying graph in a specific number K of

clusters. On the one hand, using such substructures, the aggregation phenomena occurring between similar entities, e.g. community of individuals, are efficiently depicted [62]. On the other hand, in this way several real-world problems could be addressed, e.g. the design of backbone networks in telecommunication or in metropolitan areas, the optimization of irrigation systems, the design of inter-cluster topology in computer and transportation networks [149, 164, 172].

Moreover, the class of clustered problems is attracting interest also from a theoretical point of view. In fact, the generalized version of a problem could become extremely harder to solve than the original one, since the feasibility conditions are expressed through constraints on clusters in addition to the "classic" constraints, i.e. those of the original problem [47].

Notably, the concept of generalization has been carried out in the formulation by following two different approaches, i.e. by requiring that:

a) either each feasible solution contains exactly/at least/at most one node from each cluster [66, 149],

b) or that the subgraphs induced by the solution within each cluster are connected [43, 172].

For this purpose, exploiting a generalization of type b), D'Emidio et al. [47] defined the *Clustered Shortest Path Tree Problem* (CluSPT) as the problem of determining a minimum cost shortest path tree on a clustered graph, such that the subgraphs induced within each cluster are connected and proved that this problem is **NP**-hard [62]. Given its inherent complexity and its applicability in the context of optimized communication networks and irrigation systems, the CluSPT has been addressed with different heuristic approaches. Binh et al. [21] and Thanh et al. [164] designed two Evolutionary Algorithms based on different encoding functions, Thanh et al. [165] proposed a heuristic based on the combination of randomized greedy method and shortest path tree algorithm; finally, Cosma et al. [43] devised an ad hoc Genetic Algorithm.

At the same time, the design of realistic optimized routes in transportation or telecommunication networks generally requires finding optimal paths accounting for assigned link attributes. As the resources represent a heterogeneous set of attributes, several real-world problems could be modeled and solved through Resource Constrained SPTs [107]. In Ferone et al. [78] we exploit the generalization of type b) to define a clustered version of the Resource Constrained Shortest Path Tree Problem (RC-SPT), namely the *Resource Constrained Clustered Shortest Path Tree Problem* (RC-CluSPT) with local resource constraints.

Though the RC-SPT has been widely studied, to the best of our knowledge, no generalization of this type has been proposed yet though it presents several fields of application. In fact, the RC-CluSPT lies at the intersection between the CluSPT and the RC-SPT. Accordingly, its possible contexts of application naturally comprise those of the classic problems [106]. For instance, generalized versions of specific communication network issues could be addressed through the RC-CluSPT, such as the SPT subject to bandwidth constraints and hop limited [177]. In this **NP**-hard problem, letting the bandwidth of a path be the minimum

available residual bandwidth at any edge along the path, the aim is to compute a minimum cost broadcast tree in which each path respects bandwidth constraints and a bound on the number of traversed edges. Indeed, it is crucial to address the *broadcast routing* problem in the design of different types of communication networks, where the *broadcasting* is a function that allows a message to be sent from a source node to all the other nodes in the network [45].

5.2 Literature Review

As pointed out in Section 5.1, the RC-CluSPT can be seen either as a refinement of the CluSPT or as a generalized version of the classic RC-SPT. Therefore, in this Section we provide: i) a brief literature review on generalized combinatorial optimization problems, focusing on both widely studied and recently proposed ones (Section 5.2.1); ii) an overview of Resource CSPPs (Section 5.2.2).

5.2.1 Brief Overview of Generalized Optimization Problems

The class of clustered problems comprises generalizations of well established problems such as: the *Generalized Traveling Salesman Problem* (GTSP), the *Generalized Minimum Spanning Tree Problem* (GMSTP), and the *Generalized Steiner Tree Problem* (GSteTP). The GTSP is one of the earliest formulated clustered problem [99, 161]; in particular, in scientific literature appeared contributions dealing with both the mentioned concepts of generalization (see Section 5.1). Just to cite a few, Guttmann-Beck et al. [96] addressed the problem of finding a minimum cost Hamiltonian tour so that the nodes of each cluster are visited consecutively. They proposed approximation algorithms for different variants of the problem, depending on whether or not the starting and ending nodes of a cluster have been specified. In particular, the approximation ratio of their algorithm for this latter variant of GTSP was later improved by Bao and Liu [12]. Instead, Fischetti et al. [84] proposed Integer Linear Programs for the Symmetric GTSPs in which the least cost solution cycle has to visit each cluster at least/exactly once. The same authors tackled these variants with a Branch&Cut algorithm [85]. More recently, Smith and Imeson [160] addressed the version of the Symmetric GTSP with exactly one visit for each cluster implementing a heuristic based on adaptive large neighbourhood search. However, it is worthy to note that the classic Traveling Salesman Problem is an **NP**-hard problem; thus all the possible GTSPs are themselves **NP**-hard.

The GMSTP was firstly formulated by Myung et al. [136] as the problem of finding a minimum cost tree spanning a subset of nodes with exactly one node from each cluster (EGMSTP); later, Ihler et al. [104] defined the version in which at least a node from each cluster has to be included in the spanning tree, namely LGMSTP. However, in contrast with the classic counterpart – which is polynomially solvable – both these variants have been proved to be **NP**-hard. Though, D'Emidio et al. [62] pointed out that the GMSTP in which the solution tree is required to induce a connected subgraph in each cluster remains

polynomially solvable. In literature, (meta-)heuristic and exact approaches have been proposed for the above mentioned GMSTPs: for example, Öncan et al. [143] devised a Tabu Search algorithm for EGMSTP and adapted it to the resolution of LGMSTP; Hu et al. [101] tackled the EGMSTP with a Variable Neighbourhood Search heuristic combining three different neighbourhood types; finally, Ferreira et al. [79] found that, independently from the construction heuristic, a GRASP algorithm for EGMSTP performed the best when employing path relinking and Iterated Local Search. As regards the exact approaches, Feremans [65] designed a Branch&Cut algorithm for EGMSTP; instead, Pop [149] proposed a Dynamic Programming algorithm for EGMSTP.

As regards the GSteTP, Reich and Widmayer [153] defined a first version in which, supposing that the subset R of *required nodes* is divided in K clusters, the aim is to find a minimum cost tree of the graph containing at least a node from each cluster. Ihler [103] tackled this GSteTP with a $(K-1)$-approximation algorithm, while Yang and Gillard [176] computed a lower bound on the problem using Lagrangean relaxation and sub-gradient optimization. Exploiting the idea of generalization in terms of cluster connection, Wu and Lin [172] recently defined a variant of GSteTP on metric graphs: this problem aims at finding a minimum cost Steiner tree inducing mutually disjoint minimal spanning trees in the clusters of R. The authors tackled this problem with a $(2+\rho)$-approximation algorithm, where ρ is the best known approximation ratio for the Minimum Steiner Tree Problem (MSteTP). Trivially, these GSteTPs are **NP**-hard since they generalize an **NP**-hard problem; however, unlike the classic MSteTP, they remain computationally hard problems even when the input graph is a tree or there are no Steiner nodes [176].

As thoroughly described in Feremans et al. [66], among the other classic problems whose generalization has been formulated with the exactly/at least/at most approach, there are: the *Vehicle Routing Problem* [91], the *Minimum Clique Problem* [117], the *Shortest Path Tree Problem* [125].

To conclude, we mention the *Minimum Routing Cost Clustered Tree Problem* in which the generalization is formulated in terms of cluster connection. Lin and Wu [127] recently proposed this problem which aims at finding a clustered spanning tree with minimum routing cost that is the total distance summed over all pairs of vertices. The authors proved the **NP**-hardness of the problem, when there are more than two clusters, and proposed an Approximation Algorithm.

5.2.2 Brief Overview of Resource CSPPs

The Resource Constrained Shortest Path Problem (Resource CSPP) was formulated by Desrochers [49] as a sub-problem of a Bus Driver Scheduling problem. Since then, the investigation of Resource CSPPs has represented a flourishing field of research; in particular, different variants of the problem have been addressed for two main reasons: the resource constraints are versatile, thus they allow to depict a heterogeneous variety of real-world problems, and the formulation of these constraints depends itself on the nature of the attributes represented through the resources [107]. At this purpose, Avella et al. [10]

classified these attributes in three categories and presented the different formulations of the corresponding constraints. According to them, resources can represent attributes that are: *numerical* and *cumulative* (e.g. travel time, fuel consumption); *numerical* and *non-cumulative* (e.g. road width, height); *indexed* or *categorical* (e.g., parking restrictions, type of roads).

Moreover, the **NP**-hardness of the Resource CSPP regardless of the type and the number of resources [76] motivates the continuous development of new solution approaches, both exact and heuristic. Specifically, the proposed exact methods can be categorized as follows: Path Ranking strategies, which find the first $\mathcal{K}$ shortest paths and select among these the one with the least resource consumption; Dynamic Programming approaches; Polyhedral and Branch&Cut approaches. Just to mention few recent contributions, Sedeño-Noda and Alonso-Rodríguez [159] devised an algorithm which encompasses the binary partition strategy adopted in the $\mathcal{K}$-SP Path Ranking algorithm [146] and several pruning strategies, obtaining a sensible speedup on large instances, compared to the state of the art. Righini and Salani [157] improved the classic Dynamic Programming approach for RC-SPP [50] through bidirectional search with resource-based bounding; moreover, the authors adopted the Dynamic Programming paradigm to compute lower bounds in a Branch & Bound algorithm and proposed the well-performing *Decremental State-Space Relaxation*, of which the two previously mentioned algorithms are special cases. Finally, Horváth and Kis [100] recently generalized and extended some valid inequality devised by Garcia [89] for a Branch & Bound approach to solve Resource CSPP. Moreover, a heuristic procedure to obtain an initial feasible solution at each node of the branching tree and a variable fixing techniques are presented in this paper.

Instead, among the recently proposed heuristic approaches, Dong et al. [59] devised an Ant Colony algorithm to solve a particular Resource CSPP arising from a Multi-modal Transport Problem featuring time windows and constraints on the overall number of transshipment. To solve the Resource CSPP, Marinakis et al. [130] proposed a hybridized version of a Particle Swarm Optimization method featuring a Variable Neighbourhood Search and two different Expanding Neighbourhood topologies. Specifically, different local search strategies were tested in their computational experiments to detect the most effective one.

Finally, it is worthwhile mentioning that Resource CSPP often appears as Pricing Problem in Column Generation, e.g. for the Vehicle Routing Problem [6] and the Crew Scheduling Problem [35]; consequently, its resolution has been further refined in order to optimize this type of procedure. At this purpose, Zhu and Wilhelm [183] devised a pseudo-polynomial three-stage solution approach which performs the first two stages, namely pre-processing and setup, only once to transform Resource CSPP in a Shortest Path Problem; then, the last stage solves this problem at each iteration of Column Generation.

5.3 Mathematical Formulation

In this Section we report the mathematical formulation of the RC-CluSPT, as presented in Ferone et al. [78]. At this purpose, the graph structure is detailed before presenting the *path-based formulation* in Section 5.3.1.

Let $G = (V, E, c, r)$ be a simple undirected graph where $c : E \longrightarrow \mathbb{R}^+$ is a non-negative cost function, with $c(i, j) := c_{ij}$, and $r : E \longrightarrow \mathbb{R}^+$ is a *resource consumption* function representing *numerical cumulative attributes* [10], with $r(i, j) := r_{ij}$. Consider a collection $\{C_k\}_{k=1\ldots K}$ of node-sets that partition V, referred to as *clusters*, and a source node $s \in V$; then, the *Resource Constrained Clustered Shortest Path Tree Problem* (RC-CluSPT) aims to find a minimum-cost tree T_s in G rooted in s, such that:

i) T_s spans all the nodes of the graph G;

ii) for each cluster C_k the subgraph induced by T_s is connected;

iii) every path in T_s respects a maximum resource consumption constraint.

In particular, the set E can be partitioned in two subsets: the set of edges connecting nodes within the same cluster, called *intra-cluster edges*, and the set of edges connecting nodes belonging to two distinct clusters i.e. the *inter-cluster edges*. In other words, given $[i, j] \in E$, it is an intra-cluster edge if both i and j belong to the same cluster C_k; otherwise, if $i \in C_k$ and $j \in C_h$ with $k \neq h$, then $[i, j]$ is an inter-cluster edge.

In the following, we will denote with E_k the set of intra-cluster edges of cluster C_k, $k = 1 \ldots K$. Moreover, for any node-set $S \subseteq V$, we will denote with $E(S) \subseteq E$ the edge-set of the subgraph induced by S.

Finally, let n be the cardinality of V and n_k be the cardinality of the cluster C_k, $k = 1, \ldots K$; clearly, it holds that $\sum_{k \leq K} n_k = n$.

5.3.1 Path-based Formulation

The RC-CluSPT can be formulated as a *multi-commodity flow problem*. At this purpose, we will use the directed graph $D = (V, A, c, r)$ underlying G, in which $A = \left\{(i, j), (j, i) \colon [i, j] \in E\right\}$; i.e. for each edge $[i, j] \in E$, we define both the arc (i, j) and the corresponding anti-parallel arc (j, i) in the arc-set of the graph D. Moreover, c and r extend the corresponding cost and resource consumption functions defined on G, with $c_{ij} = c_{ji}$ and $r_{ij} = r_{ji}$. Then, the path-based formulation relies on the definition of two types of variables:

1. non-negative continuous flow variables f_{hij}, defined $\forall h \in V \setminus \{s\}$ and $\forall (i, j) \in A$, which assume a positive value if the arc (i, j) is used in the path from s to h in T_s;

2. Boolean decision variables x_{ij} defined $\forall [i, j] \in E$, such that $x_{ij} = 1$ if $[i, j] \in T_s$.

To describe the particular tree structure of a generic feasible solution (conditions (i)-(ii)), it is necessary to state the following constraints:

- the *activation* constraints (Equation (5.1)) which couple the f variables with the corresponding x ones:

$$f_{hij} + f_{hji} \leq x_{ij}, \quad \forall [i,j] \in E, h \in V \setminus \{s\} \tag{5.1}$$

- the *flow balancing* constraints given in Equation (5.2):

$$\sum_{j \in FS(i)} f_{hij} - \sum_{j \in BS(i)} f_{hji} = \begin{cases} 1 & \text{if } i = s, \\ -1 & \text{if } i = h, \\ 0, & \text{otherwise.} \end{cases} \quad \forall i, h \in V,\, h \neq s \tag{5.2}$$

- the *dimension* constraints, for the whole solution (Equation (5.3a)) as well as for each cluster C_k (Equation (5.3b)):

$$\sum_{[i,j] \in E} x_{ij} = n - 1, \tag{5.3a}$$

$$\sum_{[i,j] \in E_k} x_{ij} = n_k - 1, \quad \forall k = 1, \dots K. \tag{5.3b}$$

In particular, constraints (5.3a)-(5.3b) yield that each feasible solution should contain $n - K$ intra-cluster arcs and $K - 1$ inter-cluster arcs, connecting clusters so that there is no isolated cluster, otherwise constraints (5.2) would be violated. Moreover, condition (ii) for a generic solution holds because of the combination of constraints (5.2) and (5.3b).

Remark 10. It is worth to note that, for any feasible solution, condition (ii) could be guaranteed replacing constraints (5.3b) with the *acyclicity* constraints (5.4) within each cluster:

$$\sum_{[i,j] \in E(S)} x_{ij} \leq |S| - 1, \quad \forall S \subseteq C_k, |S| \geq 2. \tag{5.4}$$

In fact, in each feasible solution T'_s, condition (5.4) has to hold as an equality when $S = C_k$, for each $k = 1, \dots, K$ otherwise there would exist a cluster with an isolated vertex. Consequently, also constraints (5.4) state that each feasible solution should contain $n - K$ intra-cluster arcs. □

Finally, the resource constraints can be added to each single path, i.e. the single commodity flows (cfr. Equation (5.5f)).

The model for the RC-CluSPT is given in (5.5), where the objective function (5.5a) minimizes the cost of the resulting tree.

$$\text{(RC-CluSPT)} \ \min \sum_{h \in V \setminus \{s\}} \sum_{(i,j) \in A} c_{ij} f_{hij} \tag{5.5a}$$

subject to:

$$f_{hij} + f_{hji} \leq x_{ij}, \qquad \forall [i,j] \in E, h \in V \setminus \{s\} \quad (5.5b)$$

$$\sum_{j \in FS(i)} f_{hij} - \sum_{j \in BS(i)} f_{hji} = \begin{cases} 1 & \text{if } i = s, \\ -1 & \text{if } i = h, \\ 0, & \text{otherwise.} \end{cases} \qquad \forall i, h \in V,\ h \neq s \quad (5.5c)$$

$$\sum_{[i,j] \in E} x_{ij} = n - 1, \quad (5.5d)$$

$$\sum_{[i,j] \in E_k} x_{ij} = n_k - 1, \qquad \forall k = 1, \dots K \quad (5.5e)$$

$$\sum_{(i,j) \in A} r_{ij} f_{hij} \leq R_h, \qquad \forall h \in V \setminus \{s\} \quad (5.5f)$$

$$x_{ij} \in \{0,1\}, \qquad \forall [i,j] \in E \quad (5.5g)$$

$$f_{hij} \geq 0, \qquad \forall (i,j) \in A, h \in V \setminus \{s\}. \quad (5.5h)$$

Remark 11. It is widely known that the classic RC-SPT is computationally intractable, namely it is an **NP**-hard problem even in case of a single resource function [76]. As regards RC-CluSPT, it is **NP**-hard too since otherwise, letting the number of clusters be $K = 1$, the RC-SPT would be tractable. □

5.4 Solution Approach

The aim of this Section is twofold: on the one hand, we point out the motivations that led us to address the RC-CluSPT with a Branch & Price approach and briefly outline its structure. On the other hand, we detail how the components of this algorithmic framework are characterized for the problem under consideration.

The idea of tackling RC-CluSPT with a Branch & Price method comes from the observations that: i) the mathematical model (5.5) comprises a huge number of variables, and ii) its coefficient matrix presents a *double-bordered block diagonal form* [118]. Thus, it is possible to apply a *Dantzig-Wolfe (DW) decomposition* [46] to (5.5) to efficiently solve the resulting problem with Branch & Price.

Indeed, Branch & Price is an algorithmic paradigm which integrates the procedure of Branch & Bound (see Section 4.3) with *Column Generation* (CG) to solve large-scale Integer Programming problems. Specifically, it consists in solving a restricted version of the linear relaxation of the original problem, namely the *Restricted Master Problem* (RMP), which comprises only a subset of variables [48]. Then, solving the so-called *Pricing Problem(s)* either the RMP solution is proved to be optimal also for the linear relaxation, or columns (and variables) to add to the RMP are identified. In particular, the branching occurs when the optimal solution of the linear relaxation does not satisfy integrality conditions.

The Branch & Price algorithm devised in Ferone et al. [78] entails that

Column Generation (CG) is adopted to solve the linear relaxations in each node of the search tree, involving a *Multiple Pricing* scheme.

The DW decomposition of (5.5), the resulting Pricing Problems and the strategy adopted to solve them are described in Sections 5.4.1 to 5.4.3.

5.4.1 Dantzig-Wolfe Decomposition

The DW decomposition of the model (5.5) follows from the observation that, for each node $h \in V \setminus \{s\}$, the constraints (5.5c), (5.5f) and (5.5h) describe the feasible region of a Resource CSPP from s to h in D, while constraints (5.5b) are the *linking constraints*. Thus, the variables f_{hij} can be decomposed using variables representing the paths from s to h in D which respect the resource constraints.

Formally, given a node $h \in V \setminus \{s\}$, let P_h be the set of all the possible paths in D from s to h which respect the corresponding resource constraint (5.5f), and define a continuous non-negative variable λ_h^p for every path $p \in P_h$. Then, the flow along each arc $(i, j) \in D$ in a path from s to h, is the convex combination of flows along all the possible paths from s to h in P_h (see Equation (5.6)):

$$f_{hij} = \sum_{p \in P_h} \lambda_h^p f_{hij}^p. \tag{5.6}$$

Thus, for each node $h \in V \setminus \{s\}$, the cost of the path from s to h in the solution is a convex combination of the costs c_h^p of the paths $p \in P_h$ (see Equation (5.7)).

$$\sum_{(i,j)\in A} c_{ij} \sum_{p\in P_h} \lambda_h^p f_{hij}^p = \sum_{(i,j)\in A} \sum_{p\in P_h} c_{ij} \lambda_h^p f_{hij}^p = \sum_{p\in P_h} \lambda_h^p \underbrace{\sum_{(i,j)\in A} c_{ij} f_{hij}^p}_{c_h^p} = \sum_{p\in P_h} c_h^p \lambda_h^p. \tag{5.7}$$

Then, the DW decomposition (5.8) of (5.5) is obtained by: replacing the flow variables with the expression in (5.6), and reformulating constraints (5.5b) and the objective function (5.5a) with (5.7). The resulting problem is the so-called *Master Problem*.

$$\text{(DW-MP)} \ \min \sum_{h\in V\setminus\{s\}} \sum_{p\in P_h} c_h^p \lambda_h^p \tag{5.8a}$$

subject to:

$$\sum_{p\in P_h} (f_{hij}^p + f_{hji}^p) \lambda_h^p \le x_{ij}, \qquad \forall [i,j] \in E, \forall h \in V \setminus \{s\} \tag{5.8b}$$

$$\sum_{[i,j]\in E} x_{ij} = n - 1, \tag{5.8c}$$

$$\sum_{[i,j]\in E_k} x_{ij} = n_k - 1, \qquad \forall k = 1, \dots K \tag{5.8d}$$

$$\sum_{p \in P_h} \lambda_h^p = 1, \qquad \forall h \in V \setminus \{s\} \tag{5.8e}$$

$$x_{ij} \in \{0,1\}, \qquad \forall [i,j] \in E \tag{5.8f}$$

$$\lambda_h^p \geq 0, \qquad \forall p \in P_h, \forall h \in V \setminus \{s\}. \tag{5.8g}$$

The objective function (5.8a) minimizes the cost of the resulting tree; Equation (5.8b) represent the activation constraints while constraints (5.8c) and (5.8d) are analogous to the corresponding (5.5d) and (5.5e). Finally, (5.8e) are the *convexity* constraints for the continuous non-negative path variables, forcing the selection of exactly one path in P_h for each $h \in V \setminus \{s\}$.

5.4.2 Pricing Problem Formulation

As previously mentioned, each node of the branching tree is solved by applying a Column Generation with a Multiple Pricing Scheme. Specifically, this means that at each iteration of the CG, it is required to solve $n-1$ Pricing Problems, one for each node $h \in V \setminus \{s\}$ in order to find the columns to add to the Restricted Master Problem. Indeed, as proposed in Tilk and Irnich [167], we consider a linear relaxation of DW-MP, removing the binary constraints (5.8f). Moreover, the RMP is obtained by considering a subset of path variables.

The mathematical formulation of the Pricing Problem corresponding to a generic node $h \in V \setminus \{s\}$ is obtained from the dual formulation (5.9) of DW-MP.

$$\max \left[\sum_{h \neq s} \eta_0^h + (n-1)\mu_2 + \sum_{i \leq K} (n_i - 1)\nu_i \right] \tag{5.9a}$$

subject to:

$$\sum_{[i,j] \in E} (f_{hij}^p + f_{hji}^p)\pi_{hij} + \eta_0^h \leq c_h^p, \qquad \forall p \in P_h, \forall h \in V \setminus \{s\} \tag{5.9b}$$

$$\pi_{hij} \leq 0, \qquad \forall [i,j] \in E, \forall h \in V \setminus \{s\}. \tag{5.9c}$$

In particular, for each edge $[i,j] \in E$, the negative dual variables π_{hij} are associated to the corresponding constraints (5.8b); then, variable $\eta_0^h \in \mathbb{R}$ corresponds to the convexity constraints (5.8e). Finally, variables $\nu_i \in \mathbb{R}$ are related to constraints (5.8c) and variable $\mu_2 \in \mathbb{R}$ corresponds to constraint (5.8d).

Since the reduced costs are described by Equation (5.9b), recalling Equation (5.7), we obtain the formulation (5.10) of the Pricing Problem to solve for each node $h \in V \setminus \{s\}$. It is duly noted that this is a Resource Constrained Shortest Path Problem from s to h in D, with non-negative costs.

$$\text{(PP\_h)} \ \min \left[\Big(\sum_{(i,j) \in A} (c_{ij} - \pi_{hij}) f_{hij} \Big) - \eta_0^h \right] \tag{5.10a}$$

subject to:

$$\sum_{j \in FS(i)} f_{hij} - \sum_{j \in BS(i)} f_{hji} = \begin{cases} 1 & \text{if } i = s, \\ -1 & \text{if } i = h, \\ 0, & \text{otherwise.} \end{cases} \qquad \forall i \in V, \qquad (5.10b)$$

$$\sum_{(i,j) \in A} r_{ij} f_{hij} \leq R_h, \qquad (5.10c)$$

$$f_{hij} \geq 0, \qquad \forall (i,j) \in A. \qquad (5.10d)$$

5.4.3 Solving the Pricing Problem

As noted in Section 5.4.2, the block diagonal structure of the RC-CluSPT leads to $n-1$ Pricing Problems, each of which consists in the solution of a Resource CSPP from s to $h \in V \setminus \{s\}$ in D.

At this purpose, we resort to a Dynamic Programming (DP) algorithm, given its successful application in the solution of CSPPs [53, 75, 157]. According to this solution paradigm, the possible paths starting from s are associated to labels, recording both their cost and resource consumption. Doing so, the DP explores the solution space evaluating path labels via the concepts of *feasibility* and *dominance* (cfr. Section 4.3.2).

In this context, denoted with $R(p)$ the total resource consumption along a generic path p, given two paths p_{si} and $\hat{p}_{si}$ from s to $i \in V \setminus \{s\}$, path p_{si} is said to be *dominated* by $\hat{p}_{si}$ if the following conditions hold: (i) $c(\hat{p}_{si}) \leq c(p_{si})$; (ii) $R(\hat{p}_{si}) \leq R(p_{si})$, and at least one of such conditions is strict. Consequently, throughout the exploration process of DP, only feasible non-dominated paths can be held as candidate for extensions towards the target node h.

5.5 Computational Experiments

This Section reports the computational experiments and the related discussion we presented in Ferone et al. [78]: we have designed this testing to appraise the performance of the proposed Branch & Price algorithm (BP), and have compared it with the results achieved by the direct solution of the mathematical model (5.5) performed with the ILOG CPLEX 12.9 solver.

More specifically, Section 5.5.1 presents the data-set used in the numerical experiments, Section 5.5.2 discusses the direct comparison of the proposed methodology and the CPLEX solver, while Section 5.5.3 delves into the analysis of the results of BP when related to instance properties.

The compared approaches have been coded in C++ using the flags `-std=c++17 -O3` and compiled with `g++ 8.2`. The experiments were run on an INTEL i9-9900X@3.5 GHz processor with 4GB of RAM. A time limit of 1800 seconds has been used for both solution methods.

5.5.1 Test Problems

The network topologies featured in the computational experiments designed in Ferone et al. [78] are the small instances of Types 1, 5 and 6 used by Thanh et al. [166] for the Clustered Shortest Path Tree Problem.

In particular, as described in Mestria [133]: Type 1 networks are obtained from the TSPLIB benchmark, using a k-means algorithm to define the clusters; Type 5 instances derive from those used in the literature of the TSP, clustered by grouping the vertices in geometric centres; and, lastly, graphs of Type 6 are adapted from the TSPLIB library, by partitioning the networks into quadrilaterals, each of which corresponds to a cluster. Using these topologies, a set of feasible instances for the RC-CluSPT is generated through the procedure detailed below.

Instance Generation

Given a generic clustered undirected graph $G = (V, E)$, a resource consumption function $r : E \longrightarrow \mathbb{R}^+$ satisfying the triangle inequality can be associated to the edges of G according to the following steps:

1. randomly assign temporary resources $r_{tmp,e}$ to each $e \in E$;
2. solve an all-pairs shortest path tree problem using $r_{tmp,e}$ as costs;
3. impose r_{ij} equal to the cost of the shortest path from i to j found in step 2.

Once the non-negative resource consumption function $r : E \longrightarrow \mathbb{R}^+$ is determined, a RC-CluSPT instance is specified through its resource constraints.

Let T_s^* be the optimal solution of the classic CluSPT with r as cost function, and let $p_{s,h}^*$ be the path of T_s^* connecting the source s and the node $h \in V \setminus \{s\}$. Moreover, let $R(p_{s,h}^*)$ indicate its cost in T_s^*, i.e. the total resource consumption of $p_{s,h}^*$. Then, using a generation scheme similar the one described in Dumitrescu and Boland [61], it is possible to define three sub-classes characterized by: *low-resource* (Low-R), *medium-resource* (Medium-R), and *high-resource* (High-R), by considering convex combinations of $R(p_{s,h}^*)$ and $2\,R(p_{s,h}^*)$.

$$\text{(Low-R)} \qquad R_h = (1-\alpha)R(p_{s,h}^*) + \alpha 2\,R(p_{s,h}^*), \qquad \forall h \in V \setminus \{s\} \tag{5.11}$$

$$\text{(Medium-R)} \qquad R_h = 0.5\,R(p_{s,h}^*) + R(p_{s,h}^*), \qquad \forall h \in V \setminus \{s\} \tag{5.12}$$

$$\text{(High-R)} \qquad R_h = \alpha R(p_{s,h}^*) + (1-\alpha)2\,R(p_{s,h}^*). \qquad \forall h \in V \setminus \{s\} \tag{5.13}$$

As Dumitrescu and Boland [61], we used $\alpha = 0.05$ for all graph topologies. It is duly noted how in all three cases T_s^* is a feasible (possibly sub-optimal) solution. As a result of this instance generation process we obtained 255 feasible instances for the RC-CluSPT, whose characterizing features are summarized in Table 5.1.[1]

[1]The full data-set can be found at `https://figshare.com/s/7ef8c99aedc027b208c9`.

Table 5.1 Summary of instances properties.

Type	Number	Nodes	Edges	Clusters
1	81	51–105	1275–5460	5–75
5	63	30–120	435–7140	5–10
6	111	51–105	1275–5460	2–42

5.5.2 Comparison of Algorithms

The first set of computational analyses is devoted to the comparison of BP with the performance achieved by CPLEX on the entire data-set.

Table 5.2 summarizes the comparative numerical analysis, averaging the results according to three distinct categories: instances solved by both algorithms ("avg both"); instances solved by the corresponding method, i.e. all the instances solved by BP (resp. CPLEX) in the corresponding column ("avg single"); and all instances ("avg all"). For each of such categories, Table 5.2 reports the average objective function value ("Cost"), the total time and time to best ("TtB"), and the optimality gap, computed as

$$GAP = \frac{\hat{C} - LB}{LB}, \tag{5.14}$$

where $\hat{C}$ is the cost of the incumbent solution, and LB is the best lower bound found by the algorithm. Since BP and CPLEX derive lower bounds using different strategies, different solutions may yield the same optimality gap. Finally, the last two rows of Table 5.2 indicate, for each method, the number of instances for which at least a feasible solution has been found, and the number of instances solved to optimality.

Table 5.2 BP vs CPLEX on the entire data-set.

	BP				CPLEX			
	Cost	**Total time**	**GAP**	**TtB**	**Cost**	**Total time**	**GAP**	**TtB**
avg both	130690.33	126.46	0.01	85.29	130729.76	522.85	0.01	332.39
avg single	133472.00	180.65	0.02	133.63	133693.89	589.25	0.02	369.93
avg all		408.15				869.31		
Feasible	221/255				193/255			
Optimal	208/255				139/255			

Observing the results of Table 5.2 it emerges that the proposed Branch & Price found at least feasible solutions for approximately 87% of the instances (221/255); in comparison CPLEX was able to find solutions to 76% of the data-set (193/255). Moreover, when taking into account the number of optimal solutions found, the edge of the proposed algorithm with respect to CPLEX is evident, with a total of 208 instances solved to optimality against 139.

The difference in terms of the number of optimal solutions found is also reflected in the average solution cost in the category *avg both*, that evidences how BP achieves objective function values lower than those obtained by CPLEX.

This first analysis of the quality characterizing the solutions achieved by both approaches evidences that the proposed Branch & Price significantly improves the performance of CPLEX. However, this behaviour is not highlighted by the GAP values, that on average are approximately the same for both algorithms; actually, this outcome can be related to the different strategies used to derive lower bounds.

Concerning the computation times, BP employs approximately 25% of the time needed by CPLEX on instances solved by both methods (126.46s against 522.85s) and around 47% on all instances (408.15s vs 869.31s). As a final remark it is possible to observe that on average the ratios of the time to best with respect to the total computation times are comparable for both methods, and are within the interval $[0.63, 0.74]$.

5.5.3 Sensitivity Analysis

In the second set of analyses conducted in Ferone et al. [78] we evaluated the performance of the proposed Branch & Price relating the results to the type and distinctive features of the instances considered.

Table 5.3 reports the averages of the total times of computations and objective function values achieved, dividing the benchmark according to network types (Type 1, Type 5, and Type 6) and resource allowance (namely, Low-R, Medium-R, High-R). Analogously, Table 5.4 for each group reports the fraction of instances for which at least a feasible solution has been found, and the instances that were solved to optimality. Lastly, Table 5.5 collects a detailed description of the behaviour of the proposed solution approach, reporting, for each group, the time spent solving reduced RMP ("MP time") and pricing sub-problems ("PR time"), the number of total RMPs solved ("#MP") and call to the pricing procedures ("#PR"), the number of columns added ("#Cols"), and the total number of branching nodes explored ("#Bnodes").

Analyzing the summaries of Table 5.3 and Table 5.4, it is evident how a lower resource allowance is related to reduced computational times and thus a higher number of optimal solutions found. This particular behaviour can be justified observing that a lower resource allowance leads to a smaller solution space, that allows BP to prove optimality by just exploring an extremely limited number of branching nodes. Table 5.5 confirms this evidence, reporting that, for the totality of instances characterized by a low resource allowance, a single branching node is sufficient to prove optimality. As a direct consequence, this behaviour is also reflected in lower number of calls to the pricing sub-problems and the number of columns added to the RMPs.

Moreover, as the resource allowance grows, a set of more permissive resource constraints allows the algorithm to explore a wider range of solutions, thus yielding average costs that are generally lower. This is of course expected, since a feasible solution for low resource allowance is still feasible in the case of medium

and high allowance. At times, though, even in presence of higher resource allowance it is possible to note higher average costs: see, for example, Type 1 Low-R and High-R instances in Table 5.3. In such cases, the average cost values are affected by the lower number of optimal solutions found in High-R instances, that leads to sub-optimal solutions possibly worse than the optima of Low-R instances of the same networks.

This last observation suggests that: (i) given the lower computational times implied by the solution of instances with low resource allowance, and (ii) given that for a fixed network topology the feasible region of Low-R instances is a subset of Medium-R and High-R instances, the solution of a surrogate problem with lower resource allowance could be used to yield good-quality feasible solutions in the case of the harder High-R instances.

Lastly, Instances of Type 5 appear to be significantly easier to solve than Type 1 and 6. This observation is equally reflected in the average computational time required and the number of instances solved to optimality. This behaviour can be related to the lower number of clusters that characterizes Type 5 instances, as reported in Table 5.1.

Table 5.3 Average times and cost values. Instances grouped according to network types and resource allowance.

	Low-R		Medium-R		High-R		Avg	
Type	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
Type_1	152065.46	8.16	147964.70	298.76	161223.54	924.06	152836.02	410.33
Type_5	63391.99	4.53	62600.26	20.48	57475.25	709.76	61405.36	237.43
Type_6	170951.92	6.63	170529.27	276.51	141784.58	1130.74	164824.85	452.98
Avg	138379.06	6.60	136425.37	220.32	120977.10	957.90	133472.00	385.97

Table 5.4 Fraction feasible/optimal solutions found. Instances grouped according to network types and resource allowance.

	Low-R		Medium-R		High-R		Total	
	Feasible	**Optimal**	**Feasible**	**Optimal**	**Feasible**	**Optimal**	**Feasible**	**Optimal**
Type_1	1	1	0.92	0.88	0.62	0.51	0.85	0.80
Type_5	1	1	1.00	1.00	0.80	0.71	0.93	0.90
Type_6	1	1	1.00	0.91	0.51	0.40	0.83	0.77
Total	1	1	0.97	0.92	0.62	0.51	0.86	0.81

Table 5.5 Detailed behaviour of the proposed Branch & Price approach.

Type	Resource	Time	MP time		#MP	PR time		#PR	#Cols	#Bnodes
Type_1	Low-R	8.16	4.66	(%57.15)	38.11	0.93	(%11.45)	3230.59	751.77	1.00
	Medium-R	298.76	172.63	(%57.78)	374.26	18.37	(%6.15)	35098.26	3869.37	65.89
	High-R	924.06	728.36	(%78.82)	978.59	20.82	(%2.25)	69528.33	11153.70	188.85
Type_5	Low-R	4.53	2.49	(%55.05)	29.48	0.51	(%11.21)	2079.57	545.29	1.00
	Medium-R	20.48	12.43	(%60.70)	118.81	3.28	(%16.02)	8796.43	2163.71	1.00
	High-R	709.76	482.25	(%67.94)	3645.85	70.46	(%9.93)	164114.40	20703.35	415.50
Type_6	Low-R	6.63	3.71	(%56.02)	36.30	0.75	(%11.28)	2864.41	699.65	1.00
	Medium-R	276.51	112.51	(%40.69)	586.89	25.88	(%9.36)	48973.46	4407.84	35.76
	High-R	1130.74	815.22	(%72.10)	1528.79	47.47	(%4.20)	107163.82	16456.26	136.71
Average		375.52	259.36	60.69	815.23	20.94	9.09	49094.36	6750.11	94.08

5.6 Conclusions

In this Chapter we addressed the *Resource Constrained Clustered Shortest Path Tree Problem* (RC-CluSPT), which we recently proposed in Ferone et al. [78]. In particular, this problem lies at the intersection between the Clustered Shortest Path Problem and Resource Constrained Shortest Path Tree Problem (RC-SPT). Indeed, being the clustered version of the RC-SPT, also the RC-CluSPT is **NP**-hard.

In Ferone et al. [78] we firstly proposed a mathematical model for the RC-CluSPT and then, due to its block-diagonal structure, we devised a Dantzig-Wolfe decomposition. The problem is tackled with a Branch & Price method, featuring a Column Generation approach with Multiple Pricing Scheme. The performance of this method are compared with those attained by CPLEX solver in the solution of the mathematical model, on a data-set obtained by adapting instances of the Clustered SPT from literature.

As reported here, the results evidence that the proposed Branch & Price significantly improves the performance of CPLEX, solving to optimality a greater number of instances and taking less than half the computation time of CPLEX. Moreover, the performed sensitivity analysis reveals that a lower resource allowance implies reduced computational times and thus a higher number of optimal solutions found, namely a single branching node is sufficient to prove optimality. Moreover, from the collected data it is possible to observe that the instances characterized by a lower number of clusters are easier to solve for the Branch & Price.

Due to the computational intractability of the RC-CluSPT, as further investigation we will exploit meta-heuristic techniques to obtain sub-optimal solutions of good quality in a reasonable computation time. Moreover, since only the case of local resources has been addressed in Ferone et al. [78], as a future line of research we plan to formalize the RC-CluSPT with global resource constraints, and to develop an ad hoc solution approach.

Part II

A Scheduling Problem in 3D Printing

CHAPTER 6

The Additive Manufacturing Machine Scheduling Problem

The mathematical model of the *Additive Manufacturing Machine Scheduling Problem* (AMM-SP) was recently proposed in Kucukkoc [119]. However, due to the intractability of AMM-SP, it cannot be employed to solve instances of relevant size. At this purpose, in Alicastro et al. [3] we devised a *Reinforcement Learning Iterated Local Search* meta-heuristic, based on the implementation of a *Q-Learning Variable Neighbourhood Search*, to provide heuristically good solutions in reasonable computational times.

After an introduction about additive manufacturing process planning (Section 6.1) and a comparison between the AMM-SP and a related machine scheduling problem (Section 6.2), the mathematical model for AMM-SP is given in Section 6.3. Then, Section 6.4 is fully devoted to an in-depth description of the proposed approach, while the computational experiments and their results are described in Section 6.5. Lastly, in the Appendix 6.A, a numerical example illustrates the key procedures of the proposed solution approach.

6.1 Overview of Additive Manufacturing Process Planning

Additive Manufacturing (AM) is defined by the American Society for Testing and Materials as the "process of joining materials to make parts from 3D model data, usually layer upon layer, as opposed to subtractive manufacturing and formative manufacturing methodologies".

This technology is gaining ground in the industry and experiencing continuous breakthrough because of high purchasing, affordable maintenance, and low processing costs of AM machines [57]. However, a general optimization problem

for the planning of AM processes can be represented coarsely by means of three interacting components: orientation of parts and their packing in *jobs* – referred to jointly as *nesting* – and *scheduling* of operations.

As thoroughly surveyed by Oh et al. [142], the scientific contributions proposed in this field all resort to slightly different levels of detail for each one of such three interacting components. For example, in the context of determining location and orientation of parts, the maximization of nesting rate has been addressed. At this purpose, considering irregular shaped parts with due dates, Wang et al. [171] proposed a computer vision-based approach to tackle efficiently the 2D Packing Problem which arises once that parts have been sorted according to heights, areas and time remained before deadline. Analysing the nesting issues from a 3D perspective, Wu et al. [174] devised a Genetic Algorithm to address the problem of packing several irregular shaped parts in a single job such that the total height, surface roughness, and support volume are minimized. Finally, both the orientation of job and the related Packing Problem were addressed in [95]. The authors considered 2D irregular shaped parts and aimed at minimizing the total construction time of jobs through a six-stages procedure, combining a Tabu Search to solve build orientation and a two-stage procedure for the 2D Packing Problem. On the other hand, focusing on the production planning of parts with different shapes and geometries, Oh et al. [141] optimized the operations to be performed on multiple AM machines. The authors proposed a heuristic algorithm to solve the related packing problem and analyzed two different build orientations for the parts: *standing policy* and *laying policy*, showing how the former outperforms the latter when the number of parts increases. Accounting for parts with different due dates, Chergui et al. [38] studied the minimization of the total *tardiness* in case of parallel identical AM machines. Lastly, Li et al. [123, 124] tackled the problem of dynamic order acceptance scheduling, modelling a scenario in which a series of part orders arrives dynamically at a production facility, where the on-demand production has to be optimized to maximize the average profit-per-unit-time.

The AMM-SP discussed in this Chapter considers a finite set of parts with several characteristics that have to be produced by a finite set of AM machines, each of which may have different specifications. In particular, each AM machine can produce more than one part simultaneously on its platform. Thus, the aim is to minimize the maximum completion time, i.e. the *makespan*. Indeed, it was described in the following multiple variants in Kucukkoc [119], where also the mathematical model was presented:

i) *AM single machine* when dealing with instances characterized only by one AM machine;

ii) *AM parallel identical machines* (PI-AMM-SP) if the instances are characterized by at least two AM machines having the same specifications;

iii) *AM parallel non-identical machines* (PNI-AMM-SP) when there are at least two AM machines with different specifications.

The only approach currently proposed in scientific literature relies on the use of CPLEX solver. However, our design of an efficient heuristic approach was motivated by the computational intractability of AMM-SP [119]. In fact, the i) version of AMM-SP is equivalent to the well-known *Bin Packing Problem* [115] which is strongly **NP**-hard [90]. Instead, in versions ii) and iii), the job processing time is a function of the volume and the height to be printed, hence minimizing the makespan is an even harder problem [37, 119].

6.2 AMM-SP vs BPMP

In this Section, we analyze similarities and differences between the AMM-SP and the related Batch Processing Machine Problem (BPMP), which has been widely studied in the literature, as surveyed in Section 6.2.1

The AMM-SP is based on the *Selective Laser Melting* technology, an expensive process both in terms of purchasing of the machines and layer by layer production. Moreover, the production of parts is commissioned by distributed customers: thus, to increase machines utilization and to reduce the average cost – due to set-up and post-processing operations –, it is necessary to group these parts in job batches and to schedule these batches on the machines [119], considering also that each AM machine can produce many parts simultaneously on its platform, as long as their total size is not greater than the capacity of the machine.

On the other hand, in the generic BPMP, jobs with or without identical characteristics (i.e. due date, size, ready time), have to be allocated into batches to be released on one or multiple, identical or not, machines to optimize specific performance metrics, e.g. makespan [33]. In particular, the machines can process several jobs simultaneously as a batch, given that the total size of all the jobs in the batch does not exceed the capacity of the machine on which it is scheduled. Actually, by letting *parts* and *jobs* in the AMM-SP play the roles of *jobs* and *batches* in the BPMP, the AMM-SP may look similar to the BPMP with arbitrary job sizes [126].

However, despite the similarity in the scheduling resolution procedure, there exists the following relevant differences between AMM-SP (with single or parallel machines) and BPMP with arbitrary job sizes [119, 122].

1. On the one hand, the batch processing time in the BPMP is determined by the largest processing time of the jobs included in it, where the processing time of a job is a parameter of the problem. On the other hand, the job processing time in the AMM-SP depends on the total volume of the parts included in the job, as well as on the maximum heights of those parts (see Section 6.3).

2. In the AMM-SP, the job processing time is computed via a function (Equation (6.3h) in Section 6.3) which takes into account the volume and the height to be printed. As a consequence, the parts included in a job batch strongly influence its processing time, that is to say that different combinations of parts yield to different costs.

In conclusion, in the AMM-SP, both the feasibility of a solution and the completion time strongly depend on the combination of parts considered in batches.

6.2.1 BPMP: Literature Review

Due to its computational intractability, the BPMP has been tackled with different heuristics and meta-heuristics.

In the 80-90s the earlier papers on BPMP considered only a single Batch Processing Machine (BPM). Just to cite a couple, Ikura and Gimple [105] proposed efficient algorithms for the minimization of the makespan in the case of jobs with different but compatible release times and due dates. Then, Uzsoy [169] proved the **NP**-completeness of the problems of minimizing the makespan and the total completion time in the case of arbitrary job sizes and proposed, among the others, the First-Fit Longest Processing Time heuristic.

More recently, the BPMP with several identical (IP-BPM) or not (NIP-BPM) machines has been addressed. Cheng et al. [37] described MILP models and computational complexity for both makespan and total completion time minimization for IP-BPM with non-identical job sizes. The authors also devised an Approximation Algorithm. Trindade et al. [168] proposed MILP models for four different BPMP versions, considering BPM and IP-BPM and jobs with or without release times. Jia and Leung [109] addressed the makespan minimization on IP-BPM with arbitrary job sizes through a meta-heuristic based on the Max-Min Ant System method. Then, this approach was adapted to the case of NIP-BPM in [110], where also a heuristic based on the First-Fit-Decreasing rule was presented. Instead, Suhaimi et al. [163] tackled the NIP-BPM via a Lagrangian method applied to two different relaxations of the problem. In Zhou et al. [182] the batching on identical machines with non-identical job sizes and arbitrary release times, was performed via distance matrix based heuristics. Recently, Zhang et al. [180] proposed an Evolutionary Algorithm to address the BPMP for the makespan minimization on identical AM machines.

However, to provide a more accurate description of real-world production systems, also the BPMP with *Unrelated Machines* (BPUM) has been defined by letting the speed of the machines depend on the jobs. At this purpose, Arroyo and Leung [8], addressed the version with identical BPUM and jobs with arbitrary features – size and ready time – with heuristics based on First-Fit and Best-Fit Earliest Job Ready Time rules. The same authors addressed the variant with non-identical BPUM in [7] via an Iterated Greedy Strategy.

Finally, it is worthwhile mentioning that the makespan minimization is not the only considered objective. For example, Li et al. [121] proposed heuristics procedure for the integrated production and delivery on IP-BPM of jobs with identical and non-identical sizes. Instead, Jia et al. [111] proposed an effective meta-heuristic based on Ant Colony optimization for the minimization of weighted completion time for IP-BPM with arbitrary job sizes.

6.3 Mathematical Formulation

In this Section we report a formal description of the problem, as originally presented in Kucukkoc [119]. For this purpose, first we thoroughly describe the elements involved in the AMM-SP, then we define the decision variables, and finally we present and comment the mathematical model.

As previously mentioned, the problem consists in the scheduling of operations to be performed by a collection $M = \{1, \dots, p\}$ of AM machines. The generic m-th AM machine is characterized by the following set of specifications (1-3) and known physical characteristics (4-5):

1. VT_m: the time to form per unit volume of material;
2. HT_m: the time for powder-layering, which depends on the highest produced part in each job;
3. SET_m: the time needed for set-up, e.g. initializing and cleaning, which is repeated for each job;
4. MA_m: the production area of the tray of machine;
5. MH_m: the maximum height supported by the machine.

The goal of the AMM-SP is to batch processing a collection $I = \{1, \dots, q\}$ of parts into job batches ($j \in J = \{1, \dots, k\}$) to be scheduled on AM machines optimally. Clearly, it holds that $k \leq q$, because there should be at least one part in each job. In particular, we refer to the *bounding box* of the part rather than to its actual shape [see 119]. Thus, each part $i \in I$ is only characterized by height (h_i), area (a_i), and volume (v_i). Given an arbitrary instance of AMM-SP, a part $i \in I$ is incompatible with a machine $m \in M$ when $h_i > MH_m$ or $a_i > MA_m$; i.e. a part cannot be grouped into any job on a machine when either the height of the part is greater than the maximum height supported by the machine, or its area is larger than the maximum area supported by the machine.

The mathematical formulation of the AMM-SP relies on the introduction of the following variables:

- Boolean variables X_{mji} describing the assignation of parts to jobs and machines:

$$X_{mji} = \begin{cases} 1, & \text{if part } i \text{ is assigned to job } j \text{ on machine } m, \\ 0, & \text{otherwise;} \end{cases}$$

- Boolean variables Z_{mj} describing the assignation of jobs to machines:

$$Z_{mj} = \begin{cases} 1, & \text{if job } j \text{ on machine } m \text{ contains at least one part,} \\ 0, & \text{otherwise;} \end{cases}$$

- real variables PT_{mj} representing the processing time of the j-th job on the m-th machine;
- real variables C_{mj} representing the completion time of the j-th job on the m-th machine.

The time spent to process the j-th job on the m-th machine is then defined as the sum of three terms, depending on: the set-up time, the volume and the height to be printed. Formally:

$$PT_{mj} = SET_m \cdot Z_{mj} + VT_m \cdot \sum_{i \in I} v_i \cdot X_{mji} + HT_m \cdot \max_{i \in I} \{h_i \cdot X_{mji}\}. \tag{6.1}$$

Moreover, assume that each machine is ready at time 0, i.e. $C_{m0} = 0, \forall m \in M$. Then, considering the sequential nature of the job processing, the completion time of the j-th job on the m-th machine is obtained as

$$C_{mj} \geq C_{m(j-1)} + PT_{mj}. \tag{6.2}$$

The resulting AMM-SP mathematical model, presented in Kucukkoc [119], is given in (6.3):

$$\text{(AMM-SP)} \ \min \left[\max_{m \in M, j \in J} C_{mj} \right] \tag{6.3a}$$

$$\text{s. t.}$$

$$\sum_{m \in M} \sum_{j \in J} X_{mji} = 1 \qquad \forall i \in I \tag{6.3b}$$

$$\sum_{i \in I} a_i \cdot X_{mji} \leq MA_m \qquad \forall m \in M, \forall j \in J \tag{6.3c}$$

$$h_i \cdot X_{mji} \leq MH_m \qquad \forall m \in M, \forall j \in J, \forall i \in I \tag{6.3d}$$

$$\sum_{i \in I} X_{m(j+1)i} \leq \psi \cdot \sum_{i \in I} X_{mji} \qquad \forall m \in M, \forall j \leq k-1 \tag{6.3e}$$

$$C_{m(j-1)} + PT_{mj} \leq C_{mj} \qquad \forall m \in M, \forall j \in J \tag{6.3f}$$

$$C_{m0} = 0 \qquad \forall m \in M \tag{6.3g}$$

$$PT_{mj} = SET_m \cdot Z_{mj} + VT_m \cdot \sum_{i \in I} v_i \cdot X_{mji} + HT_m \cdot \max_{i \in I} \{h_i \cdot X_{mji}\} \qquad \forall m \in M, \forall j \in J \tag{6.3h}$$

$$X_{mji}, Z_{mj} \in \{0, 1\} \qquad \forall m \in M, \forall j \in J, \forall i \in I \tag{6.3i}$$

The objective function (6.3a) represents the minimization of the maximum completion time, that is the makespan. The model is made up by: part occurrence constraints (6.3b) that restrain all parts to be processed exactly once; area (6.3c) and height (6.3d) capacity constraints, to ensure that parts are supported by the

assigned machine; constraints on job utilization (6.3e) to force the allocation of jobs with an incremental strategy, meaning that if job j of machine m is empty, then parts can not be assigned to job $j+1$; completion time constraints (6.3f) and (6.3g) and processing time constraints (6.3h) that specify how to compute completion and processing times of jobs. Finally there are sign constraints (6.3i).

6.4 Solution Approach

As discussed in Section 6.1, to the best of our knowledge the only solution approach presented in the literature for the AMM-SP tackles the linear programming model using CPLEX directly. However, due to its intractability, it is not always possible to solve optimally the problem with limited time resources. Consequently, the growth in the size of instances could foster the use of heuristic algorithms to yield promising sub-optimal solutions in short computational times.

At this purpose, in Alicastro et al. [3] we devised an *Iterated Local Search* (ILS) algorithm, whose local search employs a Learning Variable Neighbourhood Search. In particular, our algorithmic proposal leverages a Q-Learning algorithm – a model-free *Reinforcement Learning* technique – to achieve the best possible balance between three distinct neighbourhood structures. Moreover, with the aim of testing our solution proposal on large sized instances against another heuristic solution technique, we implemented an adaptation of the Evolutionary Algorithm (EA) presented by Zhang et al. [180] for the IP-BPM.

This choice was made:

- to have a reference point to better evaluate the performances of our heuristic proposal;
- because EA achieved promising results [180] on a problem that shares similar characteristics with the AMM-SP, as pointed out in Section 6.2.

The general structure of the ILS is outlined in Section 6.4.1, while its details are thoroughly described in Sections 6.4.2 to 6.4.5. In particular, Appendix 6.A illustrates the operations performed by our ILS through a numerical example. Finally, Section 6.4.6 details the *Probabilistic Stopping Rule* adopted as alternative stopping criterion for the ILS, while the adapted Evolutionary Algorithm is described in Section 6.4.7.

6.4.1 Iterated Local Search

Iterated Local Search (ILS) is among the most general single-point and memory-less meta-heuristics [4, 162]. Generally, ILS builds an initial solution randomly and at each iteration perturbs the current solution x – according to a specific heuristic procedure – and performs a local search in the neighbourhood of the perturbed solution. Both perturbation and local search procedures have a key role: the former aims to kick the current solution x out from the attraction basin of the local optimum (for *diversification* purposes), while the latter looks for better solutions, by performing an *intensification* phase.

During the years, the ILS has been implemented and hybridized in several different ways, incorporating biased-randomization [73], tabu search [139], and many other variants [129]. In Alicastro et al. [3], we designed a specific variant of ILS, in which the construction phase uses an encoded version of the solution, inspired by the genetic algorithm literature, and the local search is a Learning Variable Neighbourhood Search, based on a Q-Learning approach [151].

In particular, the construction phase (Section 6.4.2) builds randomly an encoded solution that is converted in a final solution by solving the Strip-Packing problem (Section 6.4.3); the perturbation phase changes the current encoded solution (Section 6.4.4) randomly, and the local search uses a learning and adaptive strategy to choose the best neighbourhood to explore (Section 6.4.5).

6.4.2 Construction and Encoding

In order to construct and perturb solutions efficiently, we used a strategy inherited from Random Key Genetic Algorithms which consists in encoding the solution through a vector of integers and managing the vector in the aforementioned phases.

Before presenting the encoding strategy, we point out that two machines with the same specifications –VT, HT, SET, MA and MH– are considered *logically equivalent*. Thus, letting t be the number of different types of machines of a given instance, $\widetilde{M}_1, \ldots, \widetilde{M}_t$ will denote the sets of the indexes of the logically equivalent machines, namely the *meta-machines*. We can then define the following vector representations of length q:

$$\bar{X}_i = \widetilde{m} \;\Leftrightarrow\; i \text{ is scheduled on } m \in \widetilde{M}_{\widetilde{m}}. \tag{6.4}$$

Therefore, $\bar{X}_i = \widetilde{m}$ means that the i-th part will be processed by a machine $m \in \widetilde{M}_{\widetilde{m}}$. In this encoding phase, no assumption is made about the regrouping of parts into jobs; consequently, a *Packing Problem* has to be solved during the decoding phase. For this reason, the construction phase is very efficient and simple: each part is assigned to a feasible meta-machine randomly, with respect to the constraints that limit the maximum height and area.

6.4.3 Decoding

The decoding phase aims to translate an encoded vector into an AMM-SP solution, by solving the problem of grouping parts into jobs. The aim is to minimize the total processing time needed by each machine, that is given by the sum of all the processing time of jobs related to that machine:

$$\begin{aligned}\sum_{j\in J} PT_{mj} &\stackrel{\text{def}}{=} \sum_{j\in J}\left[SET_m \cdot Z_{mj} + VT_m \cdot \sum_{i\in I} v_i \cdot X_{mji} + HT_m \cdot \max_{i\in I}\{h_i \cdot X_{mji}\}\right] \\ &= \underbrace{SET_m \cdot \sum_{j\in J} Z_{mj}}_{(a)} + \underbrace{VT_m \cdot \sum_{j\in J}\sum_{i\in I} v_i \cdot X_{mji}}_{(b)} + \underbrace{HT_m \cdot \sum_{j\in J} \max_{i\in I}\{h_i \cdot X_{mji}\}}_{(c)}.\end{aligned} \tag{6.5}$$

It is worthy to note that the term (b) is independent from the aggregation of parts into jobs. Thus, starting with an already existing part-machine association, the new – feasible – aggregation of parts into jobs ideally has to minimize: the number of performed setups (a), i.e., the number of utilized jobs, and the sum (c) of the heights of jobs in which parts are grouped, i.e. the height of the highest part contained in each job. Minimizing those quantities yields to the resolution of a particular Packing Problem known as *Strip Packing Problem* whose characteristics are briefly summarized in the following subsection, before giving the details of the decoding process.

Strip Packing Problem

An instance of Strip Packing Problem (Str-PP) is defined by a rectangular bin with finite width W and infinite height, and a set $\{o_1, \ldots, o_n\}$ of n axis-aligned rectangular items, each with width at most equal to W and finite height. The objective of this problem is to find a packing of all items into the bin so as to minimize the packing height under some physical constraints, namely: packed items can neither cover each other nor be rotated. Indeed, the Str-PP is a generalization of the Bin Packing problem, a well-known **NP**-hard problem [90]. In literature, approximation [22, 108] and exact [132] algorithms have been developed for Str-PP. Moreover, it has been proved that any offline bin packing algorithm can be applied to Str-PP maintaining the same asymptotic worst-case ratio [97]. Anyway, among the others, the Str-PP can be solved with a level-oriented approach which packs items in rows forming levels, such that in each level the items are bottom-aligned. Though, this algorithm often produces a sub-optimal solution, as depicted in Fig. 6.1 where on the left hand side, there is an optimal solution that minimizes the packing height, and, on the right hand side, there is a level-oriented feasible solution.

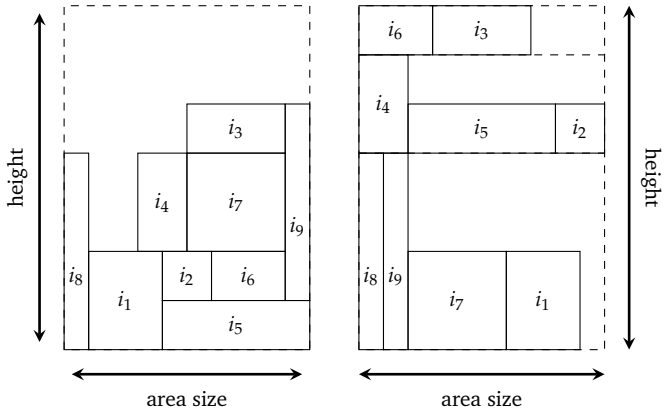


Fig. 6.1 Example of solutions for an instance of Strip Packing Problem.

However, the level-oriented solution brings an implicit partition into strips

that could be otherwise obtained only with a post-processing operation on the optimal solution. In particular, the post-processing may require an excessive time effort on large sized instances due to the complexity of the Str-PP.

Decoding Process Details

The aim of this Subsection is twofold: pointing out the analogy between the Str-PP and the Packing Problem associated to the decoding phase, and detailing the operations executed in that process.

Let $\bar{I} = \{i_1, \ldots, i_{\bar{h}}\}$ be the set of parts to be regrouped into jobs and $\widetilde{m}$ be the meta-machine on which the parts have to be processed. The problem of the aggregation of parts into jobs can be formulated as a Str-PP with a bin of width $W = MA_{\widetilde{m}}$ and items $\{o_1, \ldots, o_{\bar{h}}\}$, such that each o_j has width and height equal to those of part i_j. In particular, the feasibility of the set of items is guaranteed by the fact that, after the encoding phase, parts are associated only with compatible machines. Solving this Str-PP instance with a level-oriented approach ensures that each job can be processed by the $\widetilde{m}$-th meta-machine: regarding each strip as a job, it holds that the sum of the widths of the items of each strip – the sum of the area of the parts of each job – does not exceed the width of the bin – the maximum area capacity of meta-machine $\widetilde{m}$.

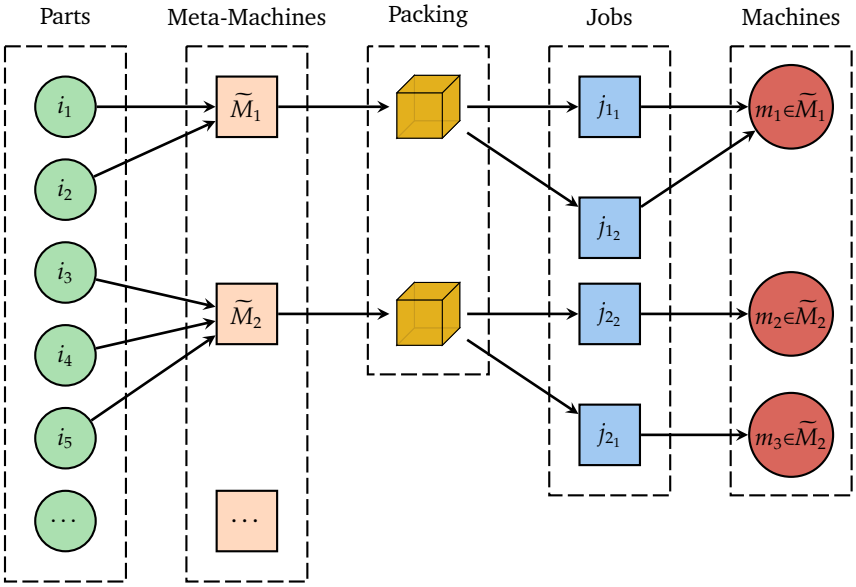


Fig. 6.2 Conceptual model of the Decoding Process.

Thus, all the parts are firstly divided into sets depending on the meta-machine they have to be scheduled on, and then, for each set, a Str-PP is solved. After that, one job at a time is scheduled on one of the less loaded machines $m \in \widetilde{M}_i$, sorting the resulting jobs by non-ascending processing times (Fig. 6.2).

As regards the level-oriented approach to solve the Str-PPs, we focus on three of the best known greedy bin packing offline algorithms [131]: *Next-Fit Decreasing Height*, *First-Fit Decreasing Height* and *Best-Fit Decreasing Height*. In particular, since none of these procedures turns out to be the best behaving, we apply all of them for each encoded solution vector v and decode v into the solution that exhibits the lowest makespan.

To describe these procedures, we suppose that: parts are sorted in non-ascending order by their heights, J is the list of already allocated jobs and the first $\bar{k}-1$ parts, with $\bar{k} \leq \bar{h}$ are already grouped into some jobs. Therefore the $\bar{k}$-th part, namely $i_{\bar{k}}$, has to be processed.

Next-Fit Decreasing Height (NFDH) According to this greedy strategy, part $i_{\bar{k}}$ is processed by the last allocated job $\bar{j} \in J$ if possible; otherwise a new job is created at this purpose and it is added to the list J (Fig. 6.3). NFDH has been proved to satisfy the worst-case performance ratio $r(\text{NFDH}) = 2$ [112].

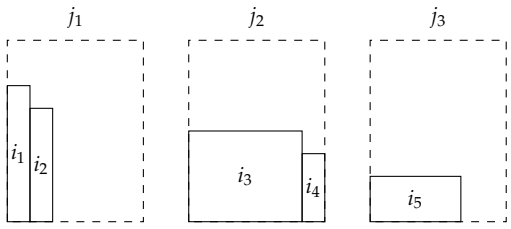


Fig. 6.3 Example of application of the Next-Fit Decreasing Height on an instance of Str-PP with 5 items.

First-Fit Decreasing Height (FFDH) It is an extension of NFDH since it attempts to use all the $|J|$ already allocated jobs: if $i_{\bar{k}}$ can be processed by at least one job in J, then the one with the lowest index – that is the job which was allocated first – is considered; otherwise, a new job is allocated, added to the list J, and used to process $i_{\bar{k}}$ (Fig. 6.4). It has been proved that the First-Fit greedy strategy satisfies the asymptotic worst-case performance ratio equal to $\frac{17}{10}$ [112]. Consequently, FFDH satisfies the asymptotic worst-case performance ratio $r^{\infty}(\text{FFDH}) = \frac{17}{10}$.

Best-Fit Decreasing Height (BFDH) BFDH works as FFDH, with the only difference that BFDH tries to maximize job utilization. If any job in J can process $i_{\bar{k}}$, then the one with the maximum occupied area – the job with the minimum residual area – is chosen. In particular, in case of a tie between two or more jobs, the one with the lowest index, that is the job which was allocated first, is considered. If none of the jobs in J can be used, a new job is allocated (Fig. 6.5). It has been proved that the Best-Fit greedy strategy satisfies the same worst-case

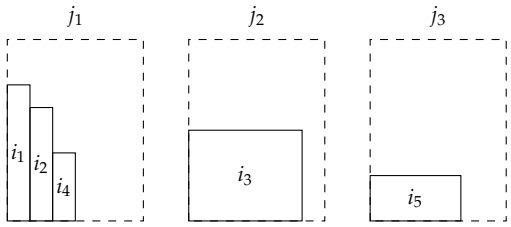


Fig. 6.4 Example of application of the First-Fit Decreasing Height on an instance of Str-PP with 5 items.

bounds as First-Fit greedy strategy [112], and, therefore, also BFDH satisfies the asymptotic worst-case performance ratio $r^{\infty}(\text{BFDH}) = \frac{17}{10}$.

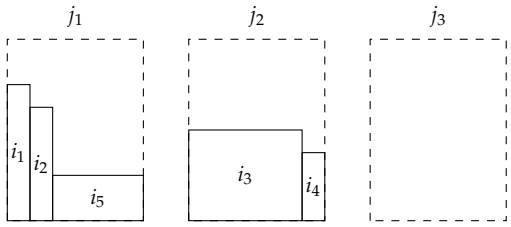


Fig. 6.5 Example of application of the Best-Fit Decreasing Height on an instance of Str-PP with 5 items.

6.4.4 ILS Perturbation

The perturbation function in our ILS follows a classic random scheme as described in Stützle and Ruiz [162]. Namely, our function changes a given number of components of the current encoded solution randomly. Specifically, given a vector of length n and a threshold $0 < \Theta \leq 1$, our perturbation selects $\lceil \Theta \cdot n \rceil$ parts randomly and schedules them on other compatible random meta-machines.

6.4.5 Local Search

The aim of the local search procedure is to explore the neighbourhood of the current solution so as to reach a local optimum that is better than the best solution found so far. In particular, in Alicastro et al. [3] we implemented a *Variable Neighbourhood Search* (VNS) [134] that uses three different neighbourhoods structures: *1-flip*, *2-swap*, and *job-alloc*. Formally, given a decoded solution x:

- **1-flip** neighbours of x are all the solutions with exactly one part scheduled on another job, belonging either to the same previous machine or to another machine;

- **2-swap** neighbours of x are all the solutions with exactly two parts swapped between the corresponding jobs, that either belong to the same machine or to two different machines;
- **job-alloc** neighbours of x are all the solutions having exactly two parts grouped into a new job on any machine.

Indeed, a key point of each VNS is the strategy adopted for selecting the neighbourhood to explore. A classic scheme is the *Variable Neighbourhood Descent* (VND), where the neighbourhoods are explored in a fixed order and the exploration re-starts from the first one each time improving solution is found.

Inspired by Queiroz Dos Santos et al. [151], we implemented a learning strategy that looks for the best neighbourhood to select, based on a policy updated at each iteration. In particular, we used a *Q-learning* algorithm, where an agent (the learner) interacts with its environment and selects the actions to be applied according to its current state and the reinforcement (i.e., rewards) it collects. The main goal of the agent is to maximize the reward, thus it provides to the learning algorithm a feedback about the effect of the taken actions.

Let $Q(t,a)$ be an estimation of the expected total reward obtained by performing action a on state t, named *action-value* function. The learning process consists of a sequence of epochs $(0,1,\dots,n,\dots)$. At epoch n, the agent is in state t and performs the action a receiving a reward $r(t,a)$; then, it moves to state t'. Hence, the action value $Q(t,a)$ is updated as follows:

$$Q(t,a) = Q(t,a) + \alpha\left[r(t,a) + \gamma \max_{a' \in \mathcal{A}} Q(t',a') - Q(t,a)\right], \tag{6.6}$$

where $\gamma \in [0,1]$ is a discount factor, $\alpha \in [0,1]$ is the learning rate, and $\mathcal{A}$ is the set of possible actions.

For the considered problem, the neighbourhood structures can play the roles of both action and states. In particular, the current state t is the last applied local search move, while the action a is the next move to be applied.

The local search procedure based on the Q-learning approach is depicted in Algorithm 8. In particular, we suppose that the set $\mathcal{A}$ of possible actions is initialized with $\mathcal{N} = (\mathcal{N}_1,\dots,\mathcal{N}_{l_{\max}})$. Moreover, in order to increase the *exploration factor*, a parameter $\epsilon \in (0,1)$ is given in input to the algorithm and is employed in the selection of the next action (Line 4): with probability ϵ, the algorithm selects the next action (local search) randomly; on the contrary, with probability $1-\epsilon$, the next action a is chosen to be the one that maximizes $Q(t,a)$.

6.4.6 Probabilistic Stopping Rule

While most meta-heuristic techniques rely on highly specific construction phases or local search procedures to achieve effective performance with small computational efforts, the same attention is not devoted to the stopping criterion, which often relies on rules that make hardly any usage of the information gathered during the execution. In fact, usual stopping criteria are based on a

Algorithm 8 Pseudo-code of local search with Q-learning

```
1: procedure QLearningLS
2:     Initialize x*.
3:     while A ≠ ∅ do
4:         Select_Search.
5:         x' = N_a(x).
6:         r(t,a) = cost(x*) − cost(x').
7:         if cost(x') < cost(x*) then
8:             x* = x'.
9:             A = N.
10:        else
11:            A = A \ {N_a}.
12:        Q(t,a) = Q(t,a) + α [r + γ max_{a'∈A} Q(a,a') − Q(t,a)].
13:        t = a.
14:    return x*.
15: end procedure
```

maximum number of iterations, total execution time, or a bound that limits the maximum number of consecutive iterations without improvement.

To shorten the computational times at the cost of negligible worsening in terms of solution quality, in Alicastro et al. [3] we implemented a *probabilistic stopping rule* that exploits the information collected at each iteration. This stopping criterion, inspired by Ribeiro et al. [155] and formally described in Felici et al. [63], estimates the probability of achieving further improvements of the incumbent solution, thus weighing-in the profitability of keep running the algorithm rather than terminating its execution.

Letting a random variable $\mathcal{Z}$ represent the objective function value achieved at the end of each iteration, the stopping rule consists of the two following successive phases: *data-fitting* and *probability estimation*.

1. The goal of the *data-fitting* is to approximate the *Probability Density Function* (PDF) $f_{\mathcal{Z}}(\cdot)$ of $\mathcal{Z}$. At this purpose, it is necessary to select a parametric family of distributions $\mathcal{F}$ whose member functions can be used to estimate $f_{\mathcal{Z}}(\cdot)$. This first choice is based mainly on empirical observations of the objective function values and their relative frequencies, obtained in a series of preliminary experiments. Then, by means of a Maximum Likelihood Estimation (MLE), the parameters characterizing the best fitting distribution of the chosen family are determined.

2. At a generic iteration, in which the algorithm holds an incumbent objective function value $\bar{z}$, the aim of the *probability estimation* is to evaluate the probability p of visiting a solution that improves $\bar{z}$. Once computed, such probability is compared with a user-defined threshold τ, and if $p < \tau$ the algorithm stops.

Following the reflection procedure described by Felici et al. [63], the parametric

family identified to be a good fit for the relative frequencies of $\mathcal{Z}$ is the Gamma distribution family, whose members are specified uniquely by two parameters: *shape* (κ), and *scale* (θ). Once the $\mathcal{F}$ has been selected, to estimate accurately the PDF, the ILS collects an initial sample of objective function values achieved in a user-defined number of iterations, and then executes a function – developed in R – which runs a MLE to compute the specific κ and θ values that identify the individual distribution of the Gamma family that best represents $f_{\mathcal{Z}}(\cdot)$, thus completing the data-fitting phase. Summarizing, the probabilistic stopping rule can be condensed in the following operations:

a) let the ILS collect the objective function values at the end of the first q iterations, where q is a user-defined positive integer;

b) using the collected values, execute the data-fitting procedure to compute the specific shape and scale parameters, κ and θ, of the best fitting Gamma distribution;

c) every time that an incumbent solution is improved, the probability estimation procedure is performed and the probability p of further improvements is computed. If p is less than or equal to τ, the stopping criterion is satisfied.

6.4.7 Evolutionary Algorithm

In order to compare our proposal with a sophisticated approach, in Alicastro et al. [3] we adapted one of the most recent approaches for the BPMP– an Evolutionary Algorithm (EA) – presented in the literature by Zhang et al. [180].

Indeed, Zhang et al. [180] addressed the makespan minimization for the IP-BPMP; however, some dissimilarities with our AMM-SP occur: on the one hand all the machines in their BPMP are identical, therefore the proposed approach cannot manage problems with non-identical machines easily. On the other hand, the parts to be scheduled in batches present a complex 3D shape, thus making the packing problem in Zhang et al. [180] more sophisticated. Taking into account these differences, we modified two aspects of their EA: the coding strategy in was adapted to check the "part-machine" associations, as to guarantee the feasibility of the solutions obtained at each phase of the algorithm (initialization, crossover, mutation...). Moreover, the packing strategy used by Zhang et al. [180] was replaced with our Strip Packing algorithm (Section 6.4.3), since only the bounding boxes of parts are considered in our problem, thus the packing problem to solve is quite different.

6.5 Computational Experiments

In this Section we detail the numerical experiments designed in Alicastro et al. [3]. In particular, we divide them in four different tests: a first preliminary analysis – as described in Section 6.5.2 – is carried out to tune the characteristic parameters of both the ILS here proposed, and the adapted EA described in

Section 6.4.7. Secondly, Section 6.5.3 details a comparison between the two meta-heuristics and ILOG CPLEX 12.8 solver on a set of small sized instances, to appraise the heuristic performance with respect to the optima computed by the solver. Section 6.5.4 compares the objective function values achieved by the ILS and EA in the solution of medium and large sized instances, for which the use of an exact solver is deemed impractical. Lastly, in Section 6.5.5 there is the analysis of the potentialities of the probabilistic stopping rule in the achievement of shorter computational times though slightly worsening the solution quality.

All the algorithms have been implemented in C++, and the experiments were run on a `INTEL i5-6400@2.70 GHz` processor with 8GB of RAM. A time limit of 60 seconds has been used for the two meta-heuristics, while for CPLEX the maximum running time has been set to 1 hour.

6.5.1 Test Problems and Evaluation Metrics

To appraise the algorithmic performance, in our experimental phase we include both already-proposed data-sets and ad-hoc generated instances[1]. In particular, the instances belong to the PI-AMM-SP and PNI-AMM-SP class and span a range of different sizes. The distinctive features characterizing each data-set and its source are reported in Table 6.1.

Table 6.1 Summary of instances properties.

Instances	Name	Type	#machines	#parts	Source
P15–P38	B	PI	2–3	15–46	Kucukkoc [119]
P39–P62	C	PNI	2–3	15–46	Kucukkoc [119]
P63–P102	D	PNI	2–6	10–420	Li et al. [122]
R01–R40	R	PNI	5–19	21–48	Randomly generated

The results discussed in Section 6.5.2 to Section 6.5.5 are reported in terms of the following statistics: the average of the objective function values; the number of times the algorithm reached the best solution; the average GAP and DEV. More specifically, we refer to GAP as the relative difference between an objective function value (OFV) and the one achieved by CPLEX (CPLEX OFV)

$$GAP = \frac{\text{OFV} - \text{CPLEX OFV}}{\text{CPLEX OFV}} \cdot 100, \tag{6.7}$$

while DEV is the relative difference between an objective function value and the best value found by one of the algorithms:

$$DEV = \frac{\text{OFV} - \text{best OFV}}{\text{best OFV}} \cdot 100. \tag{6.8}$$

[1]All the instances are available at `http://bit.ly/2sMYN7n`http://bit.ly/2sMYN7n.

6.5.2 Tuning

In order to tune the characteristic parameters of the ILS, we performed a preliminary tuning phase with the Irace package [128]. The data-set used in this tuning is composed by 25 randomly selected problem instances, and includes $\approx$ 20% of each group. The resulting parameters and their respective variation intervals are reported in Table 6.2. The perturbation percentage of the ILS is denoted by ρ; ϵ is the probability of selecting a random action in the Q-learning process; α and γ are the parameters used in the update of the policy (Algorithm 8). Finally, τ is the threshold used in the probabilistic stop.

Table 6.2 Configuration parameters obtained in the tuning phase.

Algorithm	Parameter	Interval	Selected
ILS	ρ	(0.05, 0.50)	0.35
	ϵ	(0.05,0.25)	0.11
	α	(0,1)	0.88
	γ	(0,1)	0.86
PS	τ	$\{0.05, 0.10, 0.15, 0.20, 0.25\}$	0.20

Finally, to evaluate the effectiveness of the Q-Learning process, we also implemented the ILS with a random VND in which the neighbourhoods are explored in a totally random fashion. The two approaches – ILS+Q-learning and ILS+VND-rand – have been tested on the tuning instances using 60 seconds as stopping criterion. The results are presented in Table 6.3 which reports, for both the methods, the Makespan values and the times to best (TtB).

The results reveal that the use of the Q-learning strategy yields a significant improvement of both the solution quality and the convergence speed. In fact, the average time to best of the ILS+Q-learning is $\approx$ 66% of that of the ILS+VND-rand. Finally, a two-sided Wilcoxon signed-rank test with critical value $\alpha = 0.01$ suggested that the averages are significantly different (p-value equal to 0.0007).

6.5.3 CPLEX vs Heuristics on Small-Sized Instances

A first set of experiments is devoted to the comparison between the objective function values achieved by CPLEX and the results of ILS and EA in the solution of small-sized instances, as done in Alicastro et al. [3].

For the sake of a fair comparison and continuity with respect to scientific literature, the linear programming model considered by CPLEX closely follows what described in Kucukkoc [119]. Consequently, in addition to the mathematical formulation describing the problem of interest, the computation introduces an a-priori limitation on j_n, the total number of jobs that can be scheduled.

Remark 12. It is duly noted that while a-priori limitation on j_n is not a constraint featured in the description of the problem, it is nevertheless introduced by Kucukkoc [119] to reduce computational times. In fact, guided by heuristic

Table 6.3 (Q-learning vs Random VND). Computational results in terms of objective function value (Makespan) and Time to Best.

	ILS+Q-learning		ILS+VND-rand	
	Makespan	**TtB**	**Makespan**	**TtB**
P25	434.74	41.09	435.04	25.54
P30	359.01	35.55	359.39	20.95
P34	381.19	22.94	381.73	31.41
P35	363.18	23.98	363.31	42.68
P37	438.58	39.03	438.86	22.10
P40	199.45	0.01	199.45	0.06
P50	451.51	4.65	451.51	21.57
P51	296.07	15.90	296.15	30.72
P54	356.93	25.66	356.96	26.39
P62	449.15	26.54	449.45	56.59
P66	3282.19	0.00	3282.19	0.01
P69	1457.57	0.00	1457.57	0.00
P72	5225.98	0.00	5225.98	0.00
P79	3970.98	11.45	3970.98	19.19
P81	5358.14	0.97	5358.14	3.20
P88	4486.98	8.09	4488.38	16.11
P89	6222.89	10.18	6235.03	35.60
P96	4488.72	9.80	4488.88	9.68
P102	27692.17	17.58	27779.67	55.59
R01	367.67	0.04	367.67	1.42
R06	426.02	36.42	426.89	21.17
R12	233.03	2.23	234.28	40.16
R13	232.96	0.01	232.96	2.15
R17	306.46	35.92	309.62	23.31
R27	706.65	2.12	706.65	45.54
Average	2727.53	14.81	2731.87	22.05

Table 6.4 (Data-set B). Computational results in terms of objective function value (Makespan) found by each method within the time limits (60 seconds for the two heuristics and 1 hour for CPLEX).

	CPLEX		ILS+Q-learning	EA
Instance	**Makespan**	**Total time**	**Makespan**	**Makespan**
P15	197.51	7.30	197.51	197.51
P16	203.89	7.47	203.89	204.49
P17	389.97	69.44	389.97	389.97
P18	397.78	6.98	397.78	397.78
P19*	381.17	4.50	378.76	378.76
P20	385.09	22.95	385.09	385.68
P21	280.61	68.11	280.61	281.20
P22	294.95	28.04	294.95	294.95
P23*	414.32	34.55	414.25	414.25
P24	433.10	18.68	433.10	433.10
P25*	435.43	175.77	434.74	435.83
P26	454.85	60.85	454.85	454.85
P27*	438.41	606.90	436.52	436.52
P28*	462.36	883.51	456.55	456.55
P29	348.46	1801.29	348.83	349.16
P30	358.81	3605.29	359.01	359.66
P31*	341.51	44.58	340.68	340.74
P32	349.25	1801.31	350.61	350.34
P33	368.99	1800.02	370.81	370.55
P34	378.52	3600.01	381.19	381.67
P35	361.74	1800.59	363.18	362.80
P36	372.16	1801.16	374.07	374.00
P37	436.00	1800.02	438.58	438.70
P38	447.07	3600.01	451.82	451.85

observations, the authors observed how fixing a-priori the total number of jobs, could imply sensibly shorter computational times. However, the maximum number of jobs characterizing optimal solutions can not be known a-priori, and thus the computation of such an upper bound requires some preliminary work to ensure that this additional constraint does not yield infeasibility or sub-optimality. At this purpose, every time CPLEX terminates within the given time limit and yet obtains solutions outperformed by at least one of the two heuristics, this implies sub-optimality of the solution for the corresponding instance. As a consequence, this kind of instances are marked with an asterisk in Tables 6.4-6.5. □

The results obtained (summarized in Table 6.6) evidence that, as expected, on average the best OFVs are achieved by CPLEX, yet the gap characterizing the two heuristic techniques is negligible. Additionally, we observe that for $\approx 20\%$ of the instances, the solver required the full allotted time, thus giving an average time

Table 6.5 (Data-set C). Computational results in terms of objective function value (Makespan) found by each method within the time limits (60 seconds for the two heuristics and 1 hour for CPLEX).

	CPLEX		ILS+Q-learning	EA
Instance	**Makespan**	**Total time**	**Makespan**	**Makespan**
P39	195.44	1.64	195.44	195.44
P40	199.45	1.55	199.45	199.45
P41	385.59	7.59	385.59	386.03
P42*	396.93	1.84	394.35	394.35
P43	372.58	6.38	372.58	372.58
P44	380.22	5.91	380.22	380.22
P45*	286.53	40.05	286.41	286.43
P46*	293.09	60.02	291.78	291.78
P47*	425.93	43.77	423.05	423.19
P48*	435.51	60.86	430.46	430.46
P49*	447.48	156.62	444.46	444.65
P50*	456.31	104.30	451.51	451.51
P51	296.05	112.09	296.07	300.00
P52	299.71	187.24	300.49	302.89
P53	352.16	3600.01	352.41	353.31
P54	357.82	3602.07	356.93	357.37
P55	342.30	3600.01	343.62	344.48
P56	345.05	3600.13	347.68	348.53
P57	372.58	3601.66	375.22	375.01
P58	376.90	3603.93	379.12	379.52
P59	364.12	3600.02	367.82	368.96
P60	368.43	2401.97	372.18	373.03
P61	444.08	2401.22	444.22	444.71
P62	446.55	1801.62	449.10	449.86

of $\approx 1172s$. Finally, we observe that though the additional constraint imposed on the maximum number of jobs speeds-up the computation, it yields at-least 13 instances with sub-optimal solutions.

As a final remark, in this first round of computational experiments, to evaluate the performance achieved by the three methods properly, we performed a non-parametric test to appraise the statistical significance of the differences in terms of objective function values. A *two-sided Wilcoxon signed-rank test* – with critical value $\alpha = 0.01$ – suggested that the solutions obtained by the two heuristics are not significantly different with those achieved by CPLEX (p-values of 0.5214 and 0.1123 for the ILS and EA, respectively). These results evidence that for small instances, both ILS and EA are able to find near optimal solutions efficiently.

Table 6.6 (Data-sets B and C). Summary of results for the small-sized instances.

	CPLEX	ILS+Q-learning	EA
avg(OFV)	366.10	366.20	366.56
avg(time)	1171.91	60.00	60.00
#best	34/48	28/48	18/48
avg(GAP)		0.04%	0.15%
avg(DEV)	0.16%	0.20%	0.31%

6.5.4 Comparison of Meta-heuristics

This Section is devoted to the comparison of the performance achieved by the proposed ILS and the EA adapted from the literature. To this aim, we take on the solution of medium-to-large sized instances to appraise the solution obtained on a data-set for which the use of CPLEX is impractical.

In particular, the analyses are based on the solution of two distinct data-sets, namely D and R. The former – taken from Li et al. [122] – is characterized by few parallel non-identical machines upon which a number of parts – from 10 to 42 – characterized by high variability has to be scheduled. On the contrary, the latter has been generated randomly to include in the experiments instances that presented a higher number of machines with fewer parts to be processed.

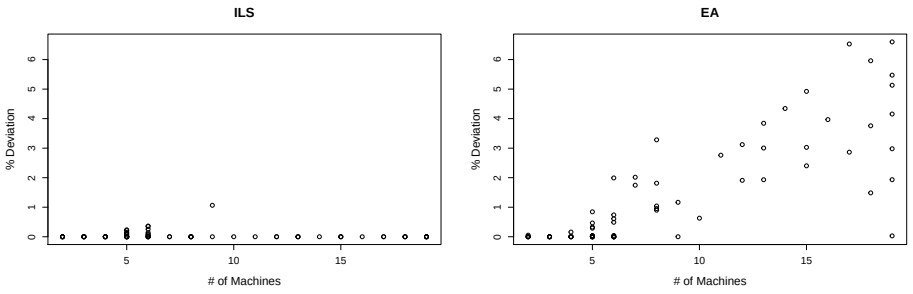
Analyzing the results – Tables 6.7 to 6.10 – it is possible to appraise the efficient implementation of both the meta-heuristics, that in 60 seconds are able to evaluate a high number of solutions (reported as #s.e.).

On data-sets D and R, ILS presents better results compared with EA (Table 6.9). In terms of solution values, the ILS found 63/77 best solutions, as opposed to the 30/77 achieved by EA. A similar difference appears also for the average OFVs and DEV. Additionally, from the recorded time to best, it is possible to note that both heuristics are able to achieve good quality solutions quickly. This behaviour, in addition to the high number of evaluated solutions, suggests that there may be room for improving the efficiency of the methods by further reducing the computation time (see Section 6.5.5). Also in this case, we performed a two-sided Wilcoxon signed-rank test – with critical value $\alpha = 0.01$ – obtaining a p-value equal to $7.454 \cdot 10^{-5}$. Therefore, the difference between the two methods is statistically significant.

Finally, Fig. 6.6 presents the percentage deviations with respect to the number of machines characterizing the instances of data-sets D and R. It is evident that the higher number of machines appears to affect the performance of EA mainly, with the ILS that consistently achieves the best solutions across almost the whole data-set. This behaviour can be related to the flexibility of the Variable Neighbourhood Search, whose neighbourhood structures are managed efficiently by the Q-learning approach.

Table 6.7 (Data-set D - PNI machines). Computational results in terms of objective function value (Makespan), Time to Best and number of solution evaluations of the two heuristics in 60s.

	ILS+Q-learning			EA		
Instance	**Makespan**	**TtB**	**#s.e.**	**Makespan**	**TtB**	**#s.e.**
P63	910.50	0.00	195455792.00	910.50	0.00	1206501.00
P64	1168.22	0.00	168122899.40	1168.22	0.00	1193286.60
P65	1516.24	0.01	186798528.40	1517.05	0.00	1184095.60
P66	3282.19	0.00	200321289.00	3282.19	0.00	1127863.60
P67	2043.57	0.69	178780217.40	2043.57	0.00	1133396.00
P68	6918.48	0.00	189170560.40	6918.48	0.00	1106312.20
P69	1457.57	0.00	172061604.60	1457.57	0.00	1147947.80
P70	1417.39	0.01	177912681.80	1417.39	0.00	1085863.80
P71	3777.21	0.00	180411625.60	3777.21	0.00	1070055.80
P72	5225.98	0.00	171606884.00	5225.98	0.00	1027955.00
P73	5993.22	0.01	164932490.00	5993.22	0.00	966560.60
P74	6871.53	0.00	168194657.40	6871.53	0.00	941965.00
P75	1912.28	0.00	172616767.60	1912.28	0.00	1060505.80
P76	1729.32	0.19	144615689.00	1729.39	20.33	990440.80
P77	5120.85	0.01	162992407.80	5120.85	0.00	955595.00
P78	6473.88	0.01	156694243.20	6473.88	0.00	931064.60
P79	3970.98	9.63	142652178.40	3977.30	3.16	886899.20
P80	8198.11	0.01	157966451.20	8198.11	0.00	876562.00
P81	5358.14	1.06	143567040.60	5360.32	25.38	960390.20
P82	1487.82	0.13	144347371.20	1488.23	8.71	934897.60
P83	4179.74	0.01	148644403.20	4179.74	0.00	880089.20



(a) Iterated Local Search **(b)** Evolutionary Algorithm

Fig. 6.6 Percentage deviations with respect to the number of machines.

Table 6.8 (Data-set D - PNI machines). Computational results in terms of objective function value (Makespan), Time to Best and number of solution evaluations of the two heuristics in 60s.

	ILS+Q-learning			EA		
Instance	**Makespan**	**TtB**	**#s.e.**	**Makespan**	**TtB**	**#s.e.**
P86	3267.40	35.80	95616747.00	3266.95	38.42	652822.80
P87	5660.99	28.12	91052416.40	5654.69	23.07	521669.00
P88	4489.16	21.32	68422597.20	4490.58	27.15	495533.40
P89	6236.83	34.99	76101501.00	6223.87	23.94	319399.20
P90	7166.71	36.63	68669884.80	7149.92	53.41	307951.80
P91	9597.28	29.25	65715122.40	9583.41	13.83	257248.00
P92	13739.83	37.61	65547032.60	13731.18	38.91	241010.20
P93	1351.09	0.12	125190577.60	1378.00	0.09	972161.60
P94	1739.14	13.56	110585111.20	1736.73	38.92	794752.40
P95	2976.48	53.64	95234381.00	2973.83	24.59	621489.40
P96	4491.54	35.30	81941102.80	4476.00	39.98	446016.40
P97	5054.10	26.98	68579136.80	5050.85	45.74	327303.40
P98	6596.19	43.07	53856672.00	6599.10	37.36	243309.40
P99	9243.98	37.98	46167338.60	9241.21	46.61	182321.60
P101	16995.16	36.95	40863319.60	16933.64	32.84	127817.80
P102	27718.10	30.92	35369305.80	27647.83	49.20	99371.00

Table 6.9 (Data-set D). Summary of results for the Comparison of Metaheuristics.

	ILS+Q-learning	EA
avg(OFV)	3125.97	3130.93
avg(TtB)	18.98	24.64
#best	63/77	30/77
avg(DEV)	0.04%	1.32%

Table 6.10 (Data-set R). Computational results in terms of objective function value (Makespan), Time to Best and number of solution evaluations of the two heuristics in 60s.

	ILS+Q-learning			EA		
Instance	**Makespan**	**TtB**	**#s.e.**	**Makespan**	**TtB**	**#s.e.**
R01	367.67	0.23	137600210.40	378.80	21.38	959656.00
R02	1004.25	34.52	85992360.40	1011.67	22.86	982115.40
R03	480.98	1.66	128732852.20	509.65	41.02	955905.00
R04	1908.18	30.33	86953258.40	1925.36	32.40	999617.60
R05	861.54	23.38	116563247.60	876.58	25.81	1057024.40
R06	426.02	36.42	81847180.20	421.53	53.29	914710.20
R07	4112.98	23.33	75267055.80	4124.90	24.73	1051679.40
R08	289.06	18.41	68087790.40	297.67	23.87	864809.80
R09	514.06	29.24	90333696.20	517.29	37.13	922032.60
R10	931.26	38.42	65000083.80	992.71	37.36	878894.00
R11	1465.57	28.81	80212525.60	1480.82	47.39	979493.80
R12	232.69	28.00	79338614.60	245.42	20.85	883067.40
R13	232.96	1.60	124442102.80	237.46	26.52	967674.00
R14	602.09	32.99	70643463.20	616.55	49.63	893650.20
R15	203.98	26.71	103814801.60	212.84	43.59	952676.40
R16	2397.33	25.50	99012670.80	2411.91	28.51	1056932.20
R17	306.46	35.92	92065567.80	316.52	41.24	897019.60
R18	432.92	32.60	88806125.00	446.43	26.03	957048.60
R19	1195.61	16.75	118001403.20	1205.71	19.47	1127769.00
R20	650.70	31.20	122556649.20	663.81	24.32	1110799.60
R21	2404.41	29.10	81461748.40	2427.35	50.18	960586.80
R22	338.62	30.24	90620928.60	355.29	33.81	943222.40
R23	398.80	14.31	73175670.60	424.83	22.86	888861.80
R24	478.96	33.86	73700161.20	503.54	38.39	858491.00
R25	488.10	0.03	137912287.20	488.24	15.90	974012.00
R26	507.71	9.23	114517026.40	513.64	23.99	1093576.80
R27	706.12	29.39	65133500.40	735.46	47.61	874182.20
R28	350.48	0.01	124502641.00	357.25	27.99	1032032.40
R29	466.19	23.34	81508788.40	483.71	50.56	870255.20
R30	693.75	29.24	65423227.60	713.62	34.23	873894.60
R31	675.53	28.15	100402056.80	694.19	39.99	980126.60
R32	379.52	6.12	99723638.60	390.93	16.99	1025732.60
R33	2560.00	26.70	108018449.40	2568.39	37.57	1047099.80
R34	454.26	28.41	98823120.00	471.72	28.11	960846.00
R35	428.43	25.56	117729310.00	436.20	33.99	1081345.60
R36	2274.81	39.73	96046417.40	2285.41	26.78	1037821.60
R37	128.12	0.18	186556339.40	130.02	15.79	987659.60
R38	1992.41	44.64	91440001.20	2002.18	35.64	1038506.80
R39	280.55	33.00	123754689.20	291.68	34.36	976395.40
R40	739.64	20.59	68327812.00	753.77	43.33	939174.40

6.5.5 Testing the Probabilistic Stopping Rule

In this Section we detail the computational experiments devised in Alicastro et al. [3] to test the probabilistic stopping rule (PS) described in Section 6.4.6. At this purpose, we compare the results obtained solving the instances of the data-set R employing the ILS algorithm with two different stopping criteria: a maximum computation time of 60 seconds (ILS + Q-learning), and the probabilistic stopping rule (ILS + Q-learning + PS).

The results reported in Table 6.12 and summarized in Table 6.11 show that with respect to the full 60 seconds, the solutions achieved using the PS present a significant decrease in terms of number of best solutions found – 3 out of 40 cases – yet the deviations recorded in singular instance are rather small, being on average equal to 1%. As a reference, comparing the results reported in Section 6.5.4 for the Evolutionary Algorithm, it can be noted that in the solution of R instances, the ILS + PS is characterized by slightly better: deviations (1% vs 2%), number of best solutions (3 vs 1), and average objective function values ($\approx$ 888 vs $\approx$ 898), as shown in Tables 6.10 and 6.11.

Moreover, the results achieved by ILS + PS evidence sensible improvements in terms of total computational time. In fact, when using the probabilistic stopping rule, the execution of the algorithm is terminated before the 60 seconds time limit in 39 out of 40 cases, with an average total execution time equal to 4.28 seconds. This result implies a 92.87% reduction of the execution times at the price of a 1% average worsening in terms of solution quality, thus suggesting that the probabilistic stopping rule can be a valuable tool to strike the proper balance between efficiency and effectiveness.

Table 6.11 (Data-set R). Statistics for the probabilistic stopping rule.

	ILS+Q-learning	ILS+Q-learning+PS
avg(OFV)	884.06	887.79
avg(time)	60.01	4.28
#best	40	3
avg(DEV)	0.00	0.01

Table 6.12 (Data-set R). Computational results achieved by ILS with and without the use of a probabilistic stopping rule.

	ILS+Q-learning			ILS+Q-learning+PS		
Instance	**Makespan**	**Time**	**#s.e.**	**Makespan**	**Time**	**#s.e.**
R01	367.67	60.00	137600210.40	368.37	1.48	499276.60
R02	1004.25	60.01	85992360.40	1007.36	2.82	2367833.00
R03	480.98	60.00	128732852.20	483.16	1.63	549159.80
R04	1908.18	60.00	86953258.40	1912.52	2.09	964546.20
R05	861.54	60.00	116563247.60	865.52	1.54	448391.20
R06	426.02	60.01	81847180.20	437.45	5.02	4666402.40
R07	4112.98	60.00	75267055.80	4116.54	2.19	1208067.40
R08	289.06	60.01	68087790.40	301.74	4.82	3567037.60
R09	514.06	60.02	90333696.20	516.33	5.19	5410968.60
R10	931.26	60.01	65000083.80	934.13	2.60	1353245.80
R11	1465.57	60.01	80212525.60	1470.37	2.64	1722243.80
R12	232.69	60.01	79338614.60	235.63	3.32	2486845.60
R13	232.96	60.00	124442102.80	233.12	1.59	756933.40
R14	602.09	60.01	70643463.20	604.02	3.55	2424805.00
R15	203.98	60.00	103814801.60	207.27	2.07	1128122.60
R16	2397.33	60.00	99012670.80	2401.33	1.57	518453.20
R17	306.46	60.01	92065567.80	318.03	4.09	3778105.40
R18	432.92	60.01	88806125.00	436.55	2.28	1365338.40
R19	1195.61	60.00	118001403.20	1199.40	1.45	361204.00
R20	650.70	60.00	122556649.20	656.13	1.51	526675.40
R21	2404.41	60.00	81461748.40	2407.18	2.32	1170283.80
R22	338.62	60.01	90620928.60	346.55	2.37	1480032.00
R23	398.80	60.01	73175670.60	403.98	2.95	1989287.80
R24	478.96	60.01	73700161.20	480.49	3.37	2405283.80
R25	488.10	60.00	137912287.20	488.10	1.49	479393.60
R26	507.71	60.00	114517026.40	509.36	14.93	19865244.80
R27	706.12	60.01	65133500.40	707.49	3.22	1898086.80
R28	350.48	60.00	124502641.00	350.48	1.39	312403.40
R29	466.19	60.01	81508788.40	468.64	3.22	2337568.80
R30	693.75	60.01	65423227.60	696.21	3.99	2815984.20
R31	675.53	60.01	100402056.80	678.34	2.09	1211934.00
R32	379.52	60.00	99723638.60	387.54	1.52	435860.80
R33	2560.00	60.00	108018449.40	2564.32	1.99	1102662.20
R34	454.26	60.00	98823120.00	457.68	2.20	1265039.40
R35	428.43	60.00	117729310.00	430.39	1.75	750645.20
R36	2274.81	60.00	96046417.40	2279.51	1.90	900709.60
R37	128.12	60.00	186556339.40	128.12	60.23	9759535.40
R38	1992.41	60.01	91440001.20	1995.01	2.11	1117554.60
R39	280.55	60.00	123754689.20	286.14	1.67	719500.80
R40	739.64	60.01	68327812.00	741.36	3.13	1811590.40

6.6 Conclusions

In this Chapter, we addressed the recently proposed Additive Manufacturing Machine Scheduling Problem (AMM-SP), outlining similarities and differences with the Parallel Batch Machine Problem (BPMP).

In particular, the AMM-SP is **NP**-hard, thus it is impractical to solve medium-large sized instances with limited time resources. At this purpose, in Alicastro et al. [3] we proposed heuristic solution approach, namely a *Reinforcement Learning Iterated Local Search* (ILS), which is thoroughly detailed in this Chapter.

With the aim of validating this approach, we compared its performances with the CPLEX solver and an Evolutionary Algorithm adapted from literature [180]. Through an extensive experimentation on data-sets both taken from scientific literature and ad-hoc generated, it has been observed that ILS turned out to be fast in reaching good solutions, and achieved notable results with respect to other approaches, especially when applied on medium-large sized instances. Moreover, we also employed a different stopping criterion in our algorithm, namely a *probabilistic stopping rule*, in order to detect whether a slight worsening of the solution quality could yield to better computational times.

However, we addressed a somehow simplified version of the AMM-SP since neither the orientation of parts has been considered nor nesting concerns have been given a relevant role. These simplifications yield to the resolution of a Packing Problem that only takes into account bounding boxes of parts. On the other hand, as thoroughly observed in Oh et al. [142], nesting outcomes – e.g. geometries of parts or orientation of jobs – may affect the chosen performance indicators, like the makespan. Consequently, as future line of investigation, the mathematical model could be further extended in order to consider true shape 2D/3D nesting algorithms to solve the resulting Packing Problem.

In addition, we retain that interesting results can be gathered by means of a deeper exploration of the multiple Strip Packing Problem [22, 23], mainly focusing on proof of an approximation ratio also for the AMM-SP of the three level-oriented algorithms considered in this Chapter.

Additionally, further researches could be focused on a more realistic context, where the information characterizing instances is related with some level of uncertainty, and could be formalized by means of stochastic variables. Given the intrinsic complexity of the AMM-SP, interesting research streams for addressing this scenario is related to the study of sim-heuristic algorithms [67, 82, 154].

Appendix

Appendix 6.A Numerical Example

The aim of this Section is to provide a numerical toy example in order to show how the proposed ILS algorithm works. The example problem consists of a set of 10 parts to be scheduled on 2 machines. The parameters of the machines and the specifications of the parts are presented in Table 6.13 and Table 6.14, respectively.

Table 6.13 Parameters of the machines in the toy example.

m	VT_m	HT_m	SET_m	MA_m	MH_m
1	0.030864	0.7	1.2	800.0	40.0
2	0.030864	0.7	1.2	600.0	35.0

Table 6.14 Specifications of the parts in the toy example.

p	h_p	a_p	v_p
1	4.27	122.62	102.83
2	2.18	178.34	214.79
3	29.58	273.83	840.17
4	18.99	89.68	683.06
5	10.77	269.75	1928.60
6	26.67	258.54	1375.90
7	14.38	114.56	989.53
8	3.50	454.89	683.48
9	3.00	615.12	722.91
10	17.04	99.53	703.08

The initial assignment of parts to meta-machines is obtained through a random initialization that produces the chromosome $[1,1,1,2,2,2,2,2,1,1]$, i.e. the parts 1, 2, 3, 9, and 10 are assigned to meta-machine $\widetilde{m_1}$, the others to meta-machine $\widetilde{m_2}$. All the three packing heuristics consider the parts according to their

height, in non-increasing order, as detailed next.

FFDH: for $\widetilde{m_1}$, parts 3, 10, and 1 all fit in the first job (j_1), while part 9 has to be scheduled on a new job (j_2), given the capacity constraint of the tray. The lower area characterizing part 2 allows it to be scheduled on job (j_1). The resulting packing is depicted in Fig. 6.7.

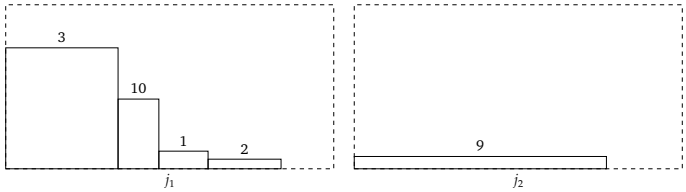


Fig. 6.7 (FFDH). Job/parts assignment for the meta-machine $\widetilde{m_1}$.

For $\widetilde{m_2}$, the remaining five parts are assigned as follows: 6, 4, and 7 all belong to the first job (j_4) on that machine, while the widths of 5 and 8 require two new jobs – j_5 and j_6 –, printing the remaining two parts separately. This packing yields the job/parts assignments of Fig. 6.8.

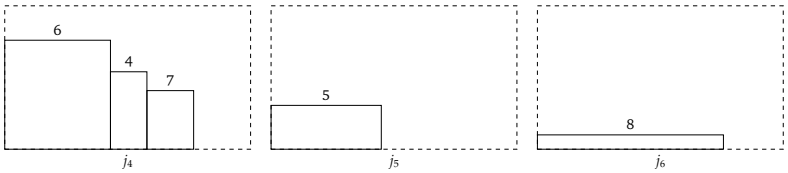


Fig. 6.8 (FFDH). Job/parts assignment for the meta-machine $\widetilde{m_2}$.

The makespan achieved by the First-Fit heuristic is 206.96.

BFDH: for $\widetilde{m_1}$, parts 3, 10, and 1, are assigned to the first job j_1, and part 9 to a second job j_2– as in the case of the First-Fit heuristic. Differently, while part 2 fits on both the first and second job, the evaluation of the Best-Fit heuristic assigns part 2 in the second job (see Fig. 6.9).

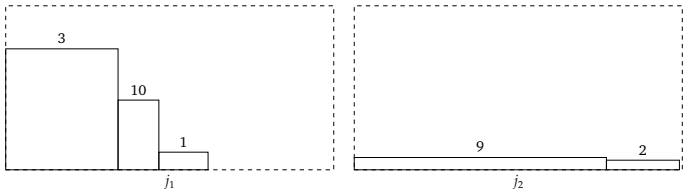


Fig. 6.9 (BFDH). Job/parts assignment for meta-machine $\widetilde{m_1}$.

The packing obtained for the second meta-machine is the same of the one

yielded by the FFDH. Also for the BFDH, the makespan yielded is 206.96.

NFDH: for the toy instance, the Next-Fit heuristic produces the same packing achieved by the Best-Fit procedure.

The proposed decoding algorithm selects the best solution out of the three. Since in this case, all the three packing heuristics produce solutions of the same quality, one of them is selected at random as the decoding design; namely, the solution produced by the First-Fit algorithm is chosen. Then, according to the Q-learning paradigms and its action values (see Section 6.4.5), one of the three possible local search procedures is selected: *1-flip*, *2-swap*, or *job-alloc*. After this selection, the algorithm evaluates all the feasible moves in the neighbourhood of the decoded solution, and the local search performs the move that improves the current objective function value, i.e. the makespan, the most.

Considering the *1-flip* as example, Table 6.15 shows all the feasible moves achievable from the decoded solution obtained by FFDH – reported in the first row –, and their corresponding makespan values. In such Table, the flip that distinguishes each solution from the decoded one is evidenced in bold font. Analysing the objective function values, the solution to be explored in the next iteration is obtained moving part 6 to either j_1 or j_2.

Table 6.15 Summary of feasible 1-flip moves and corresponding Makespan values, in the neighbourhood of the decoded solution.

$\widetilde{m_1}$		$\widetilde{m_2}$			**Makespan**
j_1	j_2	j_3	j_4	j_5	
$\{2,9,0,1\}$	$\{8\}$	$\{5,3,6\}$	$\{4\}$	$\{7\}$	206.96
$\{2,9,0,1\}$	$\{8\}$	$\{3,6\}$	$\{4,\mathbf{5}\}$	$\{7\}$	260.56
$\{2,9,0,1,\mathbf{3}\}$	$\{8\}$	$\{5,6\}$	$\{4\}$	$\{7\}$	185.88
$\{2,9,0,1\}$	$\{8,\mathbf{3}\}$	$\{5,6\}$	$\{4\}$	$\{7\}$	185.88
$\{2,9,0,1\}$	$\{8\}$	$\{5,6\}$	$\{4,\mathbf{3}\}$	$\{7\}$	233.80
$\{2,9,0,1\}$	$\{8\}$	$\{5,6\}$	$\{4\}$	$\{7,\mathbf{3}\}$	238.89
$\{2,9,0,1,\mathbf{6}\}$	$\{8\}$	$\{5,3\}$	$\{4\}$	$\{7\}$	176.42
$\{2,9,0,1\}$	$\{8,\mathbf{6}\}$	$\{5,3\}$	$\{4\}$	$\{7\}$	176.42
$\{2,9,0,1\}$	$\{8\}$	$\{5,3\}$	$\{4,\mathbf{6}\}$	$\{7\}$	240.03
$\{2,9,0,1\}$	$\{8\}$	$\{5,3\}$	$\{4\}$	$\{7,\mathbf{6}\}$	245.12

As last operation of the iteration, the action values used in the Q-learning algorithm are updated, possibly affecting the next selection of local neighbourhood structure.

Bibliography

[1] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, and K. Weihe. Network flows: theory, algorithms and applications. *ZOR-Methods and Models of Operations Research*, 41(3):252–254, 1995.

[2] C. Al Haddad, E. Chaniotakis, A. Straubinger, K. Plötner, and C. Antoniou. Factors affecting the adoption and use of urban air mobility. *Transportation research part A: policy and practice*, 132:696–712, 2020.

[3] M. Alicastro, D. Ferone, P. Festa, S. Fugaro, and T. Pastore. A reinforcement learning iterated local search for makespan minimization in additive manufacturing machine scheduling problems. *Computers & Operations Research*, page 105272, 2021.

[4] S. Alvarez Fernandez, D. Ferone, A. A. Juan, D. G. Silva, and J. de Armas. A 2-stage biased-randomized iterated local search for the uncapacitated single allocation p-hub median problem. *Transactions on Emerging Telecommunications Technologies*, 29(9):e3418, may 2018. ISSN 21613915. doi: 10.1002/ett.3418.

[5] E. Amaldi, G. Galbiati, and F. Maffioli. On minimum reload cost paths, tours, and flows. *Networks*, 57(3):254–260, 2011.

[6] C. Archetti, N. Bianchessi, and M. G. Speranza. A column generation approach for the split delivery vehicle routing problem. *Networks*, 58(4): 241–254, 2011.

[7] J. E. C. Arroyo and J. Y.-T. Leung. An effective iterated greedy algorithm for scheduling unrelated parallel batch machines with non-identical capacities and unequal ready times. *Computers & Industrial Engineering*, 105:84–100, 2017.

[8] J. E. C. Arroyo and J. Y.-T. Leung. Scheduling unrelated parallel batch processing machines with non-identical job sizes and unequal ready times. *Computers & Operations Research*, 78:117–128, 2017.

[9] D. Asprone, F. Auricchio, C. Menna, and V. Mercuri. 3d printing of reinforced concrete elements: Technology and design approach. *Construction*

and Building Materials, 165:218 – 231, 2018. ISSN 0950-0618. doi: 10.1016/j.conbuildmat.2018.01.018.

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

[11] S. Bandi and D. Thalmann. Space discretization for efficient human navigation. In *Computer Graphics Forum*, volume 17, pages 195–206. Wiley Online Library, 1998.

[12] X. Bao and Z. Liu. An improved approximation algorithm for the clustered traveling salesman problem. *Information Processing Letters*, 112(23): 908–910, 2012.

[13] C. Barrett, R. Jacob, and M. Marathe. Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

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

[15] J. Bertram, X. Yang, M. W. Brittain, and P. Wei. Online flight planner with dynamic obstacles for urban air mobility. In *AIAA Aviation 2019 Forum*, page 3625, 2019.

[16] D. Bertsekas. GRIDGEN generator, 1991. URL `http://www.mit.edu/~dimitrib/lopnet.txt`.

[17] D. Bertsekas. *Linear network optimization: algorithms and codes*. Mit Press, 1991.

[18] D. Bertsekas, W. Ding, and K. Qiu. A distributed algorithm for the assignment problem. *Lab. for Information and Decision Systems Working Paper*, March, 1979.

[19] D. P. Bertsekas. An auction algorithm for shortest paths. *SIAM Journal on Optimization*, 1(4):425–447, 1991.

[20] D. P. Bertsekas. A simple and fast label correcting algorithm for shortest paths. *Networks*, 23(8):703–709, 12 1993. doi: 10.1002/net.3230230808.

[21] H. T. T. Binh, P. D. Thanh, T. B. Trung, et al. Effective multifactorial evolutionary algorithm for solving the cluster shortest path tree problem. In *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8. IEEE, 2018.

[22] M. Bougeret, P. F. Dutot, K. Jansen, C. Otte, and D. Trystram. Approximation algorithms for multiple strip packing. In *Approximation and Online Algorithms*, pages 37–48. Springer Berlin Heidelberg, 2010. doi: 10.1007/978-3-642-12450-1_4.

[23] M. Bougeret, P. F. Dutot, K. Jansen, C. Robenek, and D. Trystram. Approximation algorithms for multiple strip packing and scheduling parallel jobs in platforms. *Discrete Mathematics, Algorithms and Applications*, 03(04): 553–586, Dec. 2011. doi: 10.1142/s1793830911001413.

[24] S. Bremen, W. Meiners, and A. Diatlov. Selective laser melting. *Laser Technik Journal*, 9(2):33–38, Apr. 2012. doi: 10.1002/latj.201290018.

[25] H. Broersma, X. Li, G. J. Woeginger, and S. Zhang. Paths and cycles in colored graphs. *Australasian Journal of Combinatorics*, 31:299–312, 2005.

[26] T. BrüGgemann, J. Monnot, and G. J. Woeginger. Local search for the minimum label spanning tree problem with bounded color classes. *Operations Research Letters*, 31(3):195–201, 2003.

[27] L. S. Buriol, M. G. Resende, and M. Thorup. Speeding up dynamic shortest-path algorithms. *INFORMS Journal on Computing*, 20(2):191–204, 2008.

[28] F. Carrabs, R. Cerulli, G. Felici, and G. Singh. Exact approaches for the orderly colored longest path problem: Performance comparison. *Computers & Operations Research*, 101:275–284, 2019.

[29] R. Cerulli, P. Festa, and G. Raiconi. Graph collapsing in shortest path auction algorithms. *Computational Optimization and Applications*, 18(3): 199–220, 2001.

[30] R. Cerulli, P. Festa, and G. Raiconi. Shortest path auction algorithm without contractions using virtual source concept. *Computational Optimization and Applications*, 26(2):191–208, 2003.

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

[32] E. P. Chan and Y. Yang. Shortest path tree computation in dynamic graphs. *IEEE Transactions on Computers*, 58(4):541–557, 2008.

[33] P.-Y. Chang, P. Damodaran*, and S. Melouk. Minimizing makespan on parallel batch processing machines. *International Journal of production research*, 42(19):4211–4220, 2004.

[34] R.-S. Chang and L. Shing-Jiuan. The minimum labeling spanning trees. *Information Processing Letters*, 63(5):277–282, 1997.

[35] S. Chen and Y. Shen. An improved column generation algorithm for crew scheduling problems. *Journal of Information and Computational Science*, 10(1):175–183, 2013.

[36] Y. F. Chen, S.-Y. Liu, M. Liu, J. Miller, and J. P. How. Motion planning with diffusion maps. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1423–1430. IEEE, 2016.

[37] B. Cheng, S. Yang, X. Hu, and B. Chen. Minimizing makespan and total completion time for parallel batch processing machines with non-identical job sizes. *Applied Mathematical Modelling*, 36(7):3161–3167, 2012.

[38] A. Chergui, K. Hadj-Hamou, and F. Vignat. Production scheduling and nesting in additive manufacturing. *Computers & Industrial Engineering*, 126:292 – 301, 2018. ISSN 0360-8352. doi: https://doi.org/10.1016/j.cie.2018.09.048. URL http://www.sciencedirect.com/science/article/pii/S0360835218304649.

[39] B. V. Cherkassky, V. Andrew, and T. Radzik. Shortest paths algorithms: Theory and experimental evaluation. *Mathematical programming*, 73(2): 129–174, 1996.

[40] B. V. Cherkassky, A. V. Goldberg, and C. Silverstein. Buckets, heaps, lists, and monotone priority queues. *SIAM Journal on Computing*, 28(4): 1326–1346, 1999.

[41] N. Christofides. *Graph theory: An algorithmic approach (Computer science and applied mathematics)*. Academic Press, Inc., 1975.

[42] T. H. Cormen. *Introduction to algorithms*. MIT press, 2009.

[43] O. Cosma, P. C. Pop, and I. Zelina. A novel genetic algorithm for solving the clustered shortest-path tree problem. *Carpathian Journal of Mathematics*, 36(3):401–414, 2020.

[44] J. Coykendall, M. Cotteleer, J. Holdowsky, and M. Mahto. 3d opportunity in aerospace and defense: Additive manufacturing takes flight. *A Deloitte series on additive manufacturing*, 1, 2014.

[45] P. Crescenzi, P. Fraigniaud, M. Halldorsson, H. A. Harutyunyan, C. Pierucci, A. Pietracaprina, and G. Pucci. On the complexity of the shortest-path broadcast problem. *Discrete Applied Mathematics*, 199:101–109, 2016.

[46] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations research*, 8(1):101–111, 1960.

[47] M. D'Emidio, L. Forlizzi, D. Frigioni, S. Leucci, and G. Proietti. On the clustered shortest-path tree problem. In *ICTCS*, pages 263–268, 2016.

[48] G. Desaulniers, J. Desrosiers, and M. M. Solomon. *Column generation*, volume 5. Springer Science & Business Media, 2006.

[49] M. Desrochers. *La fabrication d'horaires de travail pour les conducteurs d'autobus par une méthode de génération de colonnes*. Université de Montréal, Centre de recherche sur les transports, 1986.

[50] J. Desrosiers, P. Pelletier, and F. Soumis. Plus court chemin avec contraintes d'horaires. *RAIRO-Operations Research*, 17(4):357–377, 1983.

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

[52] L. Di Puglia Pugliese, D. Ferone, P. Festa, and F. Guerriero. Shortest Path Tour Problem with Time Windows. *European Journal of Operational Research*, sep 2019. ISSN 0377-2217. doi: 10.1016/J.EJOR.2019.08.052.

[53] L. Di Puglia Pugliese, D. Ferone, P. Festa, and F. Guerriero. Shortest Path Tour with Time Windows. *European Journal of Operational Research*, 282 (1):334–344, 2020. doi: 10.1016/j.ejor.2019.08.052.

[54] R. Dial, F. Glover, D. Karney, and D. Klingman. A computational analysis of alternative algorithms and labeling techniques for finding shortest path trees. *Networks*, 9(3):215–248, 1979.

[55] R. B. Dial. Algorithm 360: Shortest-path forest with topological ordering [h]. *Communications of the ACM*, 12(11):632–633, 1969.

[56] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[57] U. M. Dilberoglu, B. Gharehpapagh, U. Yaman, and M. Dolen. The role of additive manufacturing in the era of industry 4.0. *Procedia Manufacturing*, 11:545–554, 2017. doi: 10.1016/j.promfg.2017.07.148.

[58] E. D. Dolan and J. J. Morè. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[59] X. Dong, W. Li, and L. Zheng. Ant colony optimisation for a resource-constrained shortest path problem with applications in multimodal transport. *International Journal of Modelling, Identification and Control*, 18(3): 268–275, 2013.

[60] D. Dorninger. Hamiltonian circuits determining the order of chromosomes. *Discrete Applied Mathematics*, 50(2):159–168, 1994.

[61] I. Dumitrescu and N. Boland. Improved preprocessing, labeling and scaling algorithms for the weight-constrained shortest path problem. *Networks: An International Journal*, 42(3):135–153, 2003.

[62] M. D'Emidio, L. Forlizzi, D. Frigioni, S. Leucci, and G. Proietti. Hardness, approximability, and fixed-parameter tractability of the clustered shortest-path tree problem. *Journal of Combinatorial Optimization*, 38(1):165–184, 2019.

[63] G. Felici, D. Ferone, P. Festa, A. Napoletano, and T. Pastore. A GRASP for the Minimum Cost SAT Problem. In R. Battiti, D. E. Kvasov, and Y. D. Sergeyev, editors, *Learning and Intelligent Optimization. LION 2017*, volume 10556 of *Lecture Notes in Computer Science*, pages 64–78, Cham, 2017. Springer International Publishing. ISBN 978-3-319-69404-7.

[64] M. Fera, F. Fruggiero, A. Lambiase, R. Macchiaroli, and V. Todisco. A modified genetic algorithm for time and cost optimization of an additive manufacturing single-machine scheduling. *International Journal of Industrial Engineering Computations*, pages 423–438, 2018. doi: 10.5267/j.ijiec.2018.1.001.

[65] C. Feremans. Generalized spanning trees and extensions. *Universite Libre de Bruxelles, Diss*, 2001.

[66] C. Feremans, M. Labbé, and G. Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.

[67] D. Ferone, P. Festa, A. Gruler, and A. A. Juan. Combining simulation with a GRASP metaheuristic for solving the permutation flow-shop problem with stochastic processing times. In *2016 Winter Simulation Conference (WSC)*, pages 2205–2215. IEEE, dec 2016. ISBN 978-1-5090-4486-3. doi: 10.1109/WSC.2016.7822262.

[68] D. Ferone, P. Festa, F. Guerriero, and D. Laganà. The constrained shortest path tour problem. *Computers & Operations Research*, 74:64–77, 2016.

[69] D. Ferone, P. Festa, A. Napoletano, and T. Pastore. Reoptimizing shortest paths: From state of the art to new recent perspectives. In *2016 18th International Conference on Transparent Optical Networks (ICTON)*, pages 1–5. IEEE, jul 2016. ISBN 978-1-5090-1467-5. doi: 10.1109/ICTON.2016.7550605.

[70] D. Ferone, P. Festa, A. Napoletano, and T. Pastore. Shortest paths on dynamic graphs: A survey. *Pesquisa Operacional*, 37(3):487–508, 2017.

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

[72] D. Ferone, A. Gruler, P. Festa, and A. A. Juan. Enhancing and extending the classical GRASP framework with biased randomisation and simulation. *Journal of the Operational Research Society*, 70(8):1362–1375, 2019. ISSN 14769360. doi: 10.1080/01605682.2018.1494527.

[73] D. Ferone, S. Hatami, E. M. González-Neira, A. A. Juan, and P. Festa. A biased-randomized iterated local search for the distributed assembly permutation flow-shop problem. *International Transactions in Operational Research*, sep 2019. ISSN 0969-6016. doi: 10.1111/itor.12719.

[74] D. Ferone, P. Festa, S. Fugaro, and T. Pastore. A dynamic programming algorithm for solving the k-color shortest path problem. *Optimization Letters*, pages 1–20, 2020.

[75] D. Ferone, P. Festa, S. Fugaro, and T. Pastore. A dynamic programming algorithm for solving the k-color shortest path problem. *Optimization Letters*, pages 1–20, 2020.

[76] D. Ferone, P. Festa, S. Fugaro, and T. Pastore. On the shortest path problems with edge constraints. In *2020 22nd International Conference on Transparent Optical Networks (ICTON)*, pages 1–4. IEEE, 2020.

[77] D. Ferone, P. Festa, and F. Guerriero. An efficient exact approach for the constrained shortest path tour problem. *Optimization Methods and Software*, 35(1):1–20, jan 2020. ISSN 10294937. doi: 10.1080/10556788.2018.1548015.

[78] D. Ferone, P. Festa, S. Fugaro, and T. Pastore. A branch & price algorithm for the resource constrained clustered shortest path tree problem. *European Journal of Operational Research (Submitted)*, 2021.

[79] C. S. Ferreira, L. S. Ochi, V. Parada, and E. Uchoa. A grasp-based approach to the generalized minimum spanning tree problem. *Expert Systems with Applications*, 39(3):3526–3536, 2012.

[80] P. Festa and S. Pallottino. A pseudo-random networks generator. technical report, department of mathematics and applications "r. caccioppoli" , university of napoli federico ii, italy, 2003.

[81] P. Festa, F. Guerriero, and A. Napoletano. An auction-based approach for the re-optimization shortest path tree problem. *Computational Optimization and Applications*, 74(3):851–893, 2019.

[82] P. Festa, T. Pastore, D. Ferone, A. A. Juan, and C. Bayliss. Integrating biased-randomized GRASP with Monte Carlo simulation for solving the vehicle routing problem with stochastic demands. In *2018 Winter Simulation Conference (WSC)*, pages 2989–3000, 2019. doi: 10.1109/WSC.2018.8632348.

[83] P. Festa, S. Fugaro, and F. Guerriero. Shortest path reoptimization vs resolution from scratch: a computational comparison. *Optimization Methods and Software*, pages 1–23, 2021.

[84] M. Fischetti, J. J. S. González, and P. Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995.

[85] M. Fischetti, J. J. Salazar González, and P. Toth. A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, 45(3):378–394, 1997.

[86] M. Florian, S. Nguyen, and S. Pallottino. A dual simplex algorithm for finding all shortest paths. *Networks*, 11(4):367–378, 1981.

[87] S. Fugaro. Technical Report., 3 2021. URL `https://figshare.com/articles/book/Technical_Report/14140685`.

[88] G. Gallo. Reoptimization procedures in shortest path problem. *Rivista di matematica per le scienze economiche e sociali*, (3)(1):3–13, 1980.

[89] R. Garcia. *Resource constrained shortest paths and extensions*. PhD thesis, Georgia Institute of Technology, 2009.

[90] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness (Series of Books in the Mathematical Sciences)*. W. H. Freeman, 1979. ISBN 0716710455.

[91] G. Ghiani and G. Improta. An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, 122 (1):11–17, 2000.

[92] A. V. Goldberg and C. Silverstein. Implementations of dijkstra's algorithm based on multi-level buckets. In *Network optimization*, pages 292–327. Springer, 1997.

[93] L. Gourvès, A. Lyra, C. Martinhon, and J. Monnot. The minimum reload s–t path, trail and walk problems. *Discrete Applied Mathematics*, 158(13): 1404–1417, 2010.

[94] L. Gourvès, A. Lyra, C. A. Martinhon, and J. Monnot. Complexity of trails, paths and circuits in arc-colored digraphs. *Discrete Applied Mathematics*, 161(6):819–828, 2013.

[95] V. Griffiths, J. P. Scanlan, M. H. Eres, A. Martinez-Sykora, and P. Chinchapatnam. Cost-driven build orientation and bin packing of parts in selective laser melting (slm). *European Journal of Operational Research*, 273(1):334–352, 2019.

[96] N. Guttmann-Beck, R. Hassin, S. Khuller, and B. Raghavachari. Approximation algorithms with bounded performance guarantees for the clustered traveling salesman problem. *Algorithmica*, 28(4):422–437, 2000.

[97] X. Han, K. Iwama, D. Ye, and G. Zhang. Strip packing vs. bin packing. In *Algorithmic Aspects in Information and Management*, pages 358–367. Springer Berlin Heidelberg, 2007. doi: 10.1007/978-3-540-72870-2_34.

[98] P. E. Hart, N. J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[99] A. Henry-Labordere. The record balancing problem: A dynamic programming solution of a generalized traveling salesman problem rairo, vol, 1969.

[100] M. Horváth and T. Kis. Solving resource constrained shortest path problems with lp-based methods. *Computers & Operations Research*, 73:150–164, 2016.

[101] B. Hu, M. Leitner, and G. R. Raidl. Combining variable neighborhood search with integer linear programming for the generalized minimum spanning tree problem. *Journal of Heuristics*, 14(5):473–499, 2008.

[102] M. Hwangbo, J. Kuffner, and T. Kanade. Efficient two-phase 3d motion planning for small fixed-wing uavs. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1035–1041. IEEE, 2007.

[103] E. Ihler. Bounds on the quality of approximate solutions to the group steiner problem. In *International Workshop on Graph-Theoretic Concepts in Computer Science*, pages 109–118. Springer, 1990.

[104] E. Ihler, G. Reich, and P. Widmayer. Class steiner trees and vlsi-design. *Discrete Applied Mathematics*, 90(1-3):173–194, 1999.

[105] Y. Ikura and M. Gimple. Efficient scheduling algorithms for a single batch processing machine. *Operations Research Letters*, 5(2):61–65, 1986.

[106] S. Irnich. Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, 30(1):113–148, 2008.

[107] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In *Column generation*, pages 33–65. Springer, 2005.

[108] K. Jansen and M. Rau. Improved approximation for two dimensional strip packing with polynomial bounded width. In *WALCOM: Algorithms and Computation*, pages 409–420. Springer International Publishing, 2017. doi: 10.1007/978-3-319-53925-6_32.

[109] Z.-h. Jia and J. Y.-T. Leung. A meta-heuristic to minimize makespan for parallel batch machines with arbitrary job sizes. *European Journal of Operational Research*, 240(3):649–665, 2015.

[110] Z.-h. Jia, K. Li, and J. Y.-T. Leung. Effective heuristic for makespan minimization in parallel batch machines with non-identical capacities. *International Journal of Production Economics*, 169:1–10, 2015.

[111] Z.-h. Jia, H. Zhang, W.-t. Long, J. Y. Leung, K. Li, and W. Li. A meta-heuristic for minimizing total weighted flow time on parallel batch machines. *Computers & Industrial Engineering*, 125:298–308, 2018.

[112] D. S. Johnson, A. Demers, J. D. Ullman, M. R. Garey, and R. L. Graham. Worst-case performance bounds for simple one-dimensional packing algorithms. *SIAM Journal on Computing*, 3(4):299–325, Dec. 1974. doi: 10.1137/0203025.

[113] N. Jozefowiez, G. Laporte, and F. Semet. A branch-and-cut algorithm for the minimum labeling hamiltonian cycle problem and two variants. *Computers & Operations Research*, 38(11):1534–1542, 2011.

[114] C. Y. Justin and D. N. Mavris. Environment impact on feasibility of sub-urban air mobility using stol vehicles. In *AIAA Scitech 2019 Forum*, page 0530, 2019.

[115] S.-G. Koh, P.-H. Koo, J.-W. Ha, and W.-S. Lee. Scheduling parallel batch processing machines with arbitrary job sizes and incompatible job families. *International Journal of Production Research*, 42(19):4091–4107, Oct. 2004. doi: 10.1080/00207540410001704041.

[116] L. W. Kohlman and M. D. Patterson. System-level urban air mobility transportation modeling and determination of energy-related constraints. In *2018 Aviation Technology, Integration, and Operations Conference*, page 3677, 2018.

[117] A. M. C. A. Koster. *Frequency assignment: Models and algorithms*. Arie Koster, 1999.

[118] M. Kruber, M. E. Lübbecke, and A. Parmentier. Learning when to use a decomposition. In *International Conference on AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, pages 202–210. Springer, 2017.

[119] I. Kucukkoc. MILP models to minimise makespan in additive manufacturing machine scheduling problems. *Computers & Operations Research*, 105: 58–67, May 2019. doi: 10.1016/j.cor.2019.01.006.

[120] M. Leary. Surface roughness optimisation for selective laser melting (SLM). In *Laser Additive Manufacturing*, pages 99–118. Elsevier, 2017. doi: 10.1016/b978-0-08-100433-3.00004-x.

[121] K. Li, Z.-h. Jia, and J. Y.-T. Leung. Integrated production and delivery on parallel batching machines. *European Journal of Operational Research*, 247(3):755–763, 2015.

[122] Q. Li, I. Kucukkoc, and D. Z. Zhang. Production planning in additive manufacturing and 3d printing. *Computers & Operations Research*, 83: 157–172, July 2017. doi: 10.1016/j.cor.2017.01.013.

[123] Q. Li, D. Zhang, and I. Kucukkoc. Order acceptance and scheduling in direct digital manufacturing with additive manufacturing. *IFAC-PapersOnLine*, 52(13):1016–1021, 2019.

[124] Q. Li, D. Zhang, S. Wang, and I. Kucukkoc. A dynamic order acceptance and scheduling approach for additive manufacturing on-demand production. *The International Journal of Advanced Manufacturing Technology*, 105(9):3711–3729, 2019.

[125] W.-J. Li, H.-S. J. Tsao, and O. Ulular. The shortest path with at most/nodes in each of the series/parallel clusters. *Networks*, 26(4):263–271, 1995.

[126] X. Li, Y. Huang, Q. Tan, and H. Chen. Scheduling unrelated parallel batch processing machines with non-identical job sizes. *Computers & Operations Research*, 40(12):2983–2990, Dec. 2013. doi: 10.1016/j.cor.2013.06.016.

[127] C.-W. Lin and B. Y. Wu. On the minimum routing cost clustered tree problem. *Journal of Combinatorial Optimization*, 33(3):1106–1121, 2017.

[128] M. López-Ibáñez, J. Dubois-Lacoste, L. P. Cáceres, M. Birattari, and T. Stützle. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016. doi: 10.1016/j.orp.2016.09.002.

[129] H. R. Lourenço, O. C. Martin, and T. Stützle. Iterated local search: Framework and applications. In *Handbook of metaheuristics*, pages 129–168. Springer, 2019.

[130] Y. Marinakis, A. Migdalas, and A. 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.

[131] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley & Sons, Inc., 1990. ISBN 0-471-92420-2.

[132] S. Martello, M. Monaci, and D. Vigo. An exact approach to the strip-packing problem. *INFORMS Journal on Computing*, 15(3):310–319, Aug. 2003. doi: 10.1287/ijoc.15.3.310.16082.

[133] M. Mestria. A hybrid heuristic algorithm for the clustered traveling salesman problem. *Pesquisa Operacional*, 36(1):113–132, 2016.

[134] N. Mladenović and P. Hansen. Variable neighborhood search. *Computers & operations research*, 24(11):1097–1100, 1997.

[135] D. R. Morrison, S. H. Jacobson, J. J. Sauppe, and E. C. Sewell. Branch-and-bound algorithms: A survey of recent advances in searching, branching, and pruning. *Discrete Optimization*, 19:79–102, 2016.

[136] Y.-S. Myung, C.-H. Lee, and D.-W. Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[137] G. Nannicini and L. Liberti. Shortest paths on dynamic graphs. *International Transactions in Operational Research*, 15(5):551–563, 2008.

[138] A. Napoletano. *On Some Optimization Problems on Dynamic Networks*. PhD thesis, University of Naples Federico II, Italy, 2017.

[139] B. Nogueira, E. Tavares, and P. Maciel. Iterated local search with tabu search for the weighted vertex coloring problem. *Computers & Operations Research*, page 105087, sep 2020. doi: 10.1016/j.cor.2020.105087.

[140] S. Nuske, S. Choudhury, S. Jain, A. Chambers, L. Yoder, S. Scherer, L. Chamberlain, H. Cover, and S. Singh. Autonomous exploration and motion planning for an unmanned aerial vehicle navigating rivers. *Journal of Field Robotics*, 32(8):1141–1162, 2015.

[141] Y. Oh, C. Zhou, and S. Behdad. Production planning for mass customization in additive manufacturing: build orientation determination, 2d packing and scheduling. In *International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, volume 51753, page V02AT03A033. American Society of Mechanical Engineers, 2018.

[142] Y. Oh, P. Witherell, Y. Lu, and T. Sprock. Nesting and scheduling problems for additive manufacturing: A taxonomy and review. *Additive Manufacturing*, page 101492, 2020.

[143] T. Öncan, J.-F. Cordeau, and G. Laporte. A tabu search heuristic for the generalized minimum spanning tree problem. *European Journal of Operational Research*, 191(2):306–319, 2008.

[144] S. Pallottino and M. G. Scutellá. A new algorithm for reoptimizing shortest paths when the arc costs change. *Operations Research Letters*, 31(2):149–160, 2003.

[145] S. Pallottino and M. G. Scutellà. Shortest path algorithms in transportation models: classical and innovative aspects. In *Equilibrium and advanced transportation modelling*, pages 245–281. Springer, 1998.

[146] M. M. Pascoal and A. Sedeño-Noda. Enumerating k best paths in length order in dags. *European journal of operational research*, 221(2):308–316, 2012.

[147] T. Pastore, V. Mercuri, C. Menna, D. Asprone, P. Festa, and F. Auricchio. Topology optimization of stress-constrained structural elements using risk-factor approach. *Computers & Structures*, 224:106104, nov 2019. ISSN 0045-7949. doi: 10.1016/J.COMPSTRUC.2019.106104.

[148] T. Pastore, C. Menna, and D. Asprone. Combining multiple loads in a topology optimization framework for digitally fabricated concrete structures. In *RILEM International Conference on Concrete and Digital Fabrication*, pages 691–700. Springer, 2020.

[149] P. C. Pop. The generalized minimum spanning tree problem: An overview of formulations, solution procedures and latest advances. *European Journal of Operational Research*, 283(1):1–15, 2020.

[150] W. B. Powell and Z. Chen. A generalized threshold algorithm for the shortest path problem with time windows. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 40:303–318, 1998.

[151] J. P. Queiroz Dos Santos, J. D. De Melo, A. D. Duarte Neto, and D. Aloise. Reactive Search strategies using Reinforcement Learning, local search algorithms and Variable Neighborhood Search. *Expert Systems with Applications*, 41(10):4939–4949, 2014. ISSN 09574174. doi: 10.1016/j.eswa.2014.01.040.

[152] G. Ramalingam and T. Reps. An incremental algorithm for a generalization of the shortest-path problem. *Journal of Algorithms*, 21(2):267–305, 1996.

[153] G. Reich and P. Widmayer. Beyond steiner's problem: A vlsi oriented generalization. In *International Workshop on Graph-theoretic Concepts in Computer Science*, pages 196–210. Springer, 1989.

[154] L. Reyes-rubiano, D. Ferone, A. A. Juan, and J. Faulin. A simheuristic for routing electric vehicles with limited driving ranges and stochastic travel times. *SORT*, 43:1–22, 2019. doi: 10.2436/20.8080.02.77.

[155] C. C. Ribeiro, I. Rosseti, and R. C. Souza. Probabilistic stopping rules for grasp heuristics and extensions. *International Transactions in Operational Research*, 20(3):301–323, 2013.

[156] G. Righini and M. Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3(3):255–273, 9 2006. ISSN 1572-5286. doi: 10.1016/J.DISOPT.2006.05.007.

[157] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks: An International Journal*, 51(3):155–170, 2008.

[158] L. Santos, J. Coutinho-Rodrigues, and J. R. Current. An improved solution algorithm for the constrained shortest path problem. *Transportation Research Part B: Methodological*, 41(7):756–771, 2007.

[159] A. Sedeño-Noda and S. Alonso-Rodríguez. An enhanced k-sp algorithm with pruning strategies to solve the constrained shortest path problem. *Applied Mathematics and Computation*, 265:602–618, 2015.

[160] S. L. Smith and F. Imeson. Glns: An effective large neighborhood search heuristic for the generalized traveling salesman problem. *Computers & Operations Research*, 87:1–19, 2017.

[161] S. Srivastava, S. Kumar, R. Garg, and P. Sen. Generalized traveling salesman problem through n sets of nodes. *CORS journal*, 7(2):97, 1969.

[162] T. Stützle and R. Ruiz. Iterated local search. In R. Martí, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Heuristics*, pages 579–605. Springer International Publishing, Cham, 2018. ISBN 978-3-319-07124-4. doi: 10.1007/978-3-319-07124-4_8.

[163] N. Suhaimi, C. Nguyen, and P. Damodaran. Lagrangian approach to minimize makespan of non-identical parallel batch processing machines. *Computers & Industrial Engineering*, 101:295–302, 2016.

[164] P. D. Thanh, D. A. Dung, T. N. Tien, and H. T. T. Binh. An effective representation scheme in multifactorial evolutionary algorithm for solving cluster shortest-path tree problem. In *2018 IEEE congress on evolutionary computation (CEC)*, pages 1–8. IEEE, 2018.

[165] P. D. Thanh, H. T. T. Binh, N. B. Long, et al. A heuristic based on randomized greedy algorithms for the clustered shortest-path tree problem. In *2019 IEEE Congress on Evolutionary Computation (CEC)*, pages 2915–2922. IEEE, 2019.

[166] P. D. Thanh, H. T. T. Binh, and T. B. Trung. An efficient strategy for using multifactorial optimization to solve the clustered shortest path tree problem. *Applied Intelligence*, 50(4):1233–1258, 2020.

[167] C. Tilk and S. Irnich. Combined column-and-row-generation for the optimal communication spanning tree problem. *Computers & Operations Research*, 93:113–122, 2018.

[168] R. S. Trindade, O. C. B. de Araújo, M. H. C. Fampa, and F. M. Müller. Modelling and symmetry breaking in scheduling problems on batch processing machines. *International Journal of Production Research*, 56(22): 7031–7048, 2018.

[169] R. Uzsoy. Scheduling a single batch processing machine with non-identical job sizes. *The International Journal of Production Research*, 32(7):1615–1635, 1994.

[170] P. D. Vascik, R. J. Hansman, and N. S. Dunn. Analysis of urban air mobility operational constraints. *Journal of Air Transportation*, 26(4):133–146, 2018.

[171] Y. Wang, P. Zheng, X. Xu, H. Yang, and J. Zou. Production planning for cloud-based additive manufacturing—a computer vision-based approach. *Robotics and Computer-Integrated Manufacturing*, 58:145–157, 2019.

[172] B. Y. Wu and C.-W. Lin. On the clustered steiner tree problem. *Journal of Combinatorial Optimization*, 30(2):370–386, 2015.

[173] P. Wu, L. Li, J. Xie, and J. Chen. Probabilistically guaranteed path planning for safe urban air mobility using chance constrained rrt. In *AIAA AVIATION 2020 FORUM*, page 2914, 2020.

[174] S. Wu, M. Kay, R. King, A. Vila-Parrish, and D. Warsing. Multi-objective optimization of 3d packing problem in additive manufacturing. In *IIE annual conference. Proceedings*, page 1485. Institute of Industrial and Systems Engineers (IISE), 2014.

[175] H. Xu, D. M. Kilgour, K. W. Hipel, and G. Kemkes. Using matrices to link conflict evolution and resolution in a graph model. *European Journal of Operational Research*, 207(1):318–329, 2010.

[176] B. Yang and P. Gillard. The class steiner minimal tree problem: a lower bound and test problem generation. *Acta Informatica*, 37(3):193–211, 2000.

[177] A. Younes, U. A. Badawi, T. Farag, A. B. Salah, and F. A. Alghamdi. The shortest-path broadcast problem. *International Journal of Applied Engineering Research*, 13(10):7580–7584, 2018.

[178] S. Yuan, J. P. Jue, et al. Shared protection routing algorithm for optical network. *Optical Networks Magazine*, 3(3):32–39, 2002.

[179] S. Yuan, S. Varma, and J. P. Jue. Minimum-color path problems for reliability in mesh networks. In *IEEE INFOCOM*, volume 4, page 2658, 2005.

[180] J. Zhang, X. Yao, and Y. Li. Improved evolutionary algorithm for parallel batch processing machine scheduling in additive manufacturing. *International Journal of Production Research*, 58(8):2263–2282, 2020.

[181] X. Zhang, Z. Zhang, Y. Zhang, W. Daijun, and D. Yong. Route selection for emergency logistics management: A bio-inspired algorithm. *Safety science*, 54:87–91, 2013.

[182] S. Zhou, H. Chen, and X. Li. Distance matrix based heuristics to minimize makespan of parallel batch processing machines with arbitrary job sizes and release times. *Applied Soft Computing*, 52:630–641, 2017.

[183] X. Zhu and W. E. Wilhelm. A three-stage approach for the resource-constrained shortest path as a sub-problem in column generation. *Computers & Operations Research*, 39(2):164–178, 2012.