UNIVERSITÀ DI PARMA

Dottorato di Ricerca in Tecnologie dell'Informazione

XXXVI Ciclo

Motion Planning
for Multiple Autonomous Vehicles on a Graph

Coordinatore:

Chiar.mo Prof. Marco Locatelli

Tutor:

Chiar.mo Prof. Luca Consolini

Dottorando: *Stefano Ardizzoni*

A.A. 2020-2023

Summary

List of Figures

Introduction

Automated-Guided Vehicles (AGVs) are used to move items between different locations in a warehouse. They perform the same tasks assigned to operators driving forklifts with precision, flexibility and autonomy. Within logistics and production plants, AGVs make the management of goods movements efficient and completely automatic, guaranteeing perfect traceability at every stage of the process, and also allowing the overall safety of line operators to be improved. However, the coordination of a fleet of AGVs is a very complex problem, in which there are many variables to consider. This activity is entrusted to a so-called *Traffic Manager* (TM), i.e., a software tool which has to assign tasks to each AGV ensuring maximum efficiency of movements. Each task involves the assignment of a target position in the warehouse, which the AGV must reach to carry out a pick up or delivery operation. When an AGV receives a task, it chooses the path to reach the target independently from the routes of others AGVs. AGVs move along a set of virtual paths, commonly called roadmaps. A roadmap is nothing more than an abstraction of a directed graph, made up of segments (edges) and points (nodes). The segment is therefore defined as a connecting line between two points, and by appropriately joining contiguous segments we obtain the paths.

During the work of this thesis we collaborated on the MAF (Macchine Autonome e Flessibili) project with OCME packaging company S.r.l., based in Parma, Italy. This company develops internal handling and logistics solutions with the help of AGVs. MAF project aims at implementing a new *Traffic Manager* software based on newly developed approaches and algorithms to be applied to a fleet of AGVs. The objective is

primarily the development of a path planner for a fleet of AGVs. Indeed, the new TM must be able to plan the routes of all AGVs, with the aim of preventing and avoiding "deadlock" situations. A deadlock situation is a scenario in which a group of vehicles are mutually blocked along the route assigned to them, so that they are unable to reach their target position.

Main objective. The objective of this thesis is to study the algorithm for the path planner of the new *Traffic Manager*. As already mentioned, AGVs follow predefined paths, that connect the locations in which items are stored or processed. Therefore, we associate the layout of the paths (i.e., the roadmap) to a directed graph. In particular, we deal with strongly connected digraphs, directed graphs in which it is possible to reach any node starting from any other node. From now on, we consider a set of agents (or *pebbles*) on a graph $G = (V, E)$, with vertex (or node) set V and edge (or arc) set E. Each agent represents an AGV, occupies a different node and may move to unoccupied positions. The nodes represent positions in which items are picked up and delivered, together with additional locations used for routing. The directed arcs represent the precomputed paths that connect these locations. Obviously this is a simplification, as each AGV has a volume and takes up space around the node. In the first two chapters we will consider point-like agents, while in the third chapter we will introduce constraints to avoid collisions due to the real size of the agents.

From a mathematical point of view, the problem that must be solved by the path planner is the Multi-Agent Path Finding (MAPF) problem [1], which consists in computing a sequence of movements (called *moves*) that reposition all agents to assigned target nodes, avoiding collisions. Since we are primarily interested in avoiding deadlock situations, we will focus on the important task of finding a feasible solution to MAPF, even in crowded configurations. Once we find a feasible solution, we also study how to improve it, bringing it as close as possible to the optimal one. By optimal solution we mean the sequence of movements for each agent, which allows all targets to be reached in the shortest possible time.

Outline of the thesis and main contributions

The thesis is divided into four chapters, which deal with different problems associated with path planning for one or more agents on a graph.

In Chapter 1 we focus on the problem of finding the fastest path of a single AGV. In particular, we introduce a variant of the Shortest Path Problem (SPP), in which we impose additional constraints on the acceleration over the arcs, and call it Bounded Acceleration SPP (BASP). This variant is inspired by the industrial application: each AGV has to satisfy some speed and acceleration constraints depending on the vehicle position along the currently traveled path. We characterize the complexity of BASP, proving its NP-hardness. We also show that, under additional hypotheses on problem data, the problem admits a pseudo-polynomial time-complexity algorithm. Moreover, we present an approximation algorithm with polynomial time-complexity with respect to the data of the original problem and the inverse of the approximation factor ε. Furthermore, we present a solution algorithm (adaptive A*) for k-BASP, a subclass of BASP. Roughly speaking, a BASP instance is a k-BASP, with $k \in \mathbb{N}$, if the maximum number of nodes of a path that can be traveled with a speed profile of maximum acceleration, followed by one of maximum deceleration, starting and ending with null speed, without violating the maximum speed constraint, is smaller than k. We prove that k-BASP has polynomial time-complexity with respect to the graph size. The adaptive A* algorithm computes the optimal trajectory between a pair of nodes and adaptively determines the value of k. Finally, we present some computational experiments to evaluate the performance of the proposed algorithms.

Main contributions. To sum up, the main contributions of this Chapter are the introduction of a new problem (BASP) for the literature, the study of its complexity, and the proposal of two different polynomial algorithms that solve, respectively, the discretized version of BASP and a subclass of BASP (k-BASP). The topics of this Chapter were dealt with in the published papers [2, 3, 4].

In Chapter 2 we focus on the problem of finding a feasible solution for the Multi-

Agent Path Finding (MAPF) problem. Since our aim is to detect feasible solutions, here we impose that agents move one at a time, as also done, e.g., in [5, 6, 7, 8]. Of course, once we have found a feasible solution where agents move one at a time, it may be possible to find a shorter solution by allowing simultaneous moves. For instance, we can search in a neighborhood of the feasible solution, as in Chapter 4. Moreover, we assume that the edges of the graph have the same length. Indeed, if the agents can move one at a time then the length of the edges is not relevant when searching for a feasible solution.

In the first subsection, we focus on the special case of a MAPF on a tree. This variant, called *pebble motion on trees* (PMT) problem, has been widely studied because, in many cases, the more general Multi-Agent path finding (MAPF) problem on graphs can be reduced to PMT. One of the most important results about PMT is provided by Kornhauser [8], who describes a procedure, very difficult to implement, that finds solution of length $O(n^3)$. Other papers present algorithms easier to implement but with worse length complexity. In the second subsection we focus on the MAPF problem on strongly connected digraphs, which is the main problem for the purpose of this thesis. A directed graph $G = (V,E)$ is *strongly connected* if for all $v,w \in V$ there is a path from v to w. This is the type of graph most suitable for representing any warehouse, in which some segments can only be traversed in one direction, but there is always a path to go from one position to another. In literature, suboptimal algorithms exist for undirected and strongly biconnected directed graphs.

Main contributions. Our main contributions are as follows. We propose a simple and easy to implement procedure, which finds solutions of PMT of length $O(knc+n^2)$, where n is the number of nodes, k is the number of pebbles, and c the maximum length of corridors in the tree. This complexity result is more detailed than the current best known result $O(n^3)$, which is equal to our result in the worst case, but does not capture the dependency on c and k. Moreover, we discuss a variant of MAPF, the Motion Planning problem (MP in what follows), which consists in finding a plan such that a single marked agent reaches a desired target vertex, avoiding collisions. That is, in MP all non-marked agents are obstacles that need to be moved out of the way to re-position the marked one. These topics are discussed in the paper [9] submitted

to *Journal of Artificial Intelligence* with the collaboration of Bernhard Nebel. As regards the MAPF, our main contribution is to generalize the algorithms for undirected and strongly biconnected directed graphs to the more general case of strongly connected directed graphs. In particular, we describe a procedure that checks the problem feasibility in linear time with respect to the number of vertices n, and we find a necessary and sufficient condition for feasibility of any MAPF instance. Moreover, we present an algorithm (diSC) that provides a feasible and suboptimal solution with length $O(kn^2c)$, where c is the maximum length of the corridors of the graph. Some of these contributions have already been presented at the *61st Conference on Decision and Control* with the paper [10], while others will be part of a future publication.

In Chapter 3 we discuss $\mathscr{C}$-MP and $\mathscr{C}$-MAPF, generalizations of the classical Motion Planning (MP) and Multi-Agent Path Finding (MAPF) problems on a directed graph G. The motivation for these generalizations is linked to the industrial application of MP and MAPF. Indeed, until now we have considered AGVs as point-like bodies without volume. In fact, if the center of an agent is located on a node, its body may occupy the space around that node. Therefore, to avoid collisions between AGVs, new constraints that allow maintaining a safety distance between agents must be introduced. In $\mathscr{C}$-MP and $\mathscr{C}$-MAPF, we enforce an upper bound on the number of agents that occupy each member of a family of vertex subsets.

Main contributions. In addition to presenting these two problems, new to literature, we prove that finding a feasible solution of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF is NP-hard. Also, we propose a method to convert these problems to standard MP and MAPF by strengthening the constraints. The method consists in finding a subset of vertices W and a reduced graph G_W, such that a feasible solution of MP and MAPF on G_W provides, in polynomial time, a feasible solution of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF on G. However, since the conversion into standard MP and MAPF is obtained by strengthening constraints, feasible solutions of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF on G may exist even if MP and MAPF on G_W do not admit any feasible solution. We also study the problem of finding W of maximum cardinality. First, we show that such problem is strongly NP-hard. Then, we propose a heuristic approach for its solution. The topics of this Chapter are

dealt with in the paper [11] submitted to *Automatica*.

In Chapter 4 we introduce an iterative local search procedure in MAPF. While in the previous chapters we study algorithms to find a feasible suboptimal solution, that satisfies constraints related to the size of the agents, in the last Chapter we focus on how to improve this solution. diSC algorithm presented in Chapter 2 is complete, since it allows finding a feasible solution that brings each agent to its target. However, generally, it provides solutions that are much longer than the optimal one. The objective of this Chapter is to improve these solutions. We start from a feasible suboptimal solution, and we perform a local search in a neighborhood of this solution, to find a shorter one. Iteratively, we repeat this procedure until the solution cannot be shortened any longer. At the end, we obtain a solution, that is still sub-optimal, but, in general, of much better quality than the initial one.

Main contributions. The main contribution of this Chapter is the introduction of such a local search procedure, which uses dynamic programming. Under this respect, the fact that our search is *local* is fundamental to reduce the time complexity of the algorithm. Indeed, if we apply a standard dynamic programming the number of explored states grows exponentially with the number of agents. As we will see, the introduction of a locality constraint allows solving the (local) dynamic programming problem in a time that grows only polynomially with respect to the number of agents. The topics of this Chapter will be presented at the *62st Conference on Decision and Control* with the accepted paper [12] .

Notation

A directed graph is a pair $G = (V,E)$ where $V = \{\alpha_1, \ldots, \alpha_N\}$ is a set of nodes and $E \subset \{(\alpha_i, \alpha_j) \in V^2 \mid \alpha_i \neq \alpha_j\}$ is a set of directed arcs. For each $i \in \{1, \ldots, N\}$, node α_i represents an operating point $R_i \in \mathbb{R}^2$. In fact, the restriction $R_i \in \mathbb{R}^2$ is not strictly necessary but we imposed it since it holds in the AGV application, which is the main motivation of this work. Each arc $\theta = (\alpha_i, \alpha_j) \in E$ represents a fixed directed path

between two operating points and is associated to an arc-length parameterized path γ_θ of length $\ell(\theta)$, such that $\gamma_\theta(0) = R_i$ and $\gamma_\theta(\ell(\theta)) = R_j$.

A path π on G is a sequence of adjacent nodes of V, i.e., $\pi = \sigma_1\,\sigma_2 \cdots \sigma_{m-1}\,\sigma_m$ (or, equivalently, $\pi = \sigma_1 \to \sigma_2 \to \cdots \to \sigma_{m-1} \to \sigma_m$), with $(\forall i \in \{1,\dots,m\})\ (\sigma_i, \sigma_{i+1}) \in E$. An alphabet $\Sigma = \{\sigma_1,\dots,\sigma_n\}$ is a set of symbols. A word is any finite sequence of symbols. The set of all words over Σ is Σ^*, which also contains the empty word ε, while Σ_i represents the set of all words of length up to $i \in \mathbb{N}$, (i.e., words composed of up to i symbols, including ε). Given a word $w \in \Sigma^*$, $|w|$ denotes its length and $\mathrm{Pref}(w)$ denotes the set of the prefixes of w (including ε).

Given a directed graph $G = (V,E)$, we can think of V as an alphabet so that any path π of G is a word in V^*. Given $s,t \in \Sigma^*$, the word obtained by writing t after s is the concatenation of s and t, denoted by $st \in \Sigma^*$; we call t a suffix of st and s a prefix of st. For $r \in V^*$, $\vec{r}$ is the rightmost symbol of r. In the following, we represent paths of G as strings of symbols in V. This allows us to use the concatenation operation on paths and to use prefixes and suffixes to represent portions of paths.

In the following, we denote the set of all possible paths on G by $\mathscr{P}$. Similarly, for $o,d \in V$, we denote by $\mathscr{P}_o$ the subset of $\mathscr{P}$ consisting in all paths starting from o and by $\mathscr{P}_{o,d}$ the subset of $\mathscr{P}$ consisting in all paths starting from o and ending in d. We extend this definition to subsets of V, that is, if $O,D \subset V$, we denote by $¶_{O,D}$ the set of all paths starting from nodes in O and ending in nodes in D. Given a path $\pi = \sigma_1 \cdots \sigma_m$, its length $\ell(\pi)$ is defined as the sum of the lengths of its individual arcs, that is, $\ell(\pi) = \sum_{i=1}^{m-1} \ell(\sigma_i, \sigma_{i+1})$.

For $x \in \mathbb{R}$, $\lceil x \rceil = \min\{i \in \mathbb{Z} \mid i \geq x\}$ is the ceiling of x. For $a,b \in \mathbb{R}$, we set $a \wedge b = \min\{a,b\}$ and $a \vee b = \max\{a,b\}$, as the minimum and maximum operations, respectively. Finally, given an interval $I \subseteq \mathbb{R}$, we recall that $W^{1,\infty}(I)$ is the Sobolev space of functions in $L^\infty(I)$ with weak derivative of order 1 with finite L^∞-norm. For $f,g \in W^{1,\infty}(I)$, we denote with $f \wedge g$ and $f \vee g$ the point-wise minimum and maximum of f and g, respectively.

Publications

Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Efficient solution algorithms for the bounded acceleration shortest path problem. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 5729–5734. IEEE, 2021.

Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Shortest path with acceleration constraints: complexity and approximation algorithms. *Computational Optimization and Applications*, 83(2):555–592, 2022.

Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Solution algorithms for the bounded acceleration shortest path problem. *IEEE Transactions on Automatic Control*, 68(3):1910–1917, 2022.

Stefano Ardizzoni, Luca Consolini, Marco Locatelli, Irene Saccani, and Bernhard Nebel. An algorithm with improved complexity for pebble motion/multi-agent path finding on trees. Submitted to *Journal of Artificial Intelligence*.

Stefano Ardizzoni, Luca Consolini, Marco Locatelli, and Irene Saccani. Multiagent path finding on strongly connected digraphs. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 7194–7199. IEEE, 2022.

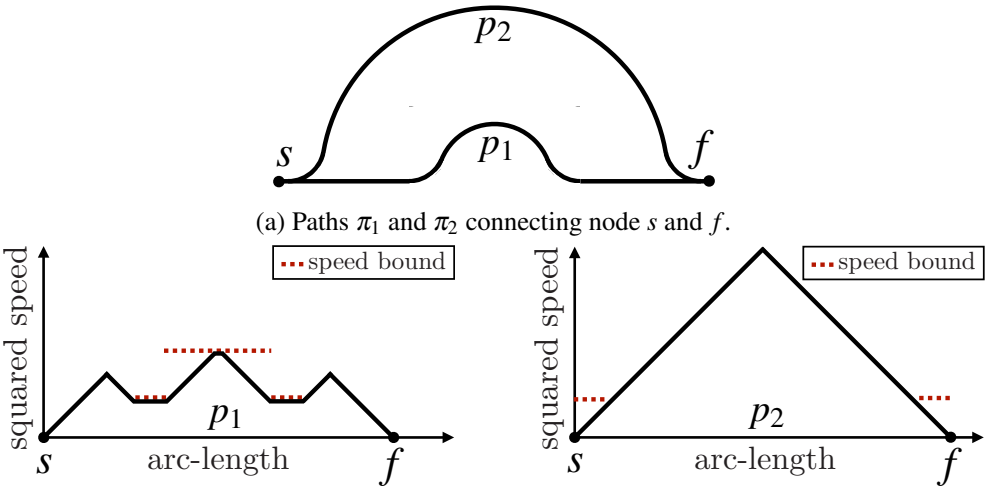
Stefano Ardizzoni, Luca Consolini, Marco Locatelli, and Irene Saccani. Constrained multi-agent path finding on directed graphs. Submitted to *Automatica*.

Stefano Ardizzoni, Luca Consolini, Marco Locatelli, and Irene Saccani. Local optimization of mapf solutions on directed graphs. To appear in *2023 IEEE 62st Conference on Decision and Control (CDC)*, 2023.

Chapter 1

The Bounded Acceleration Shortest Path Problem

The combinatorial problem of detecting the best path from a source to a destination node over an oriented graph with *constant* costs associated to its arcs, also known as Shortest Path Problem (SPP in what follows), is well known and can be efficiently solved, e.g., by Dijkstra algorithm (in case of nonnegative costs). The continuous problem of minimum-time speed planning over a *fixed* path under given speed and acceleration constraints, also depending on the position along the path, is also widely studied and very efficient algorithms for its solution exist. But the combination of these two problems, called in what follows Bounded Acceleration Shortest Path Problem (BASP), turns out to be more challenging than the two problems considered separately. More precisely, in terms of complexity theory, it is possible to prove that BASP is NP-hard, while the two problems considered separately are both polynomially solvable. In BASP we still have the combinatorial search for a best path as in SPP but, differently from SPP, the cost of an arc (more precisely, the time to traverse it) is not a constant value but depends on the speed planning along the arc itself which, in turn, depends on the speed and acceleration constraints not only over the same arc but also over those preceding and following it in the selected path. Figure 1.1a presents a simple scenario that allows to illustrate BASP and its difference with SPP; it shows two fixed



(a) Paths π_1 and π_2 connecting node s and f.

(b) Optimal speed profile on p_1.

(c) Optimal speed profile on p_2.

Figure 1.1: Comparison of BASP and SP solutions.

paths π_1 and π_2 connecting positions s and f. The vehicle starts from s with null speed and must reach f with null speed. The solution of SPP corresponds to path π_1, which is the one of shortest length. BASP consists in finding the shortest-time path under acceleration and speed constraints. In this case, we assume that the vehicle acceleration and deceleration are bounded by a common constant and that its speed is bounded only on the central, high-curvature section of π_1, in order to avoid excessive lateral acceleration, which may cause sideslip. If the bound on acceleration and deceleration is sufficiently high, the solution of BASP corresponds to path π_2. Indeed, even if the latter path is longer, it can be traveled with a greater mean speed. Figure 1.1b represents the fastest speed profile on π_1. The x-axis corresponds to the arc-length position on path π_1 and the y-axis represents the squared speed. In this representation, arc-length intervals of constant acceleration or deceleration correspond to straight lines. Figure 1.1c represents the fastest speed profile on π_2. Even if path π_2 is longer than π_1, it can be traveled in less time. In fact, the vehicle is able to accelerate till the midpoint and then to decelerate to the end position f.

The interest for BASP comes from the industrial application on the optimization of

automated guided vehicles (AGVs) motion in automated warehouses. The AGVs may be either free to move within a facility or be only allowed to move along predetermined paths. In the first case, one needs to employ environmental representations such as cell decomposition methods [13] or trajectory maps [14]. In particular, [15] presents an algorithm based on a modification of Dijkstra's algorithm in which edge weights are history-dependent. Our work is related to the second approach. As mentioned in the Introduction, we assume that AGVs cannot move freely within their environment and are instead required to move along predetermined paths that connect fixed operating points. These may be associated to shelf locations, where packages are stored or retrieved, to the end of production lines, where AGVs pick up final products, and to additional intermediate locations, used for routing. All these points are formally represented as nodes of a graph, whose arcs represent connecting paths. If AGVs are not subject to acceleration and speed constraints, the minimum-time planning problem is equivalent to SPP and can be solved by Dijkstra algorithm or its variants: see, for instance, [16, 17, 18], or other algorithms such as A* [19], Lifelong planning A* [20], D* [21], and D* Lite [22]. However, since the motion of AGVs must satisfy constraints on maximum speed and tangential and transversal accelerations, that depend on the vehicle position on the path, these approaches cannot be applied to solve BASP.

Instead, various works consider the minimum-time speed planning problem with acceleration and speed constraint on an *assigned* path. For instance, one can use the methods presented in [23, 24], or path-following techniques such as [25, 26].

As said, despite the fact that a large literature exists on SPP and on the minimum-time speed planning on an assigned path, to the authors' knowledge, BASP has never been specifically addressed in the literature. Formally, BASP can be framed as an optimal control problem for a switching system, in which switchings are associated to passages from arc to arc and each discrete state is associated with a specific set of constraints.

The results presented in this Chapter exploit the very specific structure of BASP and cannot be applied to generic switching systems. Anyway, Algorithm 1.6.2 could still apply to other switching systems satisfying an analogous of Proposition 1.6.3 and identifying a class of such systems could be the topic of future research.

This Chapter is structured as follows. In Section 1.1 we recall the definition of shortest path problem and list some variants already known in the literature. In Section 1.2 we describe and motivate the BASP.

In Section 1.3 we discuss how to compute optimal speed profiles along a single arc (with given initial and final speeds), and along a fixed path.

In Section 1.4 we present two novel complexity results. More precisely, in Section 1.4.1 we prove that BASP is NP-hard, while in Section 1.4.2 we prove that, under the assumption of integer data, BASP admits a pseudo-polynomial time algorithm. In Section 1.5 we present an ε-approximation algorithm, based on the discretization of the admissible speeds at the nodes of the graph.

In Section 1.6 we consider a subclass of BASP, called k-BASP, that can be solved with polynomial time-complexity for fixed values of k. Roughly speaking, a BASP instance is a k-BASP, with $k \in \mathbb{N}$, if the maximum number of nodes of a path that can be traveled with a speed profile of maximum acceleration, followed by one of maximum deceleration, starting and ending with null speed, without violating the maximum speed constraint, is smaller than k. k-BASP has some similarities with the problem discussed in [15]. Indeed, in both problems, the incremental for adding an edge to a path does not depend on the complete path, but only on the k last visited nodes. However, the problem addressed by [15] is different from BASP. Indeed, the goal of the problem in [15] is to obtain a feasible path taking into account the vehicle maximum curvature radius. On the other hand, in our work, we focus on selecting the optimal path among a set of possible paths which are already known to be feasible, while, at the same time, obtaining the optimal speed profile. Moreover, we do not assume that the incremental cost depends only on the last k visited node, but present conditions on problem data under which this property holds. Since constant k is problem-dependent and is not known in advance, in Section 1.6.1 we present an adaptive A* algorithm to find k, which computes the optimal trajectory between a pair of nodes and adaptively determines the value of k.

Finally, in Sections 1.7 we present some computational experiments, comparing the ε-approximation algorithm with the adaptive A* algorithm.

1.1 The Shortest Path Problem

The Shortest Path Problem (SPP in what follows) is one of the best known problems within the field of combinatorial optimization. Let $G = (V,E)$ be a directed graph and c_{ij} be the cost of an arc $(i,j) \in E$. Let $n = |V|$ and $m = |E|$. Let $o,d \in V$, $o \neq d$, be an origin and a destination node, respectively, and let $\mathscr{P}_{od}$ be the set of all directed paths in G from o to d. Each $\pi \in \mathscr{P}_{od}$ is a subset of E made up of adjacent arcs, the first one starting at o and the last one ending at d. In the SPP the minimum cost path between o and d is searched for, that is, formally:

$$\min_{\pi \in \mathscr{P}_{od}} \sum_{(i,j)\in p} c_{ij}\,.$$

It is well known that SPP is solvable in polynomial time for nonnegative costs, in particular with $O(m + n\log(n))$ operations by the Fibonacci heap implementation of Dijkstra's algorithm, and also if no negative cycle exists, for instance by the Bellman-Ford algorithm. In both these cases, one can restrict the search of optimal solutions to elementary paths (paths with no node repetitions). If negative cycles are present, then the problem has been proved to be NP-hard.

After the introduction of SPP, many variants have been proposed, together with the related complexity results and solution approaches. In what follows, we discuss some of these variants, warning the reader that the literature about such variants is so vast that the list is certainly incomplete.

Some variants do not add further input data with respect to those of SPP, but change the objective function. Many of these variants are reviewed in [27]. These include *bottleneck* SPP, where the objective function is the largest cost of an arc along the path, that is, $\max_{(i,j)\in\pi} c_{ij}$, *balanced* SPP, where the objective function is the difference between the largest and smallest costs of arcs along the path, that is, $\max_{(i,j)\in\pi} c_{ij} - \min_{(i,j)\in\pi} c_{ij}$, *minimum deviation* SPP, where the objective function is the sum of the difference between the largest cost among all arcs of the path and the cost of each arc along the path, that is, $\sum_{(h,k)\in\pi}[\max_{(i,j)\in\pi} c_{ij} - c_{hk}]$, *k-sum* SPP, where the objective function

is the sum of the k largest costs along the path. While some of these problems are solvable in polynomial time, such as bottleneck SP (see [28, 29]), some others are NP-hard, such as some variants of k-sum SPP (see [30]). In [31], the SPP with forbidden paths is addressed, where the shortest path is searched for with the additional constraint that some sub-paths cannot be part of feasible solutions. In k-SPP not only the shortest path is searched for, but also all paths from the second shortest up to the k-th shortest one (see, for instance, [32] and references therein).

Other variants add additional input values to the description of the problem. For instance, in resource constrained SPPs, given resources $k \in \{1,\dots,K\}$, each with a limited availability γ_k and a consumption value r_{ij}^k along each arc, we search for a minimum cost path π which satisfies all the resource constraints

$$(\forall k \in \{1,\dots,K\}) \sum_{(i,j)\in\pi} r_{ij}^k \leq \gamma_k.$$

Resource constrained SPPs have been proved to be NP-hard (see, for instance, [33, 34]). A detailed discussion of these problems and the related solution approaches can be found in [35].

Another interesting variant of SPP, is *time-dependent* SPP. Here, the cost associated to an arc is the time needed to traverse the arc, and such time depends on the departure time along the arc. Such variant is particularly important for road networks, where the time needed to traverse an arc varies according to traffic conditions. In this case, given a time horizon T, the cost associated to an arc is not a fixed value but is a function $c_{ij} : [0,T] \to \mathbb{R}^+$, where $c_{ij}(t)$ is the time needed to traverse arc (i,j) when departing from i at time t. In these problems the first-in first-out (FIFO) property is usually assumed. This states that along any arc (i,j) an earlier arrival at node j can never be attained by a later departure at node i (that is, for $t' > t$ it always holds that $t' + c_{ij}(t') > t + c_{ij}(t)$). Many variants of this problem are presented, for instance, in [16]. In [36], it is shown that a variant where there is a constraint on the waiting time at each node is NP-hard. Further variants are discussed in [17]. In these variants a penalty or a limit is imposed on the total waiting time spent at a given subset of the

nodes. It is proved that some variants are polynomially solvable, while some others are NP-hard depending on the subset of nodes, on the fact that a penalization or a limit is imposed, and on the magnitude of the penalty parameter or of the waiting limit parameter.

Another interesting variant is SPP with time windows. In this case we associate to each node $i \in V$ an interval $[a_i, b_i]$, and to each arc (i, j) a time t_{ij} to traverse it. Only paths in $\mathscr{P}_{od}$ where each node of the path is visited within the allowed time window are feasible. The case where only elementary paths are feasible has been proven to be strongly NP-hard in [37]. The problem is still NP-hard if we remove such restriction and, thus, if we allow multiple visits to the same node. However, in such case pseudo-polynomial time algorithms have been proposed (see, for instance, [38]). A variant with additional costs associated to the departure times at the different nodes of the path is studied in [18]. In fact, a polynomial time algorithm exists if the FIFO property holds (see [39]). Moreover, if waiting is allowed at a node, then each instance for which the FIFO property does not hold can be transformed into an equivalent one for which the property holds, thus making the problem solvable in polynomial time (see [40]).

1.2 Problem description and motivation

In this section, we describe a new variant of SPP. The following values are associated to each arc $(i, j) \in E$:

- the length ℓ_{ij};
- the maximum speed $v_{ij}^{\max}$ which can be reached along the arc, and the associated maximum squared speed $w_{ij}^{\max} := (v_{ij}^{\max})^2$;
- the minimum acceleration (or maximum deceleration) $a_{ij}^{\min} < 0$ and maximum acceleration $a_{ij}^{\max} > 0$ along the arc.

We denote with α^-, α^+ and μ^+ the acceleration and speed constraints on graph $G = (V,E)$, defined as follows

$$\alpha^-((i,j)) = a_{ij}^{\min}, \qquad \alpha^+((i,j)) = a_{ij}^{\max} \qquad \mu^+((i,j)) = w_{ij}^{\max}. \tag{1.1}$$

These functions belong to $\mathscr{E} = \{\varphi : E \to \mathbb{R}\}$. In general, if $\varphi \in \mathscr{E}$, $\theta \in E$, $\varphi(\theta)$ denotes the value of φ on edge θ. Given a path $\pi = i_1 \to \cdots \to i_m$, we associate to each $\varphi \in \mathscr{E}$ a function $\varphi_\pi : [0, \ell(\pi)] \to \mathbb{R}$ in the following way. Define functions $\Theta : [0,\ell(\pi)] \to \mathbb{N}$, $\Lambda : [0,\ell(\pi)] \to \mathbb{R}$ such that $\Theta(\lambda) = \max\{k \in \mathbb{N} \mid \ell(i_1 \cdots i_k) \leq \lambda\}$ and $\Lambda(\lambda) = \ell(i_1 \cdots i_{\Theta(\lambda)})$. In this way, $\Theta(\lambda)$ is such that $\theta(\lambda) = (i_{\Theta(\lambda)}, i_{\Theta(\lambda)+1})$ is the edge that contains the position at arc-length λ along π and $\Lambda(\lambda)$ is the sum of the lengths of all arcs up to node $i_{\Theta(\lambda)}$ in π. Then, we define $\varphi_\pi(\lambda) = \varphi(\theta(\lambda))$.

Moreover, a zero initial speed is assigned to node o and a zero final speed is assigned to node d. We would like to select a path $\pi \in \mathscr{P}_{od}$ minimizing the time needed to run along the path by fulfilling the maximum speed, the maximum and minimum acceleration constraints along the arcs, and the boundary zero conditions on o and d. Note that SPP is a special case of this problem. Indeed, SPP turns out to be equivalent to the case $a_{ij}^{\max} = +\infty$ and $a_{ij}^{\min} = -\infty$ for all arcs $(i,j) \in E$. In this case, the speed can be changed instantaneously, so that the running time along an arc is minimized by traversing it at the maximum allowed speed along the arc. Therefore, the minimum time to traverse arc (i,j) is $c_{ij} = \frac{\ell_{ij}}{v_{ij}^{\max}}$ and the problem becomes a standard SPP with such costs c_{ij} associated to the arcs. Since SPP corresponds to the case of *unbounded* acceleration limits, the proposed variant of SPP is also called Bounded Acceleration SP (BASP in what follows). In the search for an optimal solution of BASP, we should not only search for an optimal path in $\mathscr{P}_{od}$, but we should also define the speed profile along such path. As we will see later on in Section 1.3, the minimum-time speed profile along an arc (i,j) is fully determined by the initial speed v_i at node i and by the final speed v_j at node j. Thus, the speed v_i at each node i traversed by a path $\mathscr{P}_{od}$ is also part of the decision process. Such speeds must fulfill a continuity constraint, that is, if arc (i,j) is followed by arc (j,k) along the path, the final speed along arc (i,j) must be equal to the initial speed along arc (j,k). Note that

the speeds at the origin node o and at the destination node d are fixed in advance.

Before proceeding, we further motivate the interest for the BASP variant of SPP. As previously mentioned, the interest comes from an industrial application. In automated warehouses, an AGV is required to pick some good up at some point of the warehouse and deliver it at some other point. The AGV moves along predefined paths and is allowed to choose among different routes at some exchange points. Formally, the exchange points as well as the points where goods are picked up and delivered represent the nodes of the graph, while the predefined routes correspond to the arcs of the graph (see, for instance, Figures 3.14 and 1.17). Speed and acceleration limits differ across the different routes. For instance, if a route is a straight line, its maximum speed is higher than the maximum speed allowed along a curved route. Moreover, different speed limits may also be imposed at different points of the warehouse. For instance, if a route lies in a part of the warehouse where also human operators are working, then, for safety reasons, a lower speed limit along this route is imposed with respect to another route lying in a part of the warehouse where human operators are not allowed to work. Due to various reasons, also acceleration bounds may differ from arc to arc. For instance, in the same warehouse, flooring materials can vary from location to location. To avoid wheel slipping, we need to set maximum acceleration bounds depending on the floor frictional force, that varies according to the flooring material. Moreover, the presence of ramps along some arcs implies different acceleration and deceleration bounds. Finally, to reduce lateral oscillations, we should impose lower acceleration bounds along high-curvature connecting paths.

Remark 1.2.1. *In the problem description we imposed a constant speed limit $v_{ij}^{\max}$ along each arc (i,j). In a more realistic setting the speed limit should be a function of the position along a given route:*

$$v_{ij}^{\max} : [0, \ell_{ij}] \to \mathbb{R}^+.$$

For each $s \in [0, \ell_{ij}]$, $v_{ij}^{\max}(s)$ is the maximum allowed speed at position s along the route represented by arc (i,j). In particular, along a curve the speed limit varies with the curvature ray at each position along the curve. For ease of exposition, we only

discuss the case of a constant speed limit along an arc (although different from arc to arc). However, at the cost of some additional technicalities, the following discussion can be extended also to the more realistic case of variable speed limits along each arc.

Remark 1.2.2. *In a real industrial scenario, the AGV may encounter moving obstacles, such as human operators and/or other AGVs. For safety, an AGV typically halts or slows down if it perceives the presence of a human operator. When the obstacle is no longer sensed, the AGV can compute a new motion by solving a new instance of BASP, starting from its current location.*

In order to better appreciate the difference between SPP and BASP we can also consider the example illustrated in Figure 1.2. While path $o \to i_1 \to i_2 \to i_3 \to i_4 \to i_5 \to d$ is shorter than path $o \to j_1 \to j_2 \to d$, the latter is faster in view of the lower number of curves along the path.

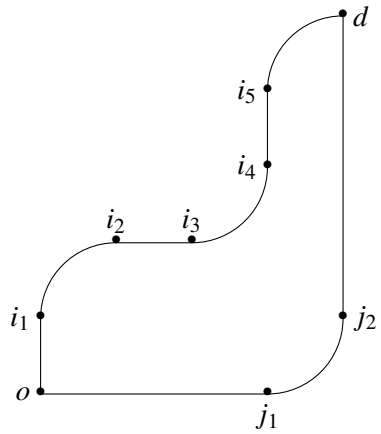


Figure 1.2: Two distinct paths from node o to node d: the shorter path is not the faster one.

1.3 Minimum traveling time along arcs and paths

As previously pointed out, for a given arc (i, j), we are able to compute the maximum speed at which we can traverse the arc and, consequently, the minimum time needed

to traverse it, as soon as we know the initial and final speeds v_i and v_j. Indeed, the maximum squared speed at each position $s \in [0, \ell_{ij}]$ along the arc is given by the following piecewise-linear function:

$$w_{ij}(s; w_i, w_j) = \min\left\{w_{ij}^{\max}, w_i + 2a_{ij}^{\max} s, w_j + 2a_{ij}^{\min}(s - \ell_{ij})\right\}, \tag{1.2}$$

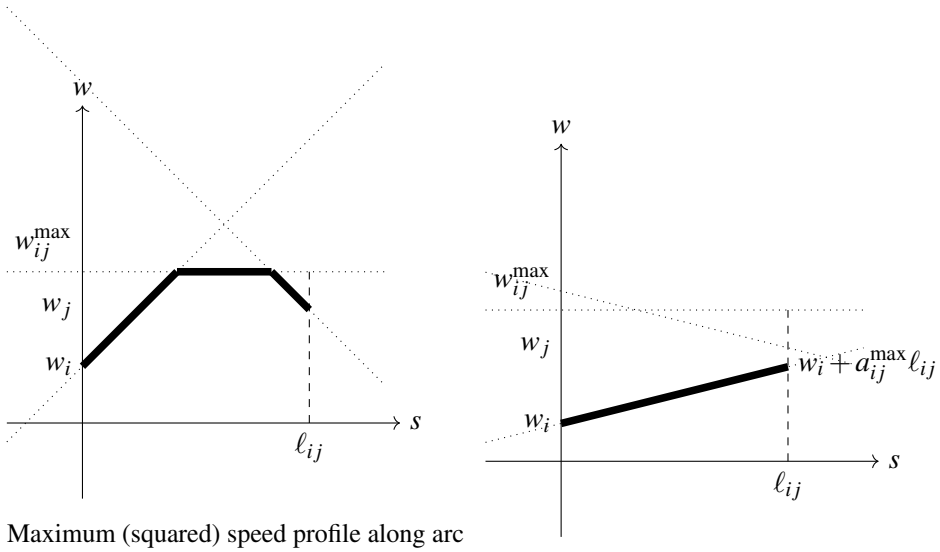
where, as defined above, w denotes the squared speed (so $w_i = v_i^2$, $w_j = v_j^2$, and so on). The result is illustrated in Figure 1.3a. Starting at node i with squared speed w_i, the speed is increased with the maximum possible acceleration $a_{ij}^{\max}$, until the maximum allowed squared speed $w_{ij}^{\max}$ along the arc is reached. Such maximum speed is maintained as long as possible (null acceleration) and, finally, the speed is decreased with the maximum deceleration $a_{ij}^{\min}$ in order to reach squared speed w_j at node j. Note that it might be possible that the maximum speed along the arc is not reachable. In such case, first the speed is increased with maximum acceleration $a_{ij}^{\max}$ and then decreased with maximum deceleration $a_{ij}^{\min}$ to reach the final squared speed w_j (no constant speed portion is present in the maximum speed profile). Consequently, the minimum time to traverse arc (i, j) is the following function of the two (squared) speeds w_i and w_j

$$c_{ij}(w_i, w_j) = \int_{s=0}^{\ell_{ij}} \frac{ds}{\sqrt{w_{ij}(s; w_i, w_j)}}, \tag{1.3}$$

whose solution can be derived in closed form. It is worthwhile to remark that w_i and w_j cannot be arbitrary values. Indeed, besides the obvious constraints $w_i, w_j \leq w_{ij}^{\max}$, it must also hold that

$$w_i + 2a_{ij}^{\max}\ell_{ij} \geq w_j, \qquad w_j - 2a_{ij}^{\min}\ell_{ij} \geq w_i. \tag{1.4}$$

If, for instance, the first inequality is not fulfilled, then it is not possible to reach the squared speed w_j at node j by starting with squared speed w_i at node i even by accelerating as much as possible (acceleration $a_{ij}^{\max}$). This is illustrated in Figure 1.3b. In this case we set $c_{ij}(w_i, w_j) = +\infty$. While we reported above the formula for the minimum time needed to traverse an arc (i, j), given the initial and final squared speeds w_i and w_j, we further point out that a slightly more complicated formula can be derived to compute the minimum time to traverse a full path in $\mathscr{P}_{od}$. In particular,



(a) Maximum (squared) speed profile along arc (i, j) with initial and final squared speed equal to w_i and w_j, respectively.

(b) A case where the squared speeds w_i and w_j are not feasible.

Figure 1.3: Squared speed profiles along an arc.

let $i_0 = o \to i_1 \to \ldots \to i_{p-1} \to i_p = d$ be a path of length p in $\mathscr{P}_{od}$. Let $s_0 = 0$ and $s_h = \sum_{r=1}^{h} \ell_{i_{r-1}i_r}$ be the overall length of the first h arcs of the path, $h \in \{1,\ldots,p\}$. Next, let us define recursively a function F as follows

$$\begin{aligned}
&(\forall s \in [s_0, s_1])\ F(s) = \min\left\{w_{i_0 i_1}^{\max}, 2a_{i_0 i_1}^{\max} s\right\},\\
&(\forall h \in \{1,\ldots,p-1\})\ (\forall s \in [s_h, s_{h+1}])\ F(s) = \min\left\{w_{i_h i_{h+1}}^{\max}, F(s_h) + 2a_{i_h i_{h+1}}^{\max}(s - s_h)\right\},
\end{aligned} \tag{1.5}$$

and a further function B as follows

$$\begin{aligned}
&(\forall s \in [s_{p-1}, s_p])\ B(s) = \min\left\{w_{i_{p-1} i_p}^{\max}, 2a_{i_{p-1} i_p}^{\min}(s - s_p)\right\},\\
&(\forall h \in \{1,\ldots,p-1\})\ (\forall s \in [s_{h-1}, s_h])\ B(s) = \min\left\{w_{i_{h-1} i_h}^{\max}, B(s_h) + 2a_{i_{h-1} i_h}^{\min}(s - s_h)\right\}.
\end{aligned} \tag{1.6}$$

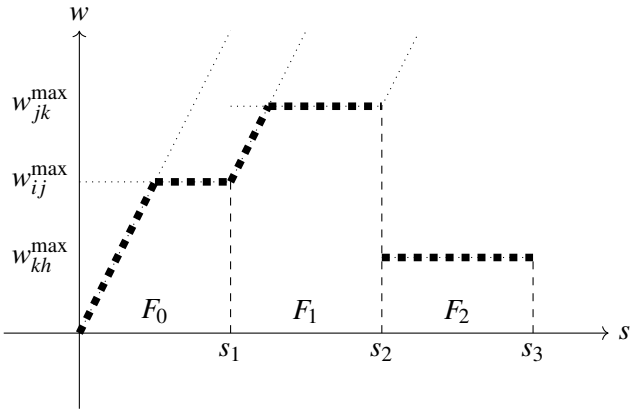
Then, it can be proved (see, for instance, [41] for a proof under more general assumptions) that the optimal (squared) speed profile is

$$(\forall s \in [s_0, s_p])\ W(s) = \min\{F(s), B(s)\}, \tag{1.7}$$

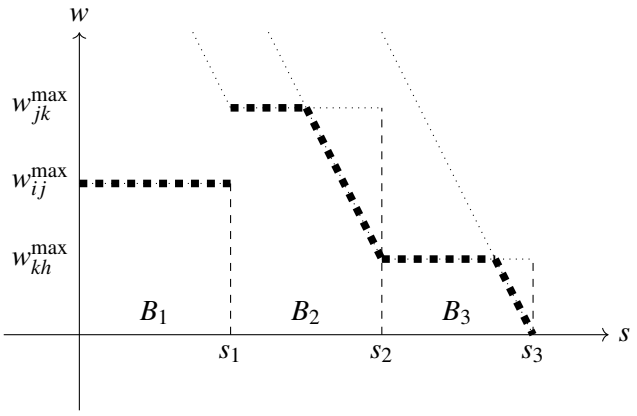
so that the minimum time to travel along the given path is

$$\int_{s=0}^{s_p} \frac{ds}{\sqrt{W(s)}}.$$

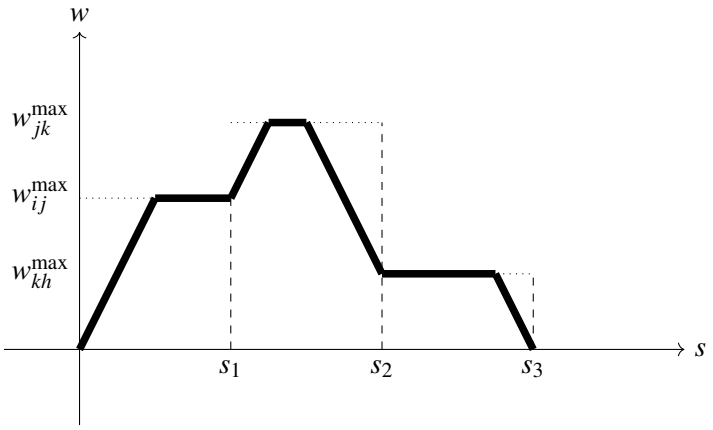
The result is illustrated in Figures 1.4a–1.4c. In Figure 1.4a we notice that F_0 (function F over the interval $[s_0, s_1]$) is obtained by starting at node o with zero speed and increasing the speed with maximum acceleration $a_{ij}^{\max}$ until the maximum squared speed $w_{ij}^{\max}$ is reached, and then keeping this speed until the end of the arc (as it is the case in the figure) or, alternatively, until the end of the arc is reached (in which case the maximum squared speed $w_{ij}^{\max}$ is not reached). Next, F_1 (function F over the interval $[s_1, s_2]$) is obtained similarly but with initial speed $F(s_1)$, while F_2 (function F over the interval $[s_2, s_3]$) is constant and equal to the maximum squared speed $w_{kh}^{\max}$ since $F(s_2) > w_{kh}^{\max}$. In Figure 1.4b we notice that B_3 (function B over the interval $[s_2, s_3]$) is obtained in a way completely similar to F_0. Moving backward, we start from the final node $d \equiv h$ with squared speed $w_h = 0$, and we increase the speed with the maximum deceleration value $a_{kh}^{\min}$ until either we reach the maximum allowed squared speed $w_{kh}^{\max}$ along the arc, in which case we keep such speed until the beginning of the arc at node k, or we reach the beginning of the arc without reaching the maximum speed along the arc (in the figure we are in the first situation). Function B_2 (function B over the interval $[s_1, s_2]$) is obtained similarly but with initial speed at node k equal to $B(s_2)$. Finally, B_1 (function B over the interval $[s_0, s_1]$) is constant and equal to the maximum squared speed $w_{ij}^{\max}$ since $B(s_1) > w_{ij}^{\max}$. The optimal (squared) speed profile is illustrated in Figure 1.4c and is obtained as the point-wise minimum of the functions F and B. While Figures 1.3a–1.3b and Figures 1.4a–1.4c give an intuitive illustration of the optimal speed profiles both for the case of a single arc (i, j) and for the case of a full path from o to d, we point out that these results can be derived as special cases of a more general result presented in [41] for problems with upper speed limits depending on the position along the arcs, as discussed in Remark 1.2.1.



(a) Function F along the path $i \to j \to k \to h$.



(b) Function B along the path $i \to j \to k \to h$.



(c) Maximum (squared) speed profile along the path $i \to j \to k \to h$.

Figure 1.4: Construction of a maximum (squared) speed profile along a path.

We also make the following remark which states two properties of the optimal speed profile and will be useful later on.

Remark 1.3.1. *Let $W(s)$ be the optimal squared speed profile along a given path $i_0 = o \to i_1 \to \ldots \to i_{p-1} \to i_p = d$ in $\mathscr{P}_{od}$, and let w_{i_h} be the optimal squared speed at some node i_h of the path, $i_h \neq o, d$. Moreover, let*

$$\bar{w}_{i_h} = \min\left\{ \min_{(i_h,j)\in E}\{w_{i_h j}^{\max}, -a_{i_h j}^{\min}\ell_{i_h j}\}, \min_{(k,i_h)\in E}\{w_{ki_h}^{\max}, a_{ki_h}^{\max}\ell_{ki_h}\} \right\}. \tag{1.8}$$

Then,

1. *for each $s \in [s_h, s_{h+1}]$, $W(s) \geq \min\{w_{i_h}, w_{i_{h+1}}\}$;*
2. *$w_{i_h} \geq \bar{w}_{i_h}$.*

According to the previous discussion, the optimal speed profile along a path π from o to d with $|\pi|+1$ nodes (here and in what follows $|\pi|$ denotes the length of path π) is identified once the speeds at nodes of the path are known. Indeed, along any edge (i, j) with given initial and final squared speeds w_i and w_j, the optimal speed profile function is equal to (1.2) and the traveling time is equal to (1.3).

Given $\mu^+, \alpha^+, \alpha^- \in \mathscr{E}$ the speed and acceleration constraints defined as in (1.1), let $\mathbb{B} = (\mu^+, \alpha^-, \alpha^+)$. Given path $\pi \in \mathscr{P}$, if we denote by $\pi(i) \in V$ the node at position i along π, we define

$$\begin{aligned} T_{\mathbb{B}}(\pi) = \min_{\mathbf{w}\in\mathbb{R}^{|\pi|+1}} & \sum_{i=1}^{|\pi|} c_{\pi(i)\,\pi(i+1)}(w_i, w_{i+1}) \\ & 0 \leq w_i \leq \min\{w_{\pi(i-1),\pi(i)}^{\max}, w_{\pi(i),\pi(i+1)}^{\max}\}, \quad i \in \{2,\ldots,|\pi|\}, \\ & w_1 = 0,\, w_{|\pi|+1} = 0, \end{aligned} \tag{1.9}$$

which represents the fastest time for traversing π. In this way, $T_{\mathbb{B}}(\pi)$ is the minimum-time required to traverse path π, respecting the speed and acceleration constraints defined in $\mathbb{B}$. We denote by $\mathbf{w}^*(\pi)$ the optimal solution of this problem and in Appendix A.2 we will describe a recursive procedure to find it, similar to the one employed to define the optimal speed profile function W. According to the discussion

above, the problem of finding the speed law which guarantees to traverse a *fixed* path from o to d at a minimum time, under speed and acceleration constraints, is easily solvable even in closed form. But our aim is to compute the minimum time to move from o to d by searching within *all* paths in $\mathscr{P}_{od}$. This is the BASP problem:

$$\pi^* = \arg\min_{\pi \in \mathscr{P}_{od}} T_{\mathbb{B}}(\pi). \tag{1.10}$$

As we will see in Section 1.4, this problem turns out to be NP-hard.

The following properties are a direct consequence of the definition of $T_{\mathbb{B}}(\pi)$.

Proposition 1.3.1. *The following properties hold:*

1. *Let* $\pi_1, \pi_2 \in \mathscr{P}$, $\pi_1\pi_2 \in \mathscr{P} \Rightarrow T_{\mathbb{B}}(\pi_1\pi_2) \geq T_{\mathbb{B}}(\pi_1) + T_{\mathbb{B}}(\pi_2)$.
2. *If* $\mathbb{B} = (\mu^+, \alpha^-, \alpha^+), \hat{\mathbb{B}} = (\hat{\mu}^+, \hat{\alpha}^-, \hat{\alpha}^+)$ *are such that* $(\forall \theta \in E)\ \mu^+(\theta) \leq \hat{\mu}^+(\theta)$ *and* $[\alpha^-(\theta), \alpha^+(\theta)] \subset [\hat{\alpha}^-(\theta), \hat{\alpha}^+(\theta)]$, *then* $(\forall \pi \in \mathscr{P})\ T_{\mathbb{B}}(\pi) \geq T_{\hat{\mathbb{B}}}(\pi)$.

In particular, the first property states that the minimum time for traveling the composite path $\pi_1\pi_2$ is greater or equal to the sum of the times needed for traveling π_1 and π_2 separately. In fact, in the first case, the speed must be continuous when passing from π_1 to π_2 (due to the acceleration bounds), but this constraint does not need to be satisfied when the speed profiles for π_1 and π_2 are computed separately.

1.4 Complexity results

In this section we provide two complexity results for BASP. The first result proves NP-hardness of BASP, while the second proves that BASP admits a pseudo-polynomial time algorithm.

1.4.1 NP-hardness

In this section we prove that, differently from SPP, the BASP variant is NP-hard. We show this by a polynomial reduction of the NP-complete *Partition* problem to

BASP. In the Partition problem, given a set $N = \{1, \dots, n\}$ of positive integer values $\beta_1, \dots, \beta_n$, we would like to establish whether N can be partitioned into two subsets N_1 and N_2 such that $\sum_{i \in N_1} \beta_i = \sum_{i \in N_2} \beta_i = \frac{W}{2}$. Given an instance of the Partition problem we polynomially reduce it to an instance of BASP as follows. Let $G = (V, E)$ be such that:

$$V = N \cup \{0, n+1, n+2\}, \qquad E = \{(i, j) \mid i, j \in V \setminus \{n+2\} \wedge i < j\} \cup \{(n+1, n+2)\}.$$

We set the following lengths for the arcs:

$$\ell_{ij} = \begin{cases} 0 & i = 0 \\ \beta_i & i \in \{1, \dots, n\}, \end{cases}$$

while $\ell_{n+1,n+2} = W^2$. For what concerns the maximum speed values, we set $(\forall (i, j) \in E)\ v_{ij}^{\max} = \sqrt{W}$, while we set the maximum acceleration $a_{ij}^{\max} = 1$ and the minimum acceleration $a_{ij}^{\min} = -1$ for all arcs except $(n+1, n+2)$, while we set $a_{n+1,n+2}^{\max} = 0$ and $a_{n+1,n+2}^{\min} = -\frac{1}{2W}$. Note that, according to the imposed restrictions, $a_{n+1,n+2}^{\max}$ should be strictly larger than 0. However, the result proved with null maximum acceleration can be extended, by continuity, to any sufficiently small and positive maximum acceleration. The origin node o is node 0, with zero speed, while the destination node d is $n+2$, with zero speed. An example of BASP instance derived from the Partition problem with $n = 3$ is illustrated in Figure 1.5. We prove the following.

Proposition 1.4.1. *The optimal value of the BASP instance introduced above is equal to $\sqrt{W} + 2W^{\frac{3}{2}}$ if and only if the partition problem admits a solution and is otherwise larger than such value.*

The proof of Proposition 1.4.1 is presented in Appendix A.1.

Remark 1.4.1. *The complexity result given above shows that BASP is NP-hard even in case all arcs except one share the same acceleration and deceleration bounds. It is still an open question whether NP-hardness still holds if all arcs have the same bound. However, in Section 1.5.1 we will show that optimal solutions for this case are elementary paths (paths with no node repetition). This allows to derive sharper approximation results.*

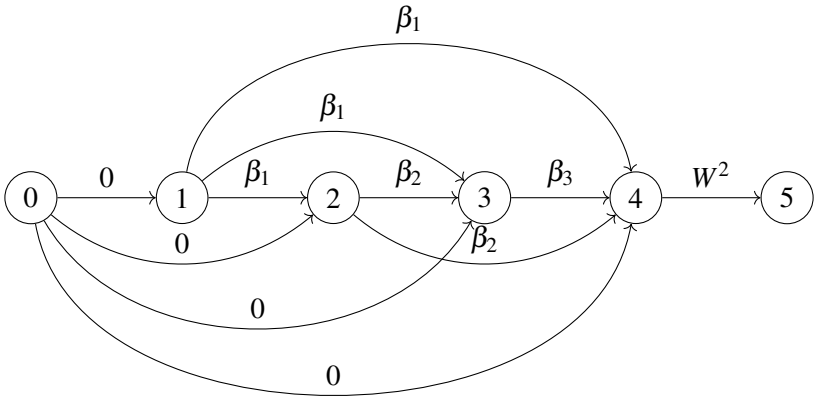


Figure 1.5: An instance of BASP derived from the Partition problem for $n = 3$. Along each arc the maximum allowed speed is $\sqrt{W}$ ($W = \beta_1 + \beta_2 + \beta_3$), $a^{\max}$ is equal to 1 and $a^{\min}$ is equal to -1 along all arcs except $(4,5)$ where $a^{\max}_{45} = 0$ and $a^{\min}_{45} = -\frac{1}{2W}$. Finally, $v_0 = v_5 = 0$.

1.4.2 Pseudo-polynomial algorithm

Although BASP is NP-complete, the following proposition shows it admits a pseudo-polynomial algorithm under the assumption of integer data.

Proposition 1.4.2. *Let us assume that all problem data, ℓ_{ij}, $v^{\max}_{ij}$, $a^{\max}_{ij}$, and $a^{\min}_{ij}$ for all $(i,j) \in E$ are integer values. Then, BASP admits a pseudo-polynomial algorithm.*

Proof. First, we observe that at optimal solutions there is a finite number of speeds which can be reached at each node and the squares of such speeds are integer values. Indeed, the squared speed at some node i is:

- either equal to $w^{\max}_{ij}$, for some j such that $(i,j) \in E$, which is an integer value by assumption;
- or equal to $w^{\max}_{ki}$, for some k such that $(k,i) \in E$, which is again an integer value by assumption;

- or i is the end point of a sub-path $j_0 \to j_1 \to \cdots j_{k-1} \to j_k = i$, with squared speed at node j_1 equal to $w^{\max}_{j_0,j_1}$ and squared speed at node i

$$w_i = w^{\max}_{j_0,j_1} + 2\sum_{h=1}^{k-1} a^{\max}_{j_h,j_{h+1}} \ell_{j_h,j_{h+1}},$$

which is an integer value due to integrality of all the data (see Figure 1.6). Note that j_1 may be the starting node o, in which case $w_{j_1} = 0$ (j_0 is not included in this case);

- or i is the starting point of a path $j_0 = i \to j_1 \to \cdots j_{k-1} \to j_k$, with squared speed at node j_{k-1} equal to $w^{\max}_{j_{k-1},j_k}$ and squared speed at node i

$$w_i = w^{\max}_{j_{k-1},j_k} - 2\sum_{h=0}^{k-2} a^{\min}_{j_h,j_{h+1}} \ell_{j_h,j_{h+1}},$$

which is, again, an integer value due to integrality of all the data (see Figure 1.7). Note that node j_{k-1} may be the destination node d, in which case $w_{j_{k-1}} = 0$ (j_k is not included in this case).

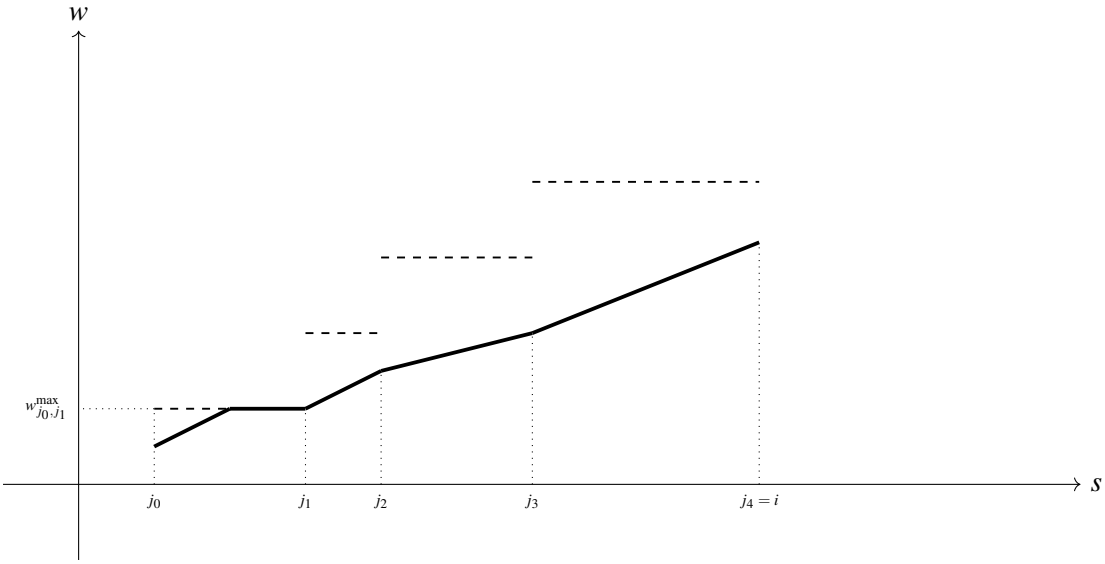


Figure 1.6: Optimal (squared) speed profile from node j_0 up to node i (continuous line) when node i is reached by accelerating as much as possible along all arcs between j_1 and i. The dashed lines represent the maximum squared speeds along the arcs.

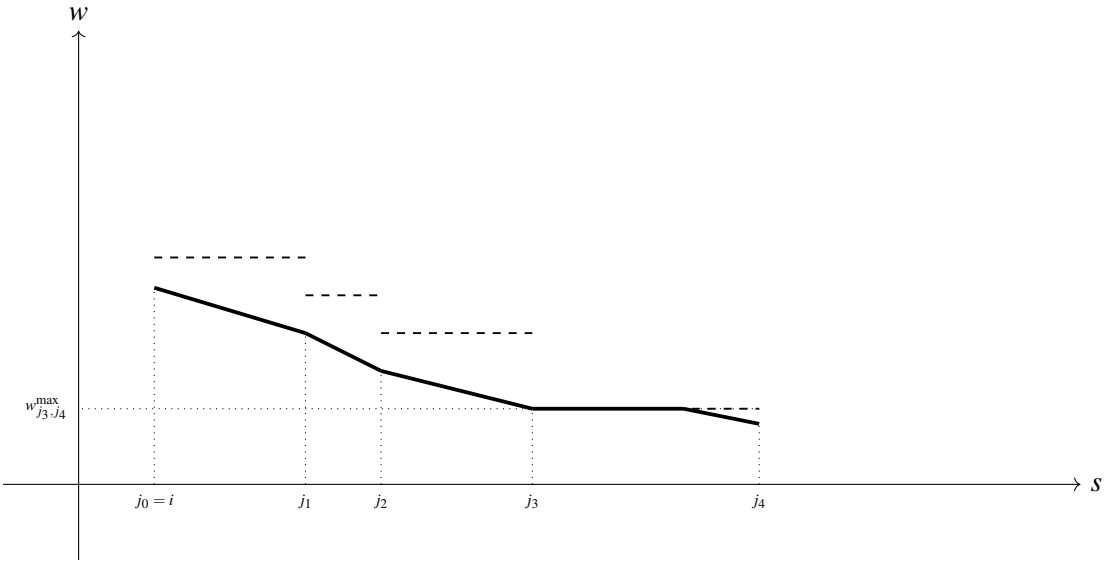


Figure 1.7: Optimal (squared) speed profile from node i up to node j_4 (continuous line) when node j_4 is reached by decelerating as much as possible along all arcs between i and j_4. The dashed lines represent the maximum squared speeds along the arcs.

Thus, the set $\mathscr{W}$ of different possible squared speeds can be taken equal to the set of all integers between 0 and $\bar{W} = \max_{(i,j)\in A} w_{ij}^{\max}$. Now we create a new graph with node set $V \times \mathscr{W}$, that is, each node is a pair made up by a node in V and one of the possible squared speeds in $\mathscr{W}$. Thus, the number of nodes is $\bar{W}|V|$. For what concerns the arc set, in this graph an arc between node (i, w_i) and node (j, w_j) exists if there exists an arc $(i, j) \in E$ and, moreover, $c_{ij}(w_i, w_j) < +\infty$, that is, there exists a feasible speed profile along arc (i, j) with initial squared speed w_i and final squared speed w_j. Then, the number of arcs is limited from above by $\bar{W}^2|E|$. The distance associated to this arc is the minimum time for a path from i to j with the boundary conditions w_i and w_j, which can be easily computed through (1.3), as discussed in Section 1.3 (recall that, in case w_i and w_j are not feasible, as illustrated in Figure 1.3b, then the arc is removed). Then we can solve our problem by applying, for instance, Dijkstra algorithm to this graph. Dijkstra's complexity is $O(m + n\log(n))$ and is, thus, polynomial with respect to the size and the data of the original problem, which proves

pseudo-polynomiality. □

Remark 1.4.2. *While Proposition 1.4.2 has been proved under the assumption of integer data, it can also be extended to rational data. In such case the squared speeds which can be reached by optimal solutions are not integer values but are multiple of a rational number $\frac{1}{t}$, where t depends on the problem data. Of course, the size of the extended graph increases with t. The approximation algorithm discussed in the following Section 1.5 is motivated by the need to consider a discretization step larger than $\frac{1}{t}$ in order to have a graph of manageable size.*

1.5 Approximation algorithm

In this section we present an approximation algorithm for BASP with a complexity that is polynomial with respect to the size and the data of the original problem and the inverse of the approximation factor. The idea is to discretize the squared speeds in order to obtain a finite set of possible squared speeds at each node of the graph. In this way, imposing that the initial and final squared speeds along each arc belong to the discretized set of squared speeds, the set of possible speed profiles over each arc becomes finite. Hence, we can define an extended graph that enables us to solve this discretized version of the problem by means of Dijkstra's algorithm. Differently from Proposition 1.4.2, here arc lengths, accelerations and speed bounds need not be integer values (actually, the approach could also be extended to the case of non-constant speed bounds along arcs as discussed in Remark 1.2.1). In this case, we just discretize the squared speeds and impose the additional constraint that the squared speeds at the beginning and at the end of each arc belong to the set of discretized squared speeds. Let

$$\Omega_h = \{\omega \in [0, \bar{W}] \mid (\exists k \in \mathbb{N})\ \omega = kh\},$$

with $\bar{W} = \max_{(i,j)\in A} w_{ij}^{\max}$, be the set of discretized squared speeds with discretization step h. Then, $|\Omega_h| = \lceil \bar{W}/h \rceil$. The discretized problem is defined over a graph that extends graph G of the original problem. Namely, the extended graph $G' = (V', E')$ is

defined as follows:

$$V' = \left\{ (i,\omega) \mid i \in V \ \wedge \ \omega \in \Omega_h \ \wedge \ \omega \leq \max \left\{ \max_{j\,|\,(i,j)\in E} w_{ij}^{\max}, \max_{k\,|\,(k,i)\in E} w_{ki}^{\max} \right\} \right\},$$

where we notice that we bound from above the possible squared speeds at node i by the maximum squared speeds along the incoming and outgoing arcs of node i, while

$$E' = \{((i,\omega_i),(j,\omega_j)) \in V' \times V' \mid (i,j) \in E \ \wedge \ \omega_i, \omega_j \leq w_{ij}^{\max} \ \wedge \ c_{ij}(\omega_i,\omega_j) < +\infty\},$$

where we recall that $c_{ij}(\omega_i,\omega_j) = +\infty$ means that no feasible profile is able to travel from i to j with initial and final squared speeds equal to ω_i and ω_j, respectively, while $c_{ij}(\omega_i,\omega_j) < +\infty$, as defined in (1.3), is the optimal travel time along arc (i,j) with the given initial and final squared speeds.

Remark 1.5.1. *The cost for constructing the extended graph G' is $O(|E| \cdot |\Omega_h|^2)$. Indeed, the number of arcs of the extended graph $|E'|$ is bounded from above by $|E| \cdot |\Omega_h|^2$ and the cost for checking whether an arc exists or not in the extended graph, that is the cost for checking conditions* (1.4)*, is constant.*

Once the extended graph has been defined, the proposed approximation algorithm is nothing but the application of Dijkstra's algorithm to solve the discretized problem. More precisely, we search for the shortest path connecting nodes $(o,0),(d,0) \in V'$ over graph G', where the cost of arc $((i,\omega_i),(j,\omega_j)) \in E'$ is equal to the value $c_{ij}(\omega_i,\omega_j) < +\infty$ defined in (1.3). Then, we have the following complexity result for the approximation algorithm.

Proposition 1.5.1. *The complexity of the approximation algorithm is*

$$O\left(m \left(\frac{\bar{W}}{h} \right)^2 + \frac{n\bar{W}}{h} \log \left(\frac{n\bar{W}}{h} \right) \right).$$

Proof. The approximation algorithm is Dijkstra's algorithm applied on the extended graph G', so that its complexity is $O(|E'| + |V'| \log |V'|)$. Then, the result immediately follows by observing that $|V'| \leq |V||\Omega_h| = n\lceil \bar{W}/h \rceil$ and $|E'| \leq |E||\Omega_h|^2 \leq m(\lceil \bar{W}/h \rceil)^2$. □

As a next step, we want to obtain an estimate of the absolute error in terms of travel time of the discretized solution returned by the approximation algorithm with respect to the continuous solution of the original BASP problem. To this end, let us consider the optimal path $\pi^* \in \mathscr{P}_{od}$

$$o \to i_1 \to i_2 \to \cdots \to d,$$

of the original BASP problem, with the corresponding squared speeds at each node $\{w_o = 0, w_{i_1}, w_{i_2}, \ldots, w_d = 0\}$. Our aim is to build a feasible solution of the discretized problem traversing the same arcs as path π^* and whose speed profile is not above the optimal speed profile of the original BASP problem but is *as close as possible* to it. Building such solution requires some care. It is tempting to proceed as follows: for each node i in the optimal path π^*, with squared speed w_i in the optimal speed profile of BASP, replace w_i with

$$\omega_i = \max\{kh \mid kh \leq w_i,\ k \in \mathbb{N}\}, \tag{1.11}$$

that is, with the largest discretized speed bounding from below w_i. Unfortunately, this does not work. Indeed, let us consider some arc $(i, j) \in \pi^*$ with the related optimal squared speeds w_i and w_j, and let ω_i and ω_j be chosen as in (1.11). Unfortunately, in the extended graph, arc $((i, \omega_i), (j, \omega_j))$ may not exist, since a situation like the one displayed in Figure 1.3b may occur. Formally, according to (1.4), it may happen that either $\omega_i + 2a_{ij}^{\max}\ell_{ij} < \omega_j$ or $\omega_j - 2a_{ij}^{\min}\ell_{ij} < \omega_i$. Therefore, we need to proceed differently to build a solution of the discretized problem starting from the optimal solution of BASP. For $x, h \in \mathbb{R}$, $h > 0$, we denote by $\langle x \rangle = \max\{i \in \mathbb{Z} \mid ih \leq x\}$ the maximum multiple of h lower than or equal to x. First, we reformulate the discretized problem as follows. Let $\mathbb{B} = (\mu^+, \alpha^-, \alpha^+)$ be the speed and acceleration constraints defined as in (1.1) and π be an assigned path from o to d. The minimum time $T_{\mathbb{B}}^h(\pi)$

to traverse π in the discretized problem is:

$$
\begin{aligned}
T^h_{\mathbb{B}}(\pi) = \min_{\mathbf{w}\in\mathbb{R}^{|\pi|+1}} & \sum_{i=1}^{|\pi|} c_{\pi(i)\,\pi(i+1)}(w_i, w_{i+1}) \\
& 0 \le w_i \le \min\{w^{\max}_{\pi(i-1),\pi(i)}, w^{\max}_{\pi(i),\pi(i+1)}\}, \quad i \in \{2,\dots,|\pi|\}, \quad (1.12) \\
& w_1 = 0,\, w_{|\pi|+1} = 0\,, \\
& \langle w_i \rangle = w_i\,, \quad i \in \{1,\dots,|\pi|+1\}.
\end{aligned}
$$

For simplicity of notation, from now on we can denote $T_{\mathbb{B}}$ simply as T. Problem (1.12) is obtained by adding to Problem (1.9) the last constraint, imposing that the coordinates of $\mathbf{w}$ have to be multiples of h. We call $\mathbf{w}^h(\pi) \in \mathbb{R}^{|\pi|+1}$ a vector that corresponds to the solution of (1.12). Note that Problem (1.12) always admits at least one feasible solution with finite objective function value, for instance the solution $w_i = 0$ for all $i \in \{1,\dots,|\pi|+1\}$. The solution of discretized BASP corresponds to path

$$
\pi^h = \arg\min_{\pi\in\mathscr{P}_{od}} T^h_{\mathbb{B}}(\pi). \tag{1.13}
$$

Recall that this problem can be solved by the application of Dijkstra's algorithm to the extended graph described above. The following lemma compares the optimal values and solutions of Problems (1.9) and (1.12).

Lemma 1.5.1. *For any path π and any $h > 0$,*

i) $T^h_{\mathbb{B}}(\pi) \ge T_{\mathbb{B}}(\pi)$,

ii) $\mathbf{w}^h(\pi) \le \mathbf{w}^*(\pi)$ (vector inequalities are intended component-wise).

Proof. Statement i) is a consequence of the fact that Problem (1.12) is obtained by adding a constraint to Problem (1.9). For Statement ii), first note that, if $\mathbf{w}_1, \mathbf{w}_2 \in \mathbb{R}^{|\pi|+1}$ are feasible values for $\mathbf{w}$ in Problem (1.9), then also their component-wise maximum $\mathbf{w} = \max\{\mathbf{w}_1, \mathbf{w}_2\}$ is feasible (see, for instance, [41]). By contradiction, if $\mathbf{w}^h(\pi) \not\le \mathbf{w}^*(P)$ then $\mathbf{w} = \max\{\mathbf{w}^h(\pi), \mathbf{w}^*(\pi)\}$ is feasible for Problem (1.9). Note that the objective function $f(\mathbf{w}) = \sum_{i=1}^{|\pi|} c_{\pi(i)\,\pi(i+1)}(w_i, w_{i+1})$ of Problem (1.9) is strictly decreasing, that is, if $\mathbf{w}_1 \ge \mathbf{w}_2$ and $\mathbf{w}_1 \ne \mathbf{w}_2$, then $f(\mathbf{w}_1) < f(\mathbf{w}_2)$. Since $\mathbf{w} \ge \mathbf{w}^*(\pi)$ and $\mathbf{w} \ne \mathbf{w}^*(\pi)$, it follows that $f(\mathbf{w}) < f(\mathbf{w}^*(\pi))$, contradicting the optimality of $\mathbf{w}^*(\pi)$. □

Next, the following lemma, whose proof is given in Appendix A.2, gives an upper bound on the components of the difference vector $\mathbf{w}^*(\pi) - \mathbf{w}^h(\pi)$. Note that such components are nonnegative in view of part ii) of Lemma 1.5.1.

Lemma 1.5.2. *The following holds for all $i \in \{1, \ldots, |\pi| + 1\}$:*

$$w^*(\pi)_i - w^h(\pi)_i \leq h|\pi|.$$

In order to use Lemma 1.5.2, we need to find an upper bound on $|\pi^*|$, the number of arcs of the optimal path. This can be done as follows.

Lemma 1.5.3. *An upper bound for the number of arcs $|\pi^*|$ of the optimal path π^* is given by*

$$|\pi^*| \leq \left(\frac{\ell(\pi_{SPP})}{\sqrt{\bar{w}}} + 2\frac{\sqrt{\bar{w}}}{a_{\min}} \right) \frac{\sqrt{\bar{W}}}{\ell_{\min}}, \tag{1.14}$$

where:

- *π_{SPP} is the shortest path connecting o to d over the original graph, when the cost of each arc $(i, j) \in E$ is equal to ℓ_{ij};*
- *$a_{\min} = \min_{(i,j)\in E} \min\{a_{ij}^{\max}, -a_{ij}^{\min}\} > 0$;*
- *$\bar{w} = \min_{i \in V \setminus \{o,d\}} \bar{w}_i$, where $\bar{w}_i > 0$ is defined in* (1.8)*;*
- *$\bar{W} = \max_{(i,j)\in E} w_{ij}^{\max}$;*
- *$\ell_{\min} = \min_{(i,j)\in E} \ell_{ij}$.*

Proof. A feasible solution for BASP is obtained by the arcs of path π_{SPP}, along which, starting from null speed at node o, we first accelerate with acceleration $a_{\min}$ until we reach speed $\bar{w}$, then we keep the speed constant and, finally, we decelerate with deceleration $a_{\min}$ until we reach the final null speed at node d. The time t_{SPP} to travel along path π_{SPP} with the given speed profile, is an upper bound for the travel time of the optimal path π^*. We have that

$$t_{\text{SPP}} \leq \frac{\sqrt{\bar{w}}}{a_{\min}} + \frac{\ell(\pi_{\text{SPP}})}{\sqrt{\bar{w}}} + \frac{\sqrt{\bar{w}}}{a_{\min}}.$$

Next we need a lower bound for the time needed to travel along any arc $(i,j) \in E$. We denote this time with $t_{\min}$ and a lower bound for it is $\ell_{\min}/\sqrt{\bar{W}}$. Then, the ratio of t_{SPP} to $t_{\min}$ is an upper bound for the number of arcs $|\pi^*|$ in the optimal path, that is,

$$|\pi^*| \leq \frac{t_{\text{SPP}}}{t_{\min}} \leq \left(\frac{\ell(\pi_{\text{SPP}})}{\sqrt{\bar{w}}} + 2\frac{\sqrt{\bar{w}}}{a_{\min}} \right) \frac{\sqrt{\bar{W}}}{\ell_{\min}}.$$

□

In the following, we estimate the difference $T^h(\pi^h) - T(\pi^*) \geq 0$ between the optimal values of the discretized BASP (1.13) and BASP (1.10) (nonnegativity follows form part i) of Lemma 1.5.1). Path π^h, corresponding to the solution of the discretized BASP, can be different from π^* and $T^h(\pi^h) \leq T^h(\pi^*)$. Hence, $T^h(\pi^h) - T(\pi^*) \leq T^h(\pi^*) - T(\pi^*)$. Quantities $T(\pi^*)$ and $T^h(\pi^*)$ correspond to the optimal values of Problems (1.9) and (1.12) on the same path π^*. In order to bound the difference $T^h(\pi^*) - T(\pi^*)$, we need a further lemma, giving, for some arc (i,j), an upper bound for the difference $c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z})$ (that is, the time difference between the times needed to travel arc (i,j) with boundary speeds w,z and $\hat{w},\hat{z}$, respectively, with $w \leq \hat{w}$ and $z \leq \hat{z}$). The lemma will be proved in Appendix A.3.

Lemma 1.5.4. *For any arc (i,j) and any $w,z,\hat{w},\hat{z} \in \mathbb{R}$, with $0 \leq w,z,\hat{w},\hat{z} \leq w_{ij}^{\max}$, $w \leq \hat{w}$, $z \leq \hat{z}$, and $c_{ij}(w,z), c_{ij}(\hat{w},\hat{z}) < +\infty$, the following bound holds:*

$$\left| c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z}) \right| \leq \frac{4\max\{|w-\hat{w}|, |z-\hat{z}|\}}{a_{\min}\sqrt{\bar{w}}},$$

where $a_{\min}$ and $\bar{w}$ are defined in the statement of Lemma 1.5.3.

Then, we can provide an estimate on $T^h(\pi^*) - T(\pi^*)$ (which also bounds $T^h(\pi^h) - T(\pi^*)$) by summing up the contributions of all arcs of π^*.

Proposition 1.5.2. *The following bound holds:*

$$T^h(\pi^h) - T(\pi^*) \leq \left(\frac{\ell(\pi_{SPP})}{\sqrt{\bar{w}}} + 2\frac{\sqrt{\bar{w}}}{a_{\min}} \right)^2 \frac{4\bar{W}}{\ell_{\min}^2 a_{\min}\sqrt{\bar{w}}} h,$$

where π_{SPP} is the shortest path connecting node o to node d.

Proof. First observe that, for any arc $e = (\pi^*(i), \pi^*(i+1))$, it holds that

$$\begin{aligned} &\left|c_e(w_i^h(\pi^*), w_{i+1}^h(\pi^*)) - c_e(w_i^*(\pi^*), w_{i+1}^*(\pi^*))\right| \leq \\ &\leq \frac{4\max\{|w_i^h(\pi^*) - w_i^*(\pi^*)|, |w_{i+1}^h(\pi^*) - w_{i+1}^*(\pi^*)|\}}{a_{\min}\sqrt{\bar{w}}} \leq \frac{4|\pi^*|h}{a_{\min}\sqrt{\bar{w}}}, \end{aligned} \tag{1.15}$$

as a consequence of Lemma 1.5.4 (first inequality) and of Lemma 1.5.2 (second inequality). Then, it follows that

$$\begin{aligned} T^h(\pi^h) - T^*(\pi^*) \leq T^h(\pi^*) - T^*(\pi^*) \leq \sum_{k=1}^{|\pi^*|} \frac{4|\pi^*|h}{a_{\min}\sqrt{\bar{w}}} = \\ = \frac{4|\pi^*|^2 h}{a_{\min}\sqrt{\bar{w}}} \leq \left(\frac{\ell(\pi_{\text{SPP}})}{\sqrt{\bar{w}}} + 2\frac{\sqrt{\bar{w}}}{a_{\min}}\right)^2 \frac{4\bar{W}}{\ell_{\min}^2 a_{\min}\sqrt{\bar{w}}} h, \end{aligned}$$

where the first inequality derives from (1.15), whilst the second one follows from Lemma 1.5.3. □

We can also provide an estimate of the relative error.

Proposition 1.5.3. *Given $\varepsilon \in (0,1)$, the relative error of the approximated problem with $h = C\varepsilon$, where C is a constant that depends on the problem data, is $1+\varepsilon$, that is, $\frac{T^h(\pi^h)}{T(\pi^*)} \leq 1+\varepsilon$.*

Proof. Let $t_{\min}$ be the travel time of the arc of shortest length $\ell_{\min}$ assuming that the squared speed along it is constantly equal to $\bar{W}$ (this is a lower bound for the travel time along any arc). Then, by Proposition 1.5.2, we have the following estimate over the relative error:

$$\begin{aligned} \frac{T^h(\pi^h)}{T(\pi^*)} \leq 1 + \frac{T(\pi^*) - T^h(\pi^h)}{T(\pi^*)} \leq 1 + \frac{T(\pi^*) - T^h(\pi^h)}{t_{\min}} = 1 + (T(\pi^*) - T^h(\pi^h))\frac{\sqrt{\bar{W}}}{\ell_{\min}} \leq \\ \leq 1 + \left(\frac{\ell(\pi_{\text{SPP}})}{\sqrt{\bar{w}}} + 2\frac{\sqrt{\bar{w}}}{a_{\min}}\right)^2 \frac{3\bar{W}^{\frac{3}{2}}}{\ell_{\min}^3 a_{\min}\sqrt{\bar{w}}} h = 1 + \varepsilon, \end{aligned}$$

with

$$h = C\varepsilon, \qquad C = \left[\left(\frac{\ell(\pi_{\text{SPP}})}{\sqrt{\bar{w}}} + 2\frac{\sqrt{\bar{w}}}{a_{\min}}\right)^2 \frac{3\bar{W}^{\frac{3}{2}}}{\ell_{\min}^3 a_{\min}\sqrt{\bar{w}}}\right]^{-1}. \tag{1.16}$$

□

The following theorem states a time-complexity and an error estimate result for the approximated problem.

Theorem 1.5.1. *Given $\varepsilon \in (0,1)$, let $h = C\varepsilon$ with C defined as in* (1.16)*. Then, $\frac{T^h(\pi^h)}{T(\pi^*)} \leq 1+\varepsilon$, that is the solution returned by the approximation algorithm has relative error at most ε, and the approximation algorithm has time-complexity*

$$O\left(\left(\frac{\bar{W}}{C\varepsilon}\right)^2 m + \frac{\bar{W}n}{C\varepsilon}\log\left(\frac{\bar{W}n}{C\varepsilon}\right)\right).$$

Proof. The thesis directly follows from Propositions 1.5.1 and 1.5.3. □

1.5.1 The case of uniform acceleration bounds

As previously mentioned in Remark 1.4.1, it is still unclear whether the case where all arcs share the same acceleration and deceleration bounds, denoted with $a^{\max}$ and $a^{\min}$, respectively, is NP-hard or not. However, we can prove the following result.

Proposition 1.5.4. *If*

$$(\forall (i,j) \in E)\ a_{ij}^{\max} = a^{\max} \ \wedge\ a_{ij}^{\min} = a^{\min},$$

that is, all arcs have the same acceleration and deceleration bounds, then the optimal solution of BASP is an elementary path (a path that does not contain loops).

Proof. Let us consider two distinct paths π and π' from node o to node d such that π is elementary and $\pi \subset \pi'$. Since π' contains the entire path π, it must contain a cycle C, as shown in Figure 1.8.

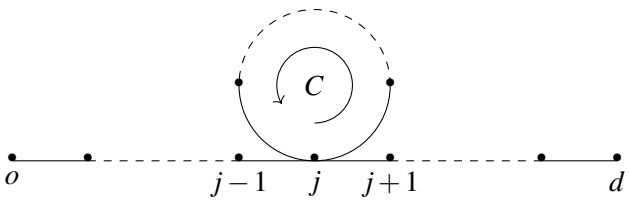


Figure 1.8: Two distinct paths from node o to node d: π and $\pi' = C \cup \pi$.

We will show that the time needed to traverse π is lower than the time needed to traverse π', so that we can restrict the attention to elementary paths. Let j be the first node of cycle C in π' as indicated in Figure 1.8. Let v_j be the initial speed at node j along path π. Along cycle C we can speed up. If, after traveling cycle C, the new speed at node j is $v_C > v_j$, then there is a time gain $\tilde{t}$ in traveling the final sub-path $j \to d$, since the new initial speed at j is higher. However, there is also a loss of time $\bar{t}$ to travel along cycle C, We prove that the latter is always greater than the former. Indeed, a lower bound $\bar{t}_{\min}$ for the time needed to travel along cycle C is obtained by assuming that the maximum acceleration $a^{\max}$ can be kept along the cycle without hitting maximum speed bounds:

$$\bar{t}_{\min} = \frac{v_C - v_j}{a^{\max}}.$$

Now, let ℓ be the length of the sub-path $j \to d$. An upper bound $\Delta t_{\max}$ on the time difference between the time to travel along the sub-path $j \to d$ with starting speed v_j and the time to travel along the same sub-path with starting speed v_C is obtained by assuming that we can keep the maximum acceleration along the sub-path:

$$\Delta t_{\max} = \frac{\sqrt{v_j^2 + \ell a^{\max}} - v_j}{a^{\max}} - \frac{\sqrt{v_C^2 + \ell a^{\max}} - v_C}{a^{\max}}$$

It follows that

$$\bar{t}_{\min} > \Delta t_{\max} \iff \sqrt{v_C^2 + \ell a^{\max}} > \sqrt{v_j^2 + \ell a^{\max}},$$

which holds true in view of $v_C > v_j$.

The same reasoning applies to maximum deceleration $a^{\min}$. Indeed, suppose that along path π the agent has to decelerate before j to reach the speed v_j. In this case, along cycle C we could speed down, allowing to maintain a higher speed along the sub-path $o \to j$. However, it is possible to prove that the lost time $\bar{t}$ to travel along cycle C is always larger than the time gain $\tilde{t}$ in traveling the initial sub-path $o \to j$ with a higher final speed, and so π is faster than π' in any case. □

This result is important because it allows us to state that in this sub-case the upper bound for the number of arcs of the optimal path π^* depends only on the number of

nodes n. In particular

$$|\pi^*| \leq n-1.$$

Therefore, we can use this bound rather than the one derived in Lemma 1.5.3 in all subsequent results. So, for instance, the upper bound in Proposition 1.5.2 can be set equal to $\frac{4n^2h}{a_{\min}\sqrt{\bar{w}}}$.

1.6 The k-BASP

As we will see in Remark 1.6.1, SPP can be viewed as a special case of BASP, namely a BASP with unbounded acceleration limits. In fact, also BASP can be viewed as an SPP but defined on a different graph with respect to the original one. More precisely, here we introduce some restrictions on parameters $\mathbb{B}$ that allow reducing BASP to a standard SPP that can be solved by Dijkstra's algorithm on an extended graph. Let $\pi \in \mathscr{P}$, define
$\ell^+(\pi) = \min\{\{\lambda \in [0,\ell(\pi)] \mid \int_0^\lambda \alpha_\pi^+(q)dq = \mu_p^+(\lambda)\}, +\infty\}$,
$\ell^-(\pi) = \max\{\{\lambda \in [0,\ell(\pi)] \mid -\int_\lambda^{\ell(\pi)} \alpha_\pi^-(q)dq = \mu_p^+(\lambda)\}, -\infty\}$.

In this way, $\ell^+(\pi)$ is the smallest value of $\lambda \in [0,\ell(\pi)]$ for which function F defined in (1.5) starting from initial condition $F(0) = 0$, reaches the squared speed upper bound $\mu^+(\lambda)$. Note that $\ell^+(\pi) = \infty$ if no such value of λ exists. Similarly, $\ell^-(\pi)$ is the largest value of $\lambda \in [0,\ell(\pi)]$ for which function B defined in (1.6) starting from initial condition $B(\ell(\pi)) = 0$, reaches $\mu^+(\lambda)$. Again, $\ell^-(\pi) = -\infty$ if no such value of λ exists. Note that if $\pi, t, \pi t \in \mathscr{P}$, $\ell^+(\pi t) \leq \ell^+(\pi)$ and $\ell^-(\pi t) \geq \ell^-(\pi)$ (actually, equalities hold if the values are all finite). Finally, we define

$$K(\mathbb{B}) = \min\{k \in \mathbb{N} \mid (\forall p \in \mathscr{P}_s)\ |\pi| \geq k \Rightarrow \ell^+(\pi) \leq \ell^-(\pi)\}. \tag{1.17}$$

We call k-BASP any instance of BASP problem that satisfies $K(\mathbb{B}) \leq k$. For instance, consider the following chain graph $G = (V = \{s,1,2,f\}, E = \{(s,1),(1,2),(2,f)\})$. Here $(\forall \theta \in E)\ \alpha^-(\theta) = -1$, $\alpha^+(\theta) = 1$, $\ell(\theta) = 1$, and $\mu^+((s,1)) = 1$, $\mu^+((1,2)) = \frac{2}{3}$, $\mu^+((2,f)) = 1$. In this case, $\mathscr{P}_s = \{s, s1, s12, s12f\}$. Moreover, $K(\mathbb{B}) > 2$, since $\ell^+(s1) = 1 > 0 = \ell^-(s1)$, as reported in Figure 1.9. Further, $\ell^+(s12) <$

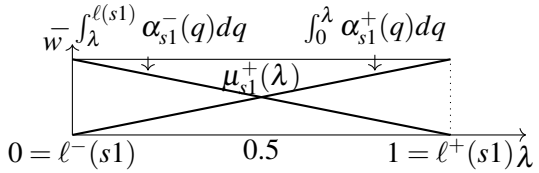


Figure 1.9: Computation of $\ell^+(s1) = 1$, $\ell^-(s1) = 0$.

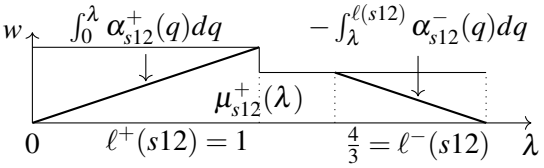


Figure 1.10: Computation of $\ell^+(s12) = 1$, $\ell^-(s12) = \frac{4}{3}$.

$\ell^-(s12)$ and $\ell^+(12f) < \ell^-(12f)$ and $s12, 12f$ are the only paths of length 3. Figure 1.10 shows the computation of $\ell^+(s12)$ and $\ell^-(s12)$; the computation of $\ell^+(12f)$ and $\ell^-(12f)$ is analogous. Hence, in this example, $K(\mathbb{B}) = 3$.

Note that $K(\mathbb{B}) - 1$ represents the maximum number of nodes of a path that can be traveled with a speed profile of maximum acceleration, followed by one of maximum deceleration, starting and ending with null speed, without violating the maximum speed constraint. The following expression provides a simple upper bound on $K(\mathbb{B})$:

$$K(\mathbb{B}) \leq 1 + \left\lceil 2 \max_{\theta \in E} \frac{\mu^+(\theta)}{\left(\alpha^+(\theta) \wedge |\alpha^-(\theta)|\right) \ell(\theta)} \right\rceil. \tag{1.18}$$

Note that $K(\mathbb{B}) = 1$ only if $\alpha^- = -\infty$ and $\alpha^+ = +\infty$, that is, if we do not consider acceleration bounds. We will present an algorithm that solves k-BASP in polynomial time complexity with respect to $|V|$ and $|E|$. However, note that the complexity is exponential with respect to k, so that a correct estimation of $K(\mathbb{B})$ is critical. In general, bound (1.18) overestimates $K(\mathbb{B})$. In Section 1.6.1 we will present a simple method for correctly estimating $K(\mathbb{B})$.

We recall that V_k represents the subset of language V^* composed of strings with maximum length k, including the empty string ε. Define $\text{Suff}_k : \mathscr{P} \to V_k$ such that, if

$|\pi| \leq k$, $\text{Suff}_k(\pi) = \pi$ and if $|\pi| > k$, $\text{Suff}_k(\pi)$ is the suffix of π of length k. Function Suff_k allows to introduce a partially defined transition function $\Gamma : V_k \times V \to V_k$ by setting $\Gamma(r,\sigma) = \text{Suff}_k(r\sigma)$ if $r\sigma \in \mathscr{P}$, otherwise, if $r\sigma \notin \mathscr{P}$, $\Gamma(r,\sigma)$ is not defined. Define the incremental cost function $\eta : \mathscr{P}_s \times V \to \mathbb{R}^+$ such that, for $p \in \mathscr{P}_s$ and $\sigma \in V$, if $p\sigma \in \mathscr{P}_s$, $\eta(p,\sigma) = T_{\mathbb{B}}(p\sigma) - T_{\mathbb{B}}(\pi)$, otherwise $\eta(\pi,\sigma) = +\infty$. In other words, $\eta(\pi,\sigma)$ is the difference between the minimum-time required for traversing $\pi\sigma$ and the minimum-time required for traversing π. For simplicity of notation, from now on we will denote $T_{\mathbb{B}}$ simply as T. The following proposition shows that the incremental cost is always strictly positive.

Proposition 1.6.1. $\eta(\pi,\sigma) \geq T(\sigma)$.

Proof. By 1) of Proposition 1.3.1, $T(\pi\sigma) \geq T(\pi) + T(\sigma)$. □

The following property, whose proof is presented in the Appendix **??**, plays a key role in the solution algorithm.

Proposition 1.6.2. *Let $\pi_1, \pi_2, t \in \mathscr{P}$, if $\pi_1 t, \pi_2 t \in \mathscr{P}$ and $\ell^+(t) \leq \ell^-(t)$, then* $(\forall \sigma \in V)\ T(\pi_1 t\sigma) - T(\pi_1 t) = T(\pi_2 t\sigma) - T(\pi_2 t)$.

The following is a direct consequence of Proposition 1.6.2. It states that, given $\pi \in \mathscr{P}$ and $\sigma \in V$, the incremental cost $\eta(\pi,\sigma)$ does not depend on the complete path π, but only on $\text{Suff}_k(\pi)$ (its last k symbols).

Proposition 1.6.3. *If $K(\mathbb{B}) \leq k$ and $\pi, \pi' \in \mathscr{P}$ are such that* $\text{Suff}_k(\pi) = \text{Suff}_k(\pi')$, *then* $(\forall \sigma \in V)\ \eta(\pi,\sigma) = \eta(\pi',\sigma)$.

Define function $\hat{\eta} : V_k \times V \to \mathbb{R}^+$, such that $\hat{\eta}(r,\sigma) = \eta(\pi,\sigma)$ where $\pi \in \mathscr{P}$ is any path such that $r = \text{Suff}_k(\pi)$. We set $\hat{\eta}(r,\sigma) = +\infty$ if such path does not exist. Note that function $\hat{\eta}$ is well-defined by Proposition 1.6.3, being $\eta(\pi,\sigma)$ identical among all paths π such that $r = \text{Suff}_k(\pi)$. In particular, Proposition 1.6.3 holds for $\pi' = \text{Suff}_k(\pi) = r$, so that we can compute $\hat{\eta}$ as $\hat{\eta}(r,\sigma) = \eta(r,\sigma)$. In the following, since $\hat{\eta}$ is the restriction of η on $V_k \times V$, we denote $\hat{\eta}$ simply by η.

The value k can be viewed as the amount of memory required to solve the problem: once a node is reached, the optimal path from such node to the target one depends on

the last k visited nodes. If $k = 1$, it only depends on the current node (i.e., no memory is required). This is the situation with the classical SPP. More generally, $k > 1$, so that the optimal way to complete the path does not only depend on the current node, but also on the sequence of $k-1$ nodes visited before reaching it. Define function $V : V_k \to \mathbb{R}$ as

$$V(r) = \min_{\pi \in \mathscr{P}_s | \mathrm{Suff}_k \pi = r} T_{\mathbb{B}}(\pi). \tag{1.19}$$

Note that the solution of BASP corresponds to $\min_{r \in V_k | \vec{r} \in F} V(r)$ (we recall that $\vec{r}$ is the last node of r). For $r \in V_k$, define the set of predecessors of r as $\mathrm{Prec}(r) = \{\bar{r} \in V_k \mid r = \Gamma(\bar{r}, \vec{r})\}$. The following proposition presents an expression for $V(r)$ that holds if $\ell^+(r') \leq \ell^-(r')$ for all predecessors r' of r.

Proposition 1.6.4. *Let $r \in V_k$, if $(\forall r' \in \mathrm{Prec}(r))\ \ell^+(r') \leq \ell^-(r')$, then*

$$V(r) = \min_{r' \in \mathrm{Prec}(r)} \{V(r') + \eta(r', \vec{r})\}. \tag{1.20}$$

Proof. Let $S_r = \{q \in \mathscr{P}_s \mid \mathrm{Suff}_k q\vec{r} = r\}$.
$V(r) = \min_{p \in \mathscr{P}_s | \mathrm{Suff}_k p = r} T(p) = \min_{q \in S_r} \{T(q\vec{r}) - T(q) + T(q)\} = \min_{q \in S_r} \{T(q) + T((\mathrm{Suff}_k q)\vec{r}) - T(\mathrm{Suff}_k q)\} = \min_{q \in S_r} \{T(q) + \eta(\mathrm{Suff}_k q, \vec{r})\} = \min_{r' \in \mathrm{Prec}(r),\ q \in S_{r'}} \{T(q) + \eta(r', \vec{r})\} = \min_{r' \in \mathrm{Prec}(r)} \{V(r') + \eta(r', \vec{r})\}$, where we used the facts that $T(q\sigma) - T(q) = T(\mathrm{Suff}_k q\sigma) - T(\mathrm{Suff}_k q)$, by Proposition 1.6.2, and that $q \in \mathscr{P}_s$ is such that $\mathrm{Suff}_k q\vec{r} = r \Leftrightarrow \mathrm{Suff}_k q \in \mathrm{Prec}(r)$. □

As a consequence of Proposition 1.6.4, if $(\forall r \in V_k)\ \ell^+(r) \leq \ell^-(r)$, $V(r)$ corresponds to the length of the shortest path from s to r on the extended directed graph $\tilde{G} = (\tilde{V}, \tilde{E})$, where $\tilde{V} = V_k$ and $(r_1, r_2) \in \tilde{E}$ if $r_2 = \Gamma(r_1, \vec{r}_2)$ is defined, in this case its length is $\eta(r_1, \vec{r}_2)$. The left part of Figure 1.11 shows a graph consisting of 3 nodes. Node $s = 1$ is the source (indicated by the entering arrow) and the double border shows the final node $F = \{3\}$. The right part of Figure 1.11 represents the corresponding extended graph, obtained for $k = 2$, consisting of 13 nodes (the cardinality of V_2). Note that some of the nodes are unreachable from the initial state, these are represented with dotted borders.

Solving k-BASP corresponds to finding a minimum-length path on $\tilde{G}$ that connects node $s \in V_k$ to $\tilde{F} = \{r \in V_k \mid \vec{f} \in F\}$. Note that the set of final states $\tilde{F}$ for the extended

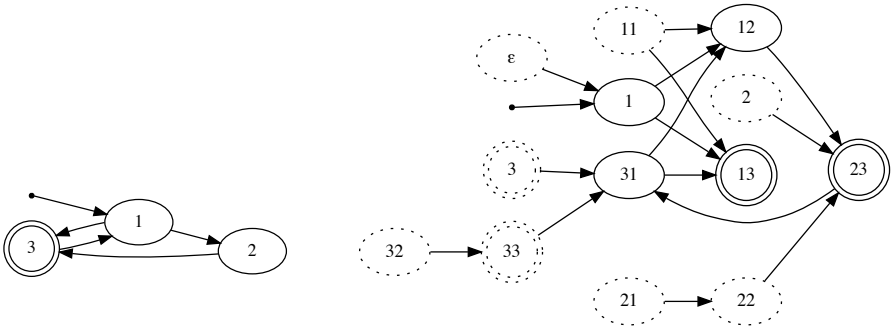


Figure 1.11: A graph and its corresponding extension for $k = 2$.

graph $\tilde{G}$ contains all paths $\pi \in V_k$ that end in an element of F. In the extended graph reported in Figure 1.11, this corresponds to finding a minimum-length path from starting node 1 to one of the final nodes 3, 13, 23, 33. Note that the unreachable nodes play no role in this procedure. We can find a minimum-length path by Dijkstra's algorithm applied to $\tilde{G}$, leading to the following complexity result.

Proposition 1.6.5. *k-BASP can be solved with complexity $O(|V|^{k-1}|E| + (|V|^k \log |V|^k))$.*

Proof. Dijkstra's algorithm has time complexity $O(|E| + |V| \log |V|)$, where $|E|$ and $|V|$ are the cardinalities of the edge and vertex sets, respectively. In our case, $|V| = |\tilde{V}| = |V_k| = \sum_{i=0}^{k} |V|^i = O(|V|^k)$, $|E| = |\tilde{E}| \leq |V_{k-1}E| = O(|V|^{k-1}|E|)$. □

The following remark establishes that SPP can be viewed as a special case of BASP without acceleration bounds.

Remark 1.6.1. *If $(\forall \theta \in E)\ \alpha^-(\theta) = -\infty$, $\alpha^+(\theta) = +\infty$, then $K(\mathbb{B}) = 1$. The resulting 1-BASP reduces to a standard SP on graph G and can be solved with time complexity $O(|E| + |V| \log |V|)$.*

1.6.1 Adaptive A* algorithm for k-BASP

The computation method based on Dijkstra's algorithm on the extended graph $\tilde{G}$, presented in the previous section, has two main disadvantages. First, $\tilde{G}$ has $\sum_{j=0}^{k} |V|^j$ nodes, so that the time required by Dijkstra's algorithm grows exponentially with k.

We will show that it is possible to mitigate this problem and reduce the number of visited nodes by using A* algorithm with a suitable heuristic. Second, the estimation of $k = K(\mathbb{B})$ from its definition is not an easy task. We will show that it is quite easy to adaptively find the correct value of k by starting from $k = 2$ and increasing k if needed.

A* algorithm is a heuristic method that allows to compute the optimal path, if it exists (see [42]), by exploring the graph beginning from the starting node along the most promising directions according to a heuristic function that estimates the cost from the current position to the target node. Hence, to implement the A* algorithm, we need to define a heuristic function $h : V_k \to \mathbb{R}$, such that, for $r \in V_k$, $h(r)$ is a lower bound on $\min_{\pi \in \mathscr{P}_{\vec{r},\tilde{F}}} T(\pi)$, that is, the minimum time needed for traveling from $\vec{r}$ to a final state in $\tilde{F}$. In general, we can compute lower bounds for BASP by relaxing the acceleration constraints α^-, α^+. Namely, let $\hat{\mathbb{B}}$ be a parameter set obtained by relaxing acceleration constraints in $\mathbb{B}$. Then, if $K(\hat{\mathbb{B}}) < K(\mathbb{B})$, by Proposition 1.6.5, the solution of BASP for parameters $\hat{\mathbb{B}}$ can be computed with a lower computational time than the solution with parameters $\mathbb{B}$. In particular, we obtain a very simple lower bound by removing acceleration bounds altogether, that is, by setting $\alpha^- = -\infty$ and $\alpha^+ = +\infty$. In this way, the vehicle is allowed to travel at maximum speed everywhere along the path and the incremental cost function $\eta(\pi, \sigma)$ is given by the time needed to travel γ_σ at maximum speed, that is: $\eta(\pi,\sigma) = \int_0^{\ell(\vec{\pi}\sigma)} \frac{1}{\sqrt{\mu^+((\vec{\pi},\sigma),\lambda)}} d\lambda$. Define the heuristic $h : V_k \to \mathbb{R}^+$ as

$$h(r) = \min_{\pi \in \mathscr{P}_{\vec{r},\tilde{F}}} T_{\hat{\mathbb{B}}}(\pi). \tag{1.21}$$

Note that, if $\alpha^- = -\infty$ and $\alpha^+ = +\infty$, h corresponds to the solution of 1-BASP and all values of h can be efficiently pre-computed by Dijkstra's algorithm (see Remark 1.6.1). The following proposition shows that h is admissible and consistent, so that A* algorithm, with heuristic h, provides the optimal solution of k-BASP and its time-complexity is no worse than Dijkstra's algorithm (see Theorems 2.9, 2.10 of [43]).

Proposition 1.6.6. *Heuristic h satisfies these two properties:*
1) *admissibility:* $(\forall r \in V_k)\ h(r) \le \min_{q \in P_{\vec{r},f}} T_{\mathbb{B}}(q)$.
2) *consistency:* $(\forall r \in V_k)\,(\forall \sigma \in V)\, h(r) \le \eta(r,\sigma) + h(\Gamma(r,\sigma))$.

Proof. 1) $h(r) = \min_{p \in P_{\vec{r},f}} T_{\hat{\mathbb{B}}}(p) \leq \min_{q \in P_{\vec{r},f}} T_{\mathbb{B}}(q)$, since $\hat{\mathbb{B}}$ is a relaxation of $\mathbb{B}$.
2) $h(r) = \min_{p \in P_{\vec{r},f}} T_{\hat{\mathbb{B}}}(p) \leq T_{\hat{\mathbb{B}}}(\sigma) + \min_{p \in P_{\sigma,f}} T_{\hat{\mathbb{B}}}(p) \leq T_{\mathbb{B}}(\sigma) + \min_{p \in P_{\sigma,f}} T_{\hat{\mathbb{B}}}(p) \leq \eta(r,\sigma) + \min_{p \in P_{\sigma,f}} T_{\hat{\mathbb{B}}}(p) = \eta(r,\sigma) + h(\Gamma(r,\sigma))$, where $T_{\hat{\mathbb{B}}}(\sigma) \leq T_{\mathbb{B}}(\sigma)$ by 2) of Proposition 1.3.1 and $T_{\mathbb{B}}(\sigma) \leq \eta(r,\sigma)$ by Proposition 1.6.1. □

Since heuristic h is admissible and consistent, A* is equivalent to Dijkstra's algorithm, with the only difference that the incremental cost function $\eta(r,\sigma)$ is replaced by modified cost

$$\tilde{\eta}(r,\sigma) = \eta(r,\sigma) + h(\Gamma(r,\sigma)) - h(r) \tag{1.22}$$

(see Lemma 2.3 of [43] for a complete discussion). A description of A* algorithm can be found in literature (for instance, see Algorithm 2.13 of [43]). We define a priority queue $\mathscr{Q}$ that contains open nodes, that is, nodes that have already been generated but have not yet been visited. Namely, $\mathscr{Q}$ is an ordered set of pairs $(r,t) \in V_k \times \mathbb{R}^+$, in which $r \in V_k$ and t is a lower bound for the time associated to the best completion of r to a path arriving at a final state. We need to perform the following operations on $\mathscr{Q}$: operation $\texttt{Insert}(\mathscr{Q},(r,t))$ inserts couple (r,t) into $\mathscr{Q}$; operation $(r,t) = \texttt{DeleteMin}(\mathscr{Q})$ returns the first couple of $\mathscr{Q}$, that is, the couple (r,t) with the minimum time t, and removes this couple from $\mathscr{Q}$; and, operation $\texttt{DecreaseKey}(\mathscr{Q},(r,t))$ assumes that $\mathscr{Q}$ already contains a couple (r,t') with $t' > t$ and substitutes this couple with (r,t). Further, we consider three partially defined maps $\texttt{value} : V_k \to \mathbb{R}$, $\texttt{parent} : V_k \to V_k$, $\texttt{closed} : V_k \to \{0,1\}$, such that, for $r \in V_k$, $\texttt{value}(r)$ is the current best upper estimate of $V(r)$, $\texttt{parent}(r)$ is the parent node of r and $\texttt{closed}(r) = 1$ if node r has already been visited. Maps `value`, `parent`, and `closed` can be implemented as hash tables.

Algorithm 1.6.1 (A* algorithm for k-BASP)**.**
1) *[initialization] Set* $\mathscr{Q} = \{(s,h(s))\}$, $\texttt{value}(s) = 0$.
2) *[expansion] Set* $(r,t) = \texttt{DeleteMin}(\mathscr{Q})$ *and set* $\texttt{closed}(r) = 1$. *If* $\vec{r} \in \tilde{F}$, *then* t *is the optimal solution and the algorithm terminates, returning maps* `value`, `parent`. *Otherwise, for each* $\sigma \in V$ *for which* $\Gamma(r,\sigma)$ *is defined, set* $r' = \Gamma(r,\sigma)$, $t' = t + \tilde{\eta}(r,\sigma)$.

If $\texttt{closed}(r') = 1$, *go to* 3). *Else, if* $\texttt{value}(r')$ *is undefined* $\texttt{Insert}(\mathscr{Q}, (r', t'))$. *Otherwise, if* $t' < \texttt{value}(r')$, *set* $\texttt{value}(r') = t'$, $\texttt{parent}(r') = r$ *and do* $\texttt{DecreaseKey}(\mathscr{Q}, (r', t'))$. 3) *[loop] If* $\mathscr{Q} \neq \varnothing$ *go back to* 2), *otherwise no solution exists.*

Proposition 1.6.7. *Algorithm 1.6.1 terminates and returns the optimal solution (if it exists), with a time-complexity not higher than Dijkstra's algorithm on the extended graph* $\tilde{G}$.

Proof. It is a consequence of the fact that heuristic h is admissible and consistent (see Theorems 2.9, 2.10 of [43]). □

Note that, at the end of Algorithm 1.6.1, $\texttt{value}(f)$ is the optimal value of k-BASP and the optimal path from s to set F can be reconstructed from map $\texttt{parent}$.

One possible limitation of Algorithm 1.6.1 is that estimating $K(\mathbb{B})$ from its definition can be difficult. A correct estimation of $K(\mathbb{B})$ is critical for the efficiency of the algorithm. Indeed, if $K(\mathbb{B})$ is overestimated, the time-complexity of the algorithm is higher than it would be with a correct estimate. On the other hand, if $K(\mathbb{B})$ is underestimated, Algorithm 1.6.1 is not correct since Proposition 1.6.4 does not hold. Here we propose an algorithm that adaptively finds a suitable value for k in Algorithm 1.6.1, such that $k \leq K(\mathbb{B})$, but, in any case, allows to find the optimal solution of BASP. First, we define the modified cost function $W : V_k \to \mathbb{R}$ as $W(r) = V(r) + h(r)$, where V is given by (1.19) and h is the heuristic given by (1.21). If $(\forall r \in V_k)\ \ell^+(r) \leq \ell^-(r)$, then W is the solution of

$$\begin{cases} W(r) = \min_{r' \in \mathrm{Prec}(r)} \{W(r') + \tilde{\eta}(r', \vec{r})\} \\ W(s) = h(s). \end{cases} \tag{1.23}$$

Indeed, following the same steps of the proof of Proposition 1.6.4, $W(r) = V(r) + h(r) = \min_{r' \in \mathrm{Prec}(r)} \{V(r') + \eta(r', \vec{r}) + h(r) + h(r') - h(r')\} = \min_{r' \in \mathrm{Prec}(r)} \{W(r') + \tilde{\eta}(r', \vec{r})\}$. Hence, $W(r)$ corresponds to the length of the shortest path from s to r on $\tilde{G}$, with arc-length given according to $\tilde{\eta}$. If condition $\ell^+(r) \leq \ell^-(r)$ is not satisfied for all $r \in V_k$, equation (1.23) does not hold for all $r \in V_k$ and W does not represent the solution of an SP. However, the following proposition shows that we can still find a lower bound $\hat{W}$ of W that does correspond to the solution of an SP.

Proposition 1.6.8. *Let $\hat{W} : V_k \to \mathbb{R}$ be the solution of*

$$\begin{cases} \hat{W}(r) = \min_{r' \in \mathrm{Prec}(r)} \{\hat{W}(r') + \hat{\eta}(r', \vec{r})\} \\ \hat{W}(s) = 0, \end{cases} \tag{1.24}$$

where, if $\ell^+(r') \leq \ell^-(r')$ or $|r'| < k$, $\hat{\eta}(r', \vec{r}) = \tilde{\eta}(r', \vec{r})$, otherwise $\hat{\eta}(r', \vec{r}) = h(r) - h(r')$. Then, $(\forall r \in V_k)$
1) $\hat{W}(r) \leq W(r)$.
2) $(\forall \bar{r} \in V_k \mid \hat{W}(\bar{r}) \leq \hat{W}(r))\ \ell^+(\bar{r}) \leq \ell^-(\bar{r}) \Rightarrow \hat{W}(r) = W(r)$.

Proof. 1) For $r \in V_k$, let $p \in \mathscr{P}_s$ be such that $\mathrm{Suff}_k p \in \mathrm{Prec}(r)$. If $\ell^+(\mathrm{Suff}_k p) \leq \ell^-(\mathrm{Suff}_k p)$, in view of Proposition 1.6.2, $T(p\vec{r}) = T(p) + \eta(\mathrm{Suff}_k p, \vec{r})$, otherwise, obviously, $T(p\vec{r}) \geq T(p)$. Hence, in both cases, by the definition of $\tilde{\eta}$ in (1.22), $T(p\vec{r}) + h(r) \geq T(p) + h(\mathrm{Suff}_k p) + \hat{\eta}(\mathrm{Suff}_k p, \vec{r})$. By contradiction, assume $(\exists A \subset V_k)\ A \neq \varnothing$ such that $(\forall r \in A)\ \hat{W}(r) > W(r)$. Let $\bar{r} = \mathrm{argmin}_{\hat{r} \in A} W(\hat{r})$ and $S_{\bar{r}} = \{q \in \mathscr{P}_s \mid \mathrm{Suff}_k q \in \mathrm{Prec}(\bar{r})\}$, then, $W(\bar{r}) = V(\bar{r}) + h(\bar{r}) = \min_{p \in \mathscr{P}_s | \mathrm{Suff}_k p = \bar{r}} T(p) + h(\bar{r}) = \min_{q \in S_{\bar{r}}} T(q\vec{r}) + h(\bar{r}) \geq \min_{q \in S_{\bar{r}}} \{T(q) + h(\mathrm{Suff}_k(q)) + \hat{\eta}(\mathrm{Suff}_k q, \vec{\bar{r}})\} = \min_{r' \in \mathrm{Prec}(\bar{r})} \{\hat{W}(r') + \hat{\eta}(r', \vec{\bar{r}})\} = \hat{W}(\bar{r})$, where we used the fact that $W(r') = \hat{W}(r')$, that follows from the definition of $\bar{r}$, since the value of r' that attains the minimum is such that $W(r') < W(\bar{r})$. Then, the obtained inequality contradicts the fact that $\hat{W}(\bar{r}) > W(\bar{r})$.

2) Let $A \subset V$ be the set of values of $r \in V$ for which ii) does not hold and, by contradiction, assume that $A \neq \varnothing$ and let $\hat{r} = \mathrm{argmin}_{r \in A} \hat{W}(r)$. Then, by definition of $\hat{r}$, it satisfies the following two properties: $(\forall \bar{r} \in V_k \mid \hat{W}(\bar{r}) \leq \hat{W}(\hat{r}))\ \ell^+(\bar{r}) \leq \ell^-(\bar{r})$, moreover $\hat{W}(\hat{r}) \neq W(\hat{r})$. Note that, from the definitions of $\hat{W}$, $W(s) = \hat{W}(s)$. Then, $W(\hat{r}) = \min_{p \in \mathscr{P}_s | \mathrm{Suff}_k p = \hat{r}} T(p) + h(\hat{r}) = \min_{q \in \mathscr{P}_s | \mathrm{Suff}_k q \in \mathrm{Prec}(\hat{r})} \{T(q\vec{\hat{r}}) + h(\mathrm{Suff}_k q) - h(\mathrm{Suff}_k q) + h(\hat{r})\} = \min_{r' \in \mathrm{Prec}(\hat{r})} \{\hat{W}(r') + \hat{\eta}(r', \vec{\hat{r}})\} = \hat{W}(\hat{r})$, which contradicts the definition of $\hat{r}$. Here, we used (1.22) and the fact that, since $\hat{W}(r') < \hat{W}(\hat{r})$ and by the definition of $\hat{r}$, $\hat{W}(r') = W(r')$. □

Proposition 1.6.8 implies that $\hat{W}(r)$ is a lower bound of $W(r)$ and that it corresponds to the length of the shortest path from s to r on the extended directed graph $\tilde{G}$, with arc-length given in accordance to (1.24), namely by the value of function $\hat{\eta}$. Hence, $\hat{W}(f)$ can be computed by Dijkstra's algorithm (which is equivalent to

compute V with A* algorithm, with heuristic h). The algorithm that we are going to present is based on the following basic observation. If A* algorithm computes $f^* = \text{argmin}_{f \in \tilde{F}} \hat{W}(f)$ by visiting only nodes for which $\ell^+(r) \leq \ell^-(r)$, then 2) of Proposition 1.6.8 is satisfied for $r = f^*$ and $\hat{W}(f^*) = W(f^*)$ is the optimal value of k-BASP. If this is not the case, we increase k by 1 and re-run the A* algorithm. Note that the algorithm starts with $k = 2$, since, according to its definition, $K(\mathbb{B})$ equals 1 only if no acceleration bounds are present and, in this case, BASP is equivalent to a standard SP and can be solved by Dijkstra's algorithm.

Algorithm 1.6.2 (Adaptive A* algorithm for k-BASP)**.**
1) *Set* $k = 2$.
2) *Execute A* algorithm and, at every visit of a new node r, if none of the two conditions* $\ell^+(r) \leq \ell^-(r)$ *and* $|r| < k$ *holds, set* $k = k + 1$ *and repeat step* 2).

Note that the algorithm does not compute the exact value $K(\mathbb{B})$. Rather, it underestimates it. More precisely, it stops with the smallest k value needed to solve BASP between the given source and destination nodes. That is, the smallest k that satisfies the k-BASP definition over the explored subgraph.

Proposition 1.6.9. *Algorithm 1.6.2 terminates with* $k \leq K(\mathbb{B})$ *and returns an optimal solution.*

Proof. By Definition (1.17) of $K(\mathbb{B})$, if $k = K(\mathbb{B})$ condition $\ell^+(r) \leq \ell^-(r)$ is satisfied for all r. Hence, there exists $k \leq K(\mathbb{B})$ for which the algorithm terminates. Let $r \in V_k$, with $\vec{r} \in F$ be the last-visited node before the termination of the algorithm. By 2) of Proposition 1.6.8, we have that $\hat{W}(r) = W(r) = V(r)$ (since $h(r) = 0$), but, by definition, $V(r)$ is the shortest time for reaching a node in F. □

1.7 Computational experiments

In this section we test the approximation algorithm introduced in Section 1.5 and the adaptive A* algorithm for k-BASP presented in Section 1.6.1. We consider two real-life industrial scenarios of warehouses. The problem data have been provided by packaging

company Ocme S.r.l., based in Parma, Italy. The simulations have been performed on a 2.7 GHz Intel Core i5 dual-core with 8GB of RAM. The acceleration and deceleration bounds on both scenarios are given by $\alpha^+ = 0.28$ ms^{-2} and $\alpha^- = -0.18$ ms^{-2}, for arcs with mean curvature smaller than or equal to 0.25. Whilst on arcs whose mean curvature is larger than this value, the bounds are given by $\alpha^+ = 0.14$ ms^{-2} and $\alpha^- = -0.09$ ms^{-2}. The first scenario is modeled as a graph $G = (V,A)$ with 368 nodes and 679 edges and is depicted in Figure 3.14. The speed bounds on both

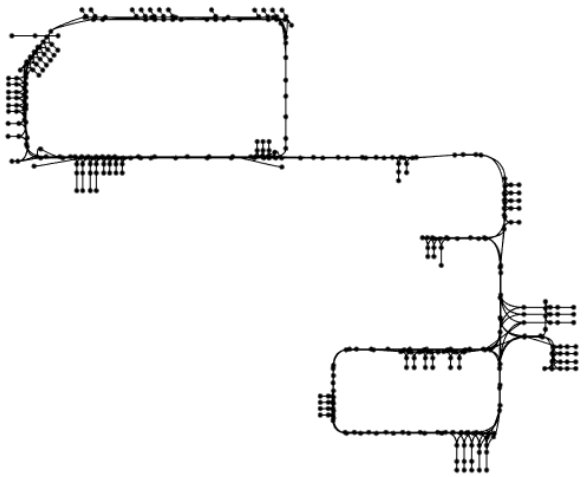


Figure 1.12: First real-life industrial scenario.

scenarios are constant for each arc but vary from arc to arc, according to the associated paths curvatures. For the first scenario they take values in interval $[0.136, 1.7]$ ms^{-1}, whilst the arc-lengths take values between 0.628 m and 10.87 m and have an average value of 2.863 m.

In what follows, we compare the approximation algorithm of Section 1.5 with the adaptive A* algorithm for k-BASP presented in Section 1.6.1. Note that the adaptive algorithm solves a k-BASP instance in polynomial time complexity with respect to the

number of edges and vertices of the associated graph, but its complexity is exponential with respect to k. In the following, the adaptive A* algorithm for k-BASP is initialized with $k = 3$.

Given the graphs associated to the considered automated warehouses, we generated the extended graphs associated to them for different values of the discretization step h of the squared speeds. For the first scenario, step h takes values in a set H of thirty logarithmically spaced values between 0.005 and 0.5 $\mathrm{m^2\,s^{-2}}$, hence, we considered thirty different sets of discretized squared speeds Ω_h and extended graphs $G'_h = (V'_h, E'_h)$. We generated 1000 random pairs of source-target nodes $\{(s_i, t_i)\}_{i\in\{1,\dots,1000\}}$ in $V \times V$. Then, for each of the previous pairs of source-target nodes, we considered the corresponding pair $(s_i^h, t_i^h) = ((s_i, 0), (t_i, 0)) \in V'_h \times V'_h$ on extended graph G'_h and ran Dijkstra's algorithm on G'_h in order to obtain a trajectory starting at node s_i with null speed and ending at node t_i with null speed, for $h \in H$. Figure 1.13 shows the box-and-whisker plot of the computational times of Dijkstra's algorithm for different values of the discretization step h for solving the 1000 randomly generated instances and compares them with those of the adaptive A* algorithm for k-BASP on the same set of instances. Note that, as the discretization step h increases, the number of dis-

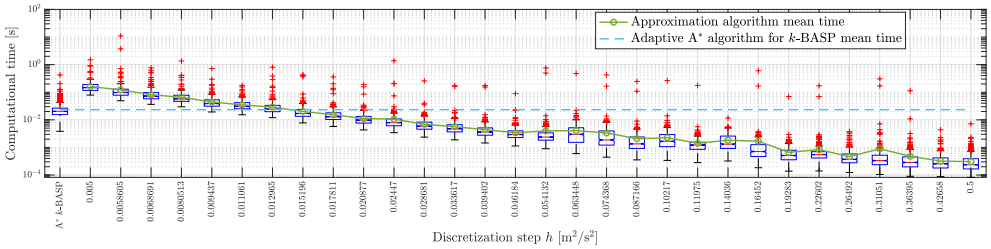


Figure 1.13: Approximation algorithm computational times for different values of h on the first scenario.

cretized squared speeds decreases, hence, the number of nodes and edges in graph G'_h decreases as well making Dijkstra's algorithm explore a smaller graph and run faster. Also, observe that, for values of h greater than 0.015159 $\mathrm{m^2\,s^{-2}}$, the mean computational times of Dijkstra's algorithm on extended graphs G'_h (represented by a green line with circles) are better than that of the adaptive A* algorithm for k-BASP

(represented by a dashed line). On the other hand, as the discretization step h increases, so does the relative error on the travel time, which, for values of $h \geq 0.19283$ $\mathrm{m^2\,s^{-2}}$, is larger than 10^{-1}. Figure 1.14 represents the box-and-whisker plot of the relative error. Note that we set a tolerance on the relative error of the trajectories obtained with the approximation algorithm, relative errors smaller than 10^{-4} are not considered. This roughly corresponds to an absolute error of the order of 10^{-2} s. A good compro-

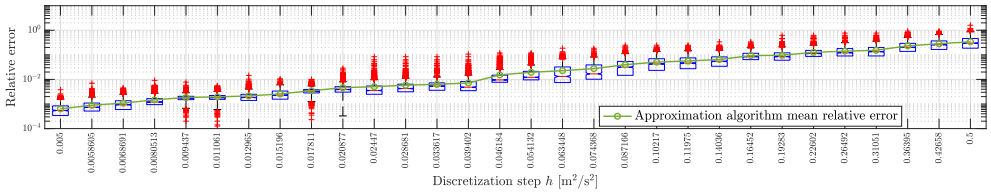


Figure 1.14: Approximation algorithm relative error for different values of h on the first scenario.

mise for this scenario could be $h = 0.063448$ $\mathrm{m^2\,s^{-2}}$ which is associated to a mean computational time that is 5.76 times faster than that of the adaptive A* algorithm for k-BASP, while at the same time maintaining a mean relative error of the order of $2 \cdot 10^{-2}$. We could also exploit the approximation algorithm just for obtaining a path and then compute the optimal speed profile along such a path by the procedure described in Section 1.3. In this way we could employ a bigger discretization step h for achieving small computational times while maintaining high precision. The speed planning algorithm described in Section 1.3 has linear-time computational complexity with respect to the number of nodes of the path. Figure 1.15 shows the box-and-whisker plot of the computational times of the approximation algorithm, as in Figure 1.13, summed with those of the speed planning algorithm applied on the obtained paths for the 1000 randomly generated instances on the first scenario. Figure 1.16 shows the box-and-whisker plot of the relative error on the travel time of the trajectories obtained coupling the approximation algorithm of Section 1.5 with the speed planning one. Again, we set a tolerance on relative errors of 10^{-4}. In this case the mean relative errors are on average two orders of magnitude smaller than those presented in Figure 1.14 and the percentage of solutions with a relative error

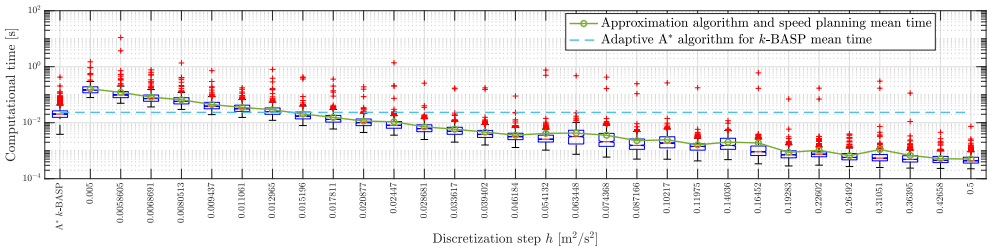


Figure 1.15: Approximation algorithm and speed planning algorithm computational times for different values of h on the first scenario.

smaller than 10^{-4} ranges from 93.1% with $h = 0.5$ m^2 s^{-2} to 100% for $h = 0.005$ m^2 s^{-2}. For this scenario, if we fix $h = 0.5$ m^2 s^{-2}, we get a mean computational time

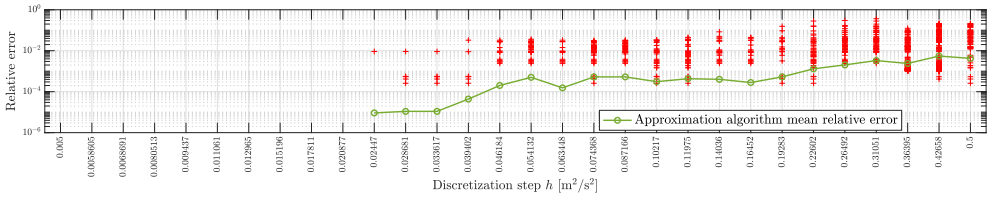


Figure 1.16: Approximation algorithm and speed planning algorithm relative error for different values of h on the first scenario.

that is 46.35 times faster than that of the adaptive A* algorithm for k-BASP, while obtaining a solution with a mean relative error of $4 \cdot 10^{-3}$, which is one order of magnitude smaller than that of the approximation algorithm alone with $h = 0.063448$ m^2 s^{-2}.

It is worthwhile to remark that there always exists a sufficiently small value h such that the optimal path of the discretized problem coincides with the optimal path of the continuous problem. Indeed, if h is chosen in such a way that the absolute error of the approximation algorithm, bounded from above as discussed in Proposition 1.5.2, is lower than the difference between the traveling times of the best and second-best path, then the approximation algorithm returns the best path (that is, according to the notation introduced in Section 1.5, π^h is equal to π^*). However, the difference is not known in advance and even in case it were known, the choice of h based on the upper bound stated in Proposition 1.5.2 may lead to an excessively small value.

The second scenario is modeled as a graph with 2419 nodes and 4255 edges and is depicted in Figure 1.17. For this scenario the speed bounds take values in interval

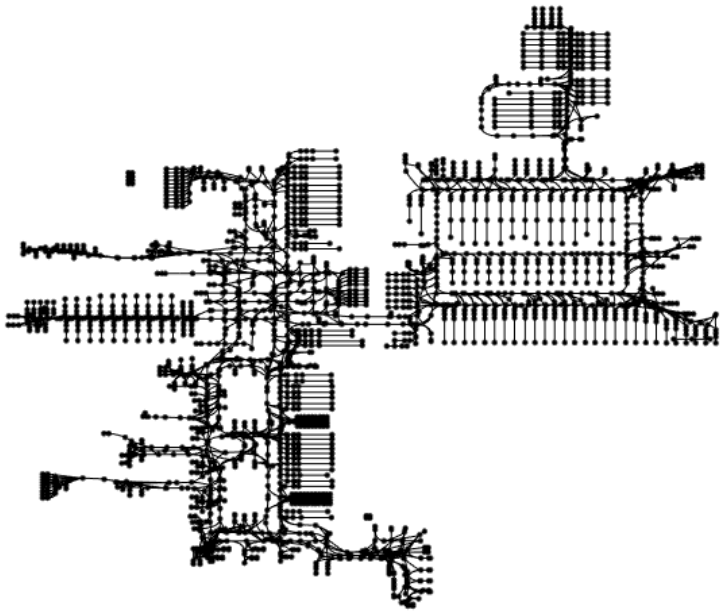


Figure 1.17: Second real-life industrial scenario.

$[0.1, 1.7]$ m s^{-1}, whilst the arc-lengths take values between 0.2 m and 18.15 m and have an average value of 4.24 m. For the first scenario, step h takes values in a set H of ten logarithmically spaced values between 0.03 and 1 m^2 s^{-2}, hence, we considered ten different sets of discretized squared speeds Ω_h and extended graphs $G'_h = (V'_h, E'_h)$. As for the previous scenario, we generated 1000 random pairs of source-target nodes $\{(s_i, t_i)\}_{i\in\{1,\dots,1000\}}$ in $V \times V$ and tested the approximation algorithm on such pairs. Figure 1.18 shows the box-and-whisker plot of the computational times of Dijkstra's algorithm for different values of the discretization step h for solving the 1000 randomly generated instances and compares them with those of the adaptive A* algorithm for

k-BASP on the same set of instances. Observe that, for all values of h, the mean

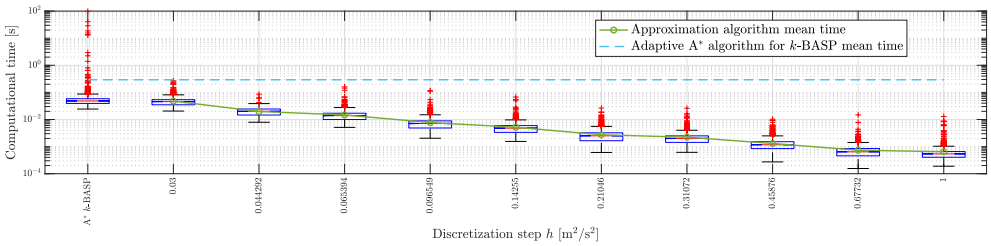


Figure 1.18: Approximation algorithm computational times for different values of h on the second scenario.

computational times of Dijkstra's algorithm on extended graphs G'_h (represented by a green line with circles) are better than that of the adaptive A* algorithm for k-BASP (represented by a dashed line). However, note that the median computational time of the latter is almost the same as that of the approximation algorithm with $h = 0.03$ $\mathrm{m}^2\,\mathrm{s}^{-2}$ (both represented as horizontal red lines within their corresponding blue boxes). This is due to the fact that the adaptive A* algorithm for k-BASP presents a small group of outliers with very high computational times compared to its median. On the other hand, as the discretization step h increases, so does the relative error on the travel time, which, for values of $h \geq 0.45876$ $\mathrm{m}^2\,\mathrm{s}^{-2}$, is larger than 10^{-1}. Figure 1.19 represents the box-and-whisker plot of the relative error for which we set a tolerance of 10^{-4}. A good compromise for this scenario could be $h = 0.21046$ $\mathrm{m}^2\,\mathrm{s}^{-2}$ which

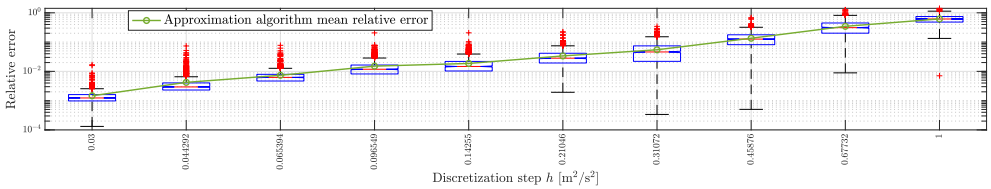


Figure 1.19: Approximation algorithm relative error for different values of h on the second scenario.

is associated to a mean computational time that is 107.3 times faster than that of the adaptive A* algorithm for k-BASP, while at the same time maintaining a mean relative error of the order of $3 \cdot 10^{-2}$. Figure 1.20 shows the box-and-whisker plot of

the computational times of the approximation algorithm, as in Figure 1.18, summed with those of the speed planning algorithm described in Section 1.3 applied on the obtained paths for the 1000 randomly generated instances on the first scenario. Fig-

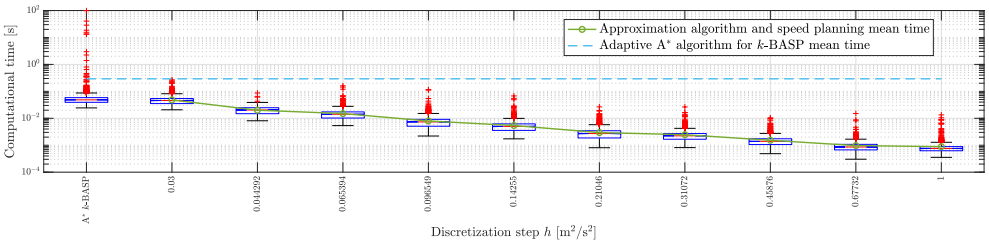


Figure 1.20: Approximation algorithm and speed planning algorithm computational times for different values of h on the second scenario.

ure 1.21 shows the box-and-whisker plot of the relative error on the travel time of the trajectories obtained coupling the approximation algorithm of Section 1.5 with the speed planning one. Again, we set a tolerance on relative errors of 10^{-4}. In this case the mean relative errors are on average two orders of magnitude smaller than those presented in Figure 1.19. For this scenario, if we fix $h = 0.21046\ \mathrm{m^2\,s^{-2}}$, we

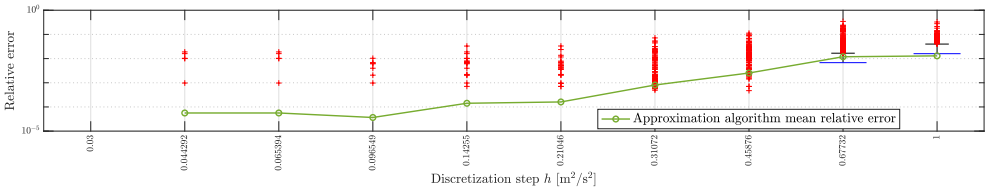


Figure 1.21: Approximation algorithm and speed planning algorithm absolute error for different values of h on the second scenario.

get a mean computational time that is 99.24 times faster than that of the adaptive A* algorithm for k-BASP, while obtaining the exact solution up to a tolerance of 10^{-4} on the relative error in 97.6% of the cases.

Observe that the construction of the extended graphs can be a time-consuming operation. One could alternatively run a dynamic programming approach by generating arcs only when (and if) needed. However, the construction has to be performed only once and the extended graphs can be stored for future use. The memory occupancy of

the extended graphs of the scenarios we considered varies from 36 KB for G'_h with $h = 0.5$ m^2 s^{-2} to 258 MB for G'_h with $h = 0.005$ m^2 s^{-2} for the first scenario, and from 78 KB for G'_h with $h = 1$ m^2 s^{-2} to 56.1 MB for G'_h with $h = 0.03$ m^2 s^{-2} for the second scenario.

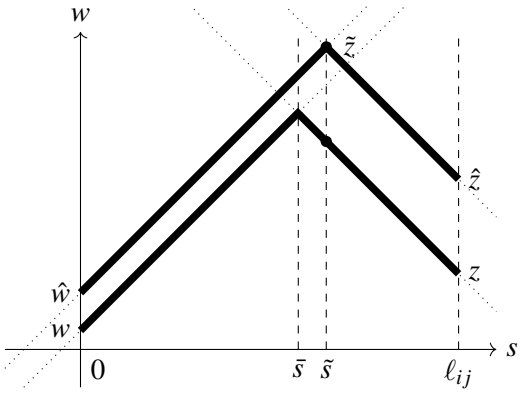


Figure 1.22: Comparison on a generic arc between the optimal speed profile starting from $\hat{w}$ and ending at $\hat{z}$ and the one starting from w and ending at z.

Data availability

The data and code that support the findings of this study are available at: OSF. May 23. `osf.io/6jupx` S. Ardizzoni, L. Consolini, M. Laurini, and M. Locatelli. 2022. "Bounded Acceleration Shortest Path Problem".

Chapter 2

Multi Agent Path Finding Problem

Multi-Agent Path-Finding (MAPF), also called pebble motion on graphs, or cooperative path-finding, is the problem of finding a collision-free movement plan for a set of agents (or pebbles) moving on a graph. This problem has been widely discussed, together with its many variants [1], on various types of graphs. For most graph classes, finding an optimal solution of MAPF (that is, a solution with a minimum number of moves) is NP-hard [44]. Instead, the complexity of checking MAPF feasibility depends on the specific graph class. For instance, it is polynomial on undirected graphs [8] and on strongly biconnected directed graphs [6]. Instead, it is NP-hard in the general case of directed graphs [45]. Optimal and suboptimal algorithms have been proposed in the last forty years [46, 47, 5, 48, 6, 7, 8, 49].

Various works address the problem of finding the optimal solution of MAPF (i.e., the solution with the minimum number of moves). For instance, Conflict Based Search (CBS) is a two-level algorithm which uses a search tree, based on conflicts between individual agents (see [49]).

Search-based suboptimal solvers aim to provide a high quality solution, but are not complete (i.e., they are not always able to return a solution). A prominent example is Hierarchical Cooperative A* (HCA*) [50], in which agents are planned one at a time according to some predefined order.

Instead, rule-based approaches include specific movement rules for different

scenarios. They favor completeness at low computational cost over solution quality. Since their aim is to detect feasible solutions, agents are supposed to move one at a time [5, 6, 7, 8]. Of course, once we have found a feasible solution with one of these algorithms, we can find a shorter solution by allowing simultaneous moves. For instance, we can search in a neighborhood of the feasible solution, as we show in Chapter 4.

Many of the rule-based algorithms deal with the special case of trees. In this case the problem is called *pebble motion on a tree* (PMT). In literature, there exist many complete sub-optimal algorithms for solving PMT. In particular, Kornhauser and coauthors [8] present a procedure which solves it in $O(n^3)$ moves. However, the approach is not described algorithmically, but must be derived from a number of proofs in the paper [51]. This requires significant effort, and, to the best of our knowledge, Kornhauser's results have never been fully implemented. Auletta and coauthors [5] present an algorithm for deciding the feasibility of PMT, from which a solution can be derived requiring $O(k^2(n-k))$ moves. However, the paper does not explicitly explain how to find such a solution. Korshid and coauthors [52] present an algorithm for PMT (called TASS) that is easy to understand and implement. However, solutions provided by TASS require $O(n^4)$ moves. Therefore, after Kornhauser and coauthors in 1984, no one has proposed a clear and detailed algorithm with length-complexity $O(n^3)$. The result of Kornhauser is a fundamental step in the study of PMT complexity. However, under a more practical point of view, the implementation of a simple and efficient algorithm for solving PMT with length-complexity at least $O(n^3)$ remains an open problem. The aim of this Chapter is to address this problem by proposing an efficient, clear, and simple PMT solver, with a more detailed complexity result with respect to Kornhauser's.

Although PMT is one of the simplest versions of MAPF, it is quite relevant. Indeed, various algorithms that solve MAPF on more general graphs are based on a reformulation of MAPF as a PMT *with trans-shipment vertices* (*ts*-PMT), a variant of the PMT problem. Trans-shipment [53, 54] is a particular type of vertex that cannot host pebbles. This variant can be solved as the classical PMT with some minor modifications and it is useful since any MAPF instance on an undirected graph can be

reduced to an instance of this problem on a tree. For example, reference [54] presents a method that converts the graph into a tree with trans-shipment vertices (as in [53]), and solves the resulting problem with TASS. Another important rule-based algorithm for MAPF on undirected graphs is *Push and Rotate* [7] [47], which solves every MAPF instance on graphs that contains at least two holes (i.e., unoccupied vertices). One of the best rule-based algorithms for MAPF in terms of length complexity is presented by Kornhauser. Indeed, in his P.h.D. thesis [8] he describes a procedure that finds solutions with $O(n^3)$ moves. However, this approach is very difficult to implement [51].

The literature cited so far concerns exclusively undirected graphs, where motion is permitted in both directions along graph edges. Fewer results are related to directed graphs. Nebel [45] proves that finding a feasible solution of MAPF on a general directed graph (digraph) is NP-hard. However, in some special cases this problem can be solved in polynomial time. One relevant reference is [6], which solves MAPF on the specific class of biconnected digraphs, i.e., strongly connected digraphs where the undirected graphs obtained by ignoring the edge orientations have no cutting vertices. The proposed algorithm has polynomial complexity with respect to the number of nodes.

In this Chapter, we also deal with two variants of MAPF: the *motion planning* problem and the *unlabeled* MAPF. The Motion Planning problem (MP in what follows), consists in finding a plan such that a single marked agent reaches a desired target vertex, avoiding collisions ([55, 56, 57, 58]). That is, in MP all non-marked agents are obstacles that need to be moved out of the way to re-position the marked one. Reference [55] proves that finding an optimal solution for an MP instance is NP-hard. Instead, [56] shows that the feasibility of MP on an acyclic digraph $G = (V, E)$ can be decided in polynomial time $O(|V||E|)$. Reference [57] generalizes this result, showing that, on general digraphs, feasibility can be decided in polynomial time $O(|V|^2|E|)$.

The *unlabeled* MAPF (also known as U-GRAPH-RP [59, 60] or *anonymous MAPF* [61]), pebbles are not labeled (i.e., there is a set of target positions D, and each pebble has to reach a vertex in D, not specified in advance).

This Chapter is structured as follows. In Section 2.1 we give the formal definition of all the problems addressed in the chapter: MAPF, MP, PMT, unlabeled PMT. For each of these definitions we consider the condition that the agents can move one at a time. This is because in this Chapter we are only interested in finding a feasible solution to each problem, not necessarily an optimal one.

In Section 2.2 we focus on the *pebble motion on a tree* (PMT) [5, 58, 53, 52, 54], which is defined as follows. Let $T = (V, E)$ be a tree with n vertices and $k < n$ distinct pebbles, numbered $1, \ldots, k$, placed on distinct vertices. A *move* consists in transferring a pebble from its current position to an adjacent unoccupied vertex. The PMT consists in finding a sequence of moves that repositions all pebbles to assigned target vertices. In particular, as already mentioned, we focus on the problem of finding a feasible solution, not necessarily optimal. In this Section, we present three main contributions:

1. In Section 2.2.2, we propose the sub-optimal CATERPILLAR algorithm to solve the motion planning problem on a tree. It provides solutions with $O(nc)$ moves, where c is the maximum length of the corridors (i.e., paths whose internal nodes all have degree two, and whose end nodes have degree different from two). We are able to guarantee this complexity since, when we move the marked pebble p to its target, we only move the obstacles that are along the path of p, sliding them section by section from one subset of vertices to another, avoiding unnecessary motions.

2. In Section 2.2.3, we propose a sub-optimal algorithm for PMT (called *Leaves procedure for* PMT). The idea of the *Leaves procedure for* PMT is to use the CATERPILLAR algorithm to move the pebbles to the leaves of the tree, used as intermediate targets. At the end, we solve an unlabeled PMT instance, which brings the pebbles to the original targets. This procedure is simpler and easier to implement than the one proposed by Kornhauser and coauthors [8]. In addition, we prove a more detailed complexity result than the one provided by Kornhauser and coauthors. Indeed, our algorithm finds solutions with a number of moves $O(knc + n^2)$, which in the worst case is $O(n^3)$. Therefore, the number of moves

of these solutions depends on the tree structure and the number of pebbles.

3. In Section 2.2.4, we discuss the PMT *with trans-shipment vertices* (*ts*-PMT). This variant is relevant since a MAPF instance on a generic graph can be reduced to an instance of this problem. *ts*-PMT can be solved with the *Leaves procedure for* PMT with some minor modifications. This permits us to provide an upper bound for the solution length of MAPF on graphs.

In Section 2.3 we consider MAPF on strongly connected digraphs, a class that is more general than biconnected digraphs, already addressed in [6]. We present three main contributions:

1. In Section 2.3.3 we present a procedure, based on [53], that checks the problem feasibility in linear time with respect to the number of nodes (see Theorem 2.3.5). This approach can also be used to solve MAPF, but it leads to very redundant solutions.

2. In Section 2.3.4 we provide a necessary and sufficient condition for feasibility of any MAPF on a strongly connected digraph, adapted from [8] and Section 2.2.

3. In Section 2.3.5 we present diSC (digraph Strongly Connected) algorithm that finds a solution for all admissible problems with at least two holes, extending the method in [54] (see Section 2.3.1). The details of diSC are described in Appendix B. Moreover, in Section 2.3.6 we prove that the length complexity of this algorithm is $O(kn^2c)$.

In Section 2.4 we perform three sets of experiments to test the CATERPILLAR algorithm, the *Leaves Procedure* and the diSC algorithm. We test these algorithms on random scenarios generating by varying the parameters n, k and c, with the aim of verifying the theoretical results on their length complexity. Finally, we run diSC algorithm on a graph with 397 nodes representing the layout of a real warehouse, varying the number of agents from 2 to 10.

2.1 Problem definition

We consider the general case of a graph $G = (V, E)$, with vertex set V and edge set E. We are also given a set P of pebbles and a set H of holes, and each vertex of G is occupied either by a pebble or by a hole, so that $|V| = |P| + |H|$. A *configuration* is a function $\mathscr{A} : P \cup H \to V$ that assigns to each pebble or hole the vertex occupied by it. A configuration is *valid* if it is one-to-one (i.e., each vertex is occupied by one and only one pebble or hole). The collection $\chi \subset \{P \cup H \to V\}$ contains all valid configurations.

Given a configuration $\mathscr{A}$ and $u, v \in V$, we denote by $\mathscr{A}[u,v]$ the configuration obtained from $\mathscr{A}$ by exchanging the pebbles (or holes) placed at u and v:

$$\mathscr{A}[u,v](q) := \begin{cases} v, & \text{if } \mathscr{A}(q) = u; \\ u, & \text{if } \mathscr{A}(q) = v; \\ \mathscr{A}(q), & \text{otherwise .} \end{cases} \tag{2.1}$$

As mentioned in the Introduction, a *move* is the movement of a pebble along an edge. For each edge $(u, v) \in E$ we can define one or two possible moves, depending on the type of graph. If G is a directed graph, $u \to v$ is the only possible move, else if G is an undirected graph or a tree, the two ordered pairs $u \to v$ and $v \to u$ are two possible moves. We indicate with $\hat{E}$ the set of all the moves on G. Function $\rho : \chi \times \hat{E} \to \chi$ is a partially defined transition function such that $\rho(\mathscr{A}, u \to v)$ is defined if and only if v is empty (i.e., occupied by a hole). In this case $\rho(\mathscr{A}, u \to v)$ is the configuration obtained by exchanging the pebble or the hole in u with the hole in v. Notation $\rho(\mathscr{A}, u \to v)!$ means that the function is defined. In other words $\rho(\mathscr{A}, u \to v)!$ if and only if $(u, v) \in E$ and $\mathscr{A}^{-1}(v) \in H$. If $\rho(\mathscr{A}, u \to v)!$, then $\rho(\mathscr{A}, u \to v) = \mathscr{A}[u, v]$. Note that the hole in v moves along $v \to u$, while the pebble or hole on u moves on v.

We represent a *plan* as an ordered sequence of moves. It is convenient to view the elements of $\hat{E}$ as the symbols of a language. We denote by E^* the Kleene star of $\hat{E}$, that is the set of ordered sequences of elements of $\hat{E}$ with arbitrary length, together with the empty string ε:

$$E^* = \bigcup_{i=1}^{\infty} \hat{E}^i \cup \{\varepsilon\}.$$

We extend the function $\rho : \chi \times \hat{E} \to \chi$ to $\rho : \chi \times E^* \to \chi$, by setting $(\forall \mathscr{A} \in \chi)\rho(A,\varepsilon)!$ and $\rho(\mathscr{A},\varepsilon) = \mathscr{A}$. Moreover, $(\forall s \in E^*, e \in \hat{E}, \mathscr{A} \in \mathscr{C})\ \rho(\mathscr{A},se)!$ if and only if $\rho(\mathscr{A},s)!$ and $\rho(\rho(\mathscr{A},s),e)!$ and, if $\rho(\mathscr{A}se)!$, then $\rho(\mathscr{A}se) = \rho(\rho(\mathscr{A}s),e)$. Note that ε is the trivial plan that keeps all pebbles and holes on their positions. We denote by $|f|$ the length of a plan f (i.e., the number of moves of f). We define an equivalence relation $\sim$ on E^*, by setting, for $s,t \in E^*$, $s \sim t \leftrightarrow (\forall \mathscr{A} \in \mathscr{C})\,\rho(\mathscr{A},s) = \rho(\mathscr{A},t)$. In other words, two plans are equivalent if they reconfigure pebbles and holes in the same way. Given a configuration $\mathscr{A}$ and a plan f such that $\rho(\mathscr{A},f)!$, a plan f^{-1} is a *reverse* of f if $\mathscr{A} = \rho(\rho(\mathscr{A},f),f^{-1})$ (i.e., f^{-1} moves each pebble and hole back to their initial positions). We can also write $ff^{-1} \sim \varepsilon$, so that f^{-1} behaves like a right-inverse.

For undirected graphs and trees, the following result holds.

Proposition 2.1.1. *For any configuration $\mathscr{A} \in \chi$ and any plan $f \in E^*$, such that $\rho(\mathscr{A},f)!$, there exists a reverse plan f^{-1} such that $|f| = |f^{-1}|$.*

Proof. The thesis can be proved by induction as follows. If $f \in \hat{E}$, then there exist $u,v \in V$ such that $f = u \to v$ and $f^{-1} = v \to u$. Now, suppose that for any $f \in E^*$ such that $|f| = n > 1$ there exists a reverse f^{-1} such that $|f| = |f^{-1}|$. We prove that the thesis is verified also for each plan of length $n+1$. Indeed, if $|f| = n+1$ then $f = ge$ with $|g| = n$ and $e \in \hat{E}$. Therefore, there exist g^{-1} and e^{-1} the corresponding reverse plans such that $|g^{-1}| = |g|$ and $|e^{-1}| = |e|$, and we can define $f^{-1} := e^{-1}g^{-1}$ which is a reverse plan of f:

$$ff^{-1} = gee^{-1}g^{-1} \sim gg^{-1} \sim \varepsilon.$$

Moreover, $|f^{-1}| = |g^{-1}| + |e^{-1}| = n+1$. □

In Proposition 2.3.1 of Section 2.3.3 we will state that also in the case of a strongly connected digraph, each plan has a reverse plan.

2.1.1 Definitions of MAPF, MP and PMT

Our main problem is the Multi-Agent Path Finding (MAPF), which consists in finding a plan that re-positions all pebbles to assigned target vertices, avoiding collisions.

For this problem, and the ones we will present later, the position of the holes is not relevant. Therefore, we introduce an equivalence relation $\dot{\sim}$ between configurations

$$\mathscr{A}^1 \dot{\sim} \mathscr{A}^2 \iff \forall\, p \in P \quad \mathscr{A}^1(p) = \mathscr{A}^2(p),$$

and we indicate with $\tilde{\mathscr{A}^1}$ the equivalence class of $\mathscr{A}^1$.

Definition 2.1.1. *(**MAPF problem**). Let $G = (V,E)$ be a graph, P be a pebble set, $\tilde{\mathscr{A}^s}$ an initial valid configuration, $\tilde{\mathscr{A}^t}$ a final valid configuration. The MAPF instance $< G, \tilde{\mathscr{A}^s}, \tilde{\mathscr{A}^t} >$ consists in finding a plan f such that $\tilde{\mathscr{A}^t} = \rho(\tilde{\mathscr{A}^s}, f)$.*

A variant of MAPF is the *motion planning problem*, which consists in finding a plan such that a single marked pebble reaches a desired target vertex, avoiding collisions with other pebbles, that are movable *obstacles* [55, 57, 56, 58]. We recall its definition.

Definition 2.1.2. ***Motion Planning (MP) problem*** *. Let $G = (V,E)$ be a graph, P a set of pebbles. Given a pebble $p \in P$, an initial configuration $\tilde{\mathscr{A}^s}$, and $t \in V$, the motion planning problem (MP) consists in finding a plan f such that $t = \rho(\tilde{\mathscr{A}^s}, f)(p)$. We indicate such a plan with notation $\tilde{\mathscr{A}}(p) \Rightarrow v$. Moreover, we indicate a MP instance with $< G, s, t, \mathscr{O} >$, where $s = \tilde{\mathscr{A}^s}(p)$ is the initial position of p and $\mathscr{O} = \tilde{\mathscr{A}^s}(P \setminus \{p\})$ is the set of initial positions of the dynamic obstacles, i.e., of all the other pebbles.*

The only difference between MP and MAPF is the following one. In MP only one pebble is assigned a target while the others are only movable obstacles. Instead, in MAPF each pebble is assigned a final destination.

Reference [56] discusses the feasibility of the motion planning problem on a strongly connected digraphs and proves the following:

Theorem 2.1.1. *(Theorem 14 of [56]) Let D be a strongly connected digraph, P a set of pebbles and H a set of holes. Then any motion planning problem on D is feasible if and only if $|H| \geq 1$.*

If G is a tree, this problem is called *pebble motion on trees* (**PMT**). A variant of PMT problem is *pebble motion on trees with trans-shipment vertices* (**ts-PMT**). This problem is a PMT on a tree in which the vertex set V is partitioned in trans-shipment and regular vertices.

Definition 2.1.3. ***(PMT problem)**. Given a tree T, a pebble set P, an initial valid configuration $\tilde{\mathscr{A}}^s$, and a final valid configuration $\tilde{\mathscr{A}}^t$, find a plan f such that $\tilde{\mathscr{A}}^t = \rho(\tilde{\mathscr{A}}^s, f)$.*

We also focus on a relaxation of the PMT problem: the *unlabeled PMT problem*. This problem consists in finding a plan such that each pebble reaches any vertex belonging to the set of targets.

Definition 2.1.4. ***(Unlabeled PMT problem)**. Let $T = (V,E)$ be a tree, P a set of pebbles, and $\tilde{\mathscr{A}}^s$ an initial valid configuration. Given $D \subset V$, a set of destinations such that $|D| = |P|$, find a plan f such that $D = \rho(\tilde{\mathscr{A}}^s, f)(P)$.*

2.2 Pebble Motion on trees

Notation

Let $T = (V,E)$ be a tree with n nodes and let $u \in V$. We denote by $F(u)$ the connected components of the *forest* obtained from T by deleting u. For some $v \in V \setminus \{u\}$, $T(F(u),v)$ is the connected component containing v, while $C(F(u),v)$ is the set of remaining connected components of $F(u)$ excluding $T(F(u),v)$.

Given two nodes $a,b \in V$, we denote by π_{ab} the set of vertices of the unique path in T from a to b, and with $d(a,b)$ the length of this path. In particular, π_{ab} is a *corridor* if a and b have degree different from two, while the internal nodes of the path all have exactly degree two. Moreover, $\pi_{ab} \setminus \{a\}$ is the set of all vertices of π_{ab}, except for a. We denote by $C(T)$ the set of all corridors of T, and by c_1 the maximum corridor length:

$$c_1(T) = \max\{d(a,b) : \pi_{ab} \in C(T)\}.$$

Let $J \subset V$ be the set of *junctions* (i.e., nodes with degree greater than two). We define the subclass of corridors $\bar{C}(T) \subset C(T)$ that have only junctions as endpoints, and we denote by $c_2(T)$ the maximum length of this subclass of corridors. Obviously, $c_2(T) \leq c_1(T)$. Note that corridors in $C(T) \setminus \bar{C}(T)$ are those for which at least one endpoint is a leaf of the tree.

Let us define the distance between a node s and a subset of nodes $W \subset V$ as

$$d(s,W) = \min_{a \in W} d(s,a).$$

Furthermore, we define the distance of W_1 from W_2 as

$$d(W_1,W_2) = \sum_{a \in W_1} d(a,W_2).$$

Then, a *subset of V of cardinality q closest to W_1* is a set W such that

$$W \in \arg \min_{\substack{W \in \mathscr{P}(V) \\ |W|=|q|}} d(W,W_1),$$

where $\mathscr{P}(V)$ denotes the power set of V (i.e., the set of all its subsets).

Assumptions

In all the problems we focus on, we assume that the following assumption holds:

$$|H| \geq c(T), \tag{2.2}$$

where $c(T) \in \mathbb{N}$ is a constant depending on the structure of the tree T. In particular, if T is a path graph (i.e., a tree with two nodes of degree 1, and the remaining $n-2$ nodes of degree 2) $c(T) := c_1(T)$. Otherwise, in all other cases, $c(T)$ is defined as follows:

$$c(T) := \max\{c_1(T)+1, c_2(T)+2\}. \tag{2.3}$$

From now on, when we write c_1, c_2 and c without the indication of a tree within the parenthesis, it is given as understood that the three parameters refer to the tree T

on which we are solving the PMT problem, while for other trees we will explicitly indicate them within parenthesis .

Kornhauser and coauthors [8] showed that Assumption (2.2) is a necessary and sufficient condition for the feasibility of PMT on trees. Obviously, it follows that this condition is also sufficient for the feasibility of any instance of the unlabeled PMT problem, and of the motion planning problem.

Basic plans

Let T be a tree and $\mathscr{A}$ a configuration on it. Given a path $\pi_{vw} = v u_2 \cdots u_{n-1} u_n \equiv w$, we define the following plans:

1. If $w \in \mathscr{A}(H)$, BRING HOLE FROM w TO v is defined as

$$\alpha_{vw} = (u_{n-1} \to w, \ldots, v \to u_2). \tag{2.4}$$

   In other words, for each j from $n-1$ to 1, if there is a pebble on u_j, we move it on u_{j+1}. For instance, see the example of Figure 2.1.

2. If $v \in \mathscr{A}(P)$ and $\pi_{vw} \setminus \{v\} \subset \mathscr{A}(H)$, MOVE PEBBLE FROM v TO w is defined as

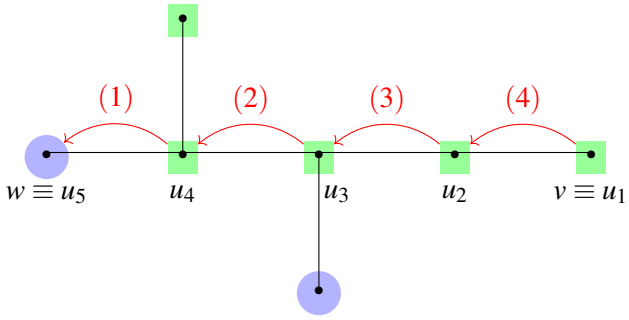
$$\beta_{vw} = (v \to u_2, \ldots, u_{n-1} \to w). \tag{2.5}$$

   For instance, see the example of Figure 2.2.

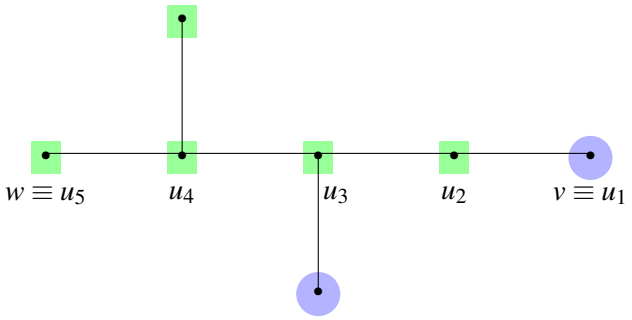
2.2.1 Unlabeled PMT problem

Let P be a set of unlabeled pebbles on a tree $T = (V, E)$. Let $\mathscr{A}^s$ be the initial configuration and D be the set of destinations. We denote by $S = \mathscr{A}^s(P)$ the set of pebbles initial positions. The goal of the *unlabeled pebble motion on trees* is to move each pebble from its initial position to any position of D. In the following, we introduce an algorithm presented by Kornhauser and coauthors [8].

1. If V is empty: terminate the procedure.



(a) Initial configuration. The red edges represent plan $\alpha_{vw} = (u_4 \to w, u_3 \to u_4, u_2 \to u_3, v \to u_2)$.



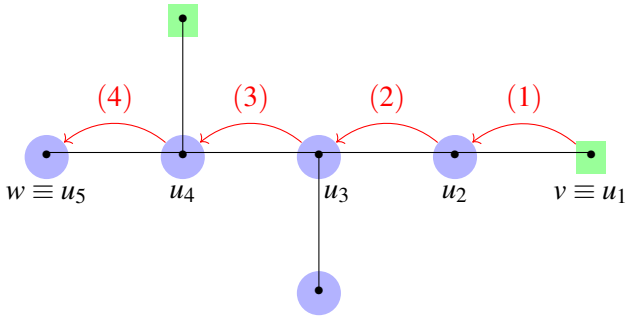
(b) Final configuration after bringing the hole from w to v.

Figure 2.1: Example of BRING HOLE FROM w TO v. Green squares represent pebbles, blue circles represent holes.

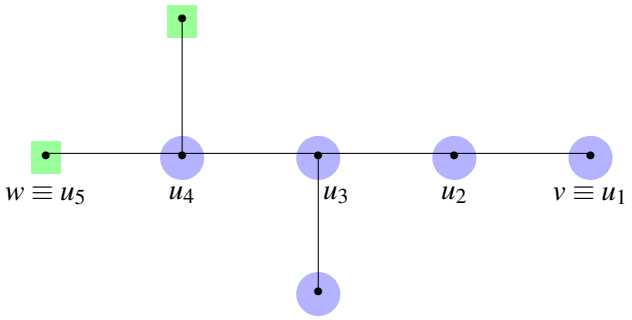
2. Select any leaf v of T.

   - If $v \in S \cap D$ or $v \notin S \cup D$, then "prune" v from T, i.e., $V = V \setminus \{v\}$, and set $S = S \setminus \{v\}$. Go to Step 1.
   - If $v \in D \setminus S$, select a pebble p on a vertex w such that

$$w \in \arg\min_{v' \in S} d(v', v).$$



(a) Initial configuration. The red edges represent plan $\beta_{vw} = (v \to u_2, u_2 \to u_3, u_3 \to u_4, u_4 \to w)$.



(b) Final configuration after moving the pebble from v to w.

Figure 2.2: Example of MOVE PEBBLE FROM v TO w. Green squares represent pebbles, blue circles represent holes.

Let p be the pebble on w. By definition of w, path π_{wv} contains only pebble p. Therefore, move p to v and update $S = S \setminus \{w\}$. Then, "prune" v from T, i.e., $V = V \setminus \{v\}$. Go to Step 1.

- if $v \in S \setminus D$. Find an unoccupied vertex u which is at minimum distance from v on T:

$$u \in \arg\min_{v' \in V \setminus S} d(v', v).$$

Then, path π_{vu} has pebbles on each vertex except u. Move each pebble on the path π_{vu} towards u with plan α_{vu} as defined in (2.4). This makes v unoccupied and u occupied. Then, set $S = (S \setminus \{v\}) \cup \{u\}$, "prune" v from T, i.e., $V = V \setminus \{v\}$, and go to Step 1.

Since at most n moves are made at each execution of Step 2, and Step 2 is executed n times (at each iteration the cardinality of V is decreased by one), the total number of required moves is at most n^2. Therefore, the complexity of this algorithm is $O(n^2)$.

Gather holes problem

In this subsection, we focus on a particular case of the *unlabeled* PMT problem: the *gather holes problem*. Let $T = (V, E)$ be a tree with n nodes, P a set of pebbles, and H the set of holes. Let $\bar{T} = (\bar{V}, \bar{E})$ be a subtree with $q = |\bar{V}| \leq |H|$. Then, *gather holes in* $\bar{T}$ consists in bringing q holes of the tree to the nodes of $\bar{T}$.

Definition 2.2.1. *(**Gather holes problem**). Let T be a tree and $\bar{T} = (\bar{V}, \bar{E})$ be a subtree. Let P be a set of pebbles, and $\tilde{\mathscr{I}}^s$ an initial valid configuration. Find a plan f such that $\bar{V} \cap \rho(\tilde{\mathscr{I}}^s, f)(P) = \emptyset$.*

Even if *gather-holes* can be solved by the previous algorithm, we present a specific procedure that allows finding a feasible solution with a lower time-complexity. The solution plan removes pebbles from vertices of $\bar{T}$, and replaces them with holes. To search for a short plan, it is convenient to bring holes that are already close to $\bar{V}$. Therefore, we choose the holes in a set M such that

$$M \in \arg\min_{W \subset \mathscr{I}^s(H):\ |W| = q} d(W, \bar{V}), \tag{2.6}$$

i.e., M is a subset of vertices with cardinality q closest to $\bar{V}$ and containing only holes of the initial configuration. Denote by $\tilde{H}$ the set of holes on M ($\mathscr{I}^s(\tilde{H}) = M$). Then,

we want to find a plan f such that $\bar{V} = \rho(\mathscr{A}^s, f)(\tilde{H})$. Moreover, let $\bar{H} = \{h \in \tilde{H} : \mathscr{A}^s(h) \notin \bar{V}\}$.

Denoting by $\bar{V}_P = \mathscr{A}^s(P) \cap \bar{V}$ the initial set of occupied vertices of $\bar{T}$, we can proceed as follows:

1. If $\bar{V}_P$ is empty: terminate the procedure;

2. Select $h \in \bar{H}$, let $v = \mathscr{A}^s(h)$ and $u \in \bar{V}_P$ be a closest node of $\bar{V}_P$ to v:

$$u \in \arg\min_{v' \in \bar{V}_P} d(v', v). \tag{2.7}$$

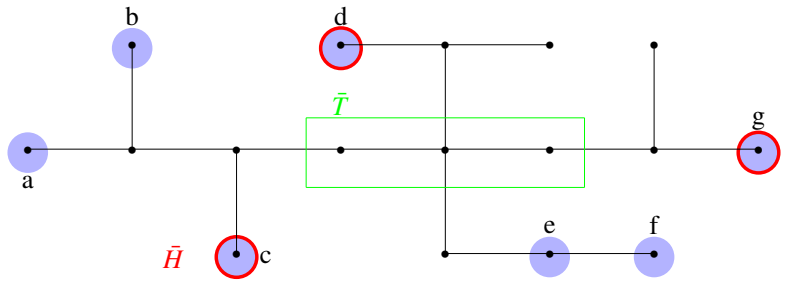
   Denote by p the pebble on u. If $\pi_{uv} \cap \bar{V} = \{u\}$, then move each pebble on the path π_{uv} towards v with plan α_{uv} defined in (2.4), and update $\mathscr{A}^s = \rho(\mathscr{A}^s, \alpha_{uv})$. Otherwise, let w be the closest node to v of $\pi_{uv} \cap \bar{V}$: move p from u to w with plan β_{uw} defined in (2.11) (note that $\pi_{uw} \setminus \{u\}$ contains only unoccupied vertices); then, move each pebble on path π_{wv} towards v with plan α_{wv}. Finally, update $\mathscr{A}^s = \rho(\mathscr{A}^s, \beta_{uw}\alpha_{wv})$. This makes u unoccupied.

3. Update $\bar{H} = \bar{H} \setminus \{h\}$ and $\bar{V}_P = \bar{V}_P \setminus \{u\}$. Go to Step 1.

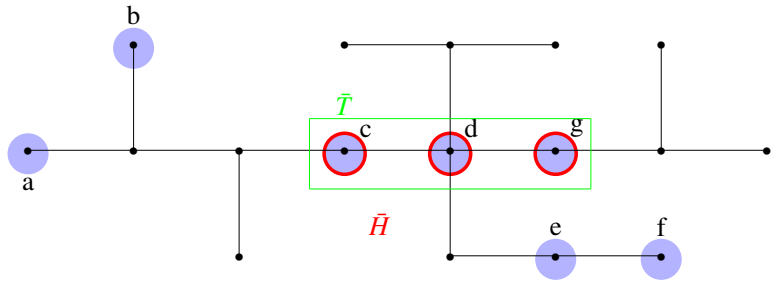
Since at most n moves (with $n = |V|$) are made at each execution of Step 2, and Step 2 is executed at most q times (with $q = |\bar{V}|$), (since the cardinality of $\bar{V}_P$ is reduced by one at each iteration), we have the following complexity result.

Proposition 2.2.1. *The length complexity of the solution provided by the gather holes procedure is* $O(nq)$.

Figure 2.3 presents an example of the execution of the procedure just described. The blue circles represent the holes, and $\bar{H} = \{c, d, g\}$ is a subset of holes closest to subtree $\bar{T}$ (Figure 2.3a). Another possible choice for $\bar{H}$ is, for example, $\bar{H} = \{c, e, g\}$. Figure 2.3b shows the final configuration, in which the holes of $\bar{H}$ have been moved to the subtree.



(a) Initial configuration. We choose $\bar{H} = \{c, d, g\}$ the subset of holes closest to subtree $\bar{T}$.



(b) Final configuration after moving holes c, d and g to $\bar{T}$.

Figure 2.3: Example of *Gather hole* problem. We want to move three closest holes to the subtree $\bar{T}$.

2.2.2 Motion planning on trees

Let $T = (V, E)$ be a tree, P a set of pebbles and H a set of holes. We assume that condition (2.2) of Section 2.2 holds. Recall that

$$c := \begin{cases} c_1, & \text{if } T \text{ is a path graph,} \\ \max\{c_1 + 1, c_2 + 2\}, & \text{otherwise,} \end{cases} \tag{2.8}$$

where c_1 is the maximum length of all the corridors and c_2 is the maximum length of the corridors with endpoints that are junctions. Let $\mathscr{A}^s$ be an initial valid configuration. Given a pebble $\bar{p}$ on $r = \mathscr{A}^s(\bar{p})$, and a target node $t \in V$, we show how to find a plan f such that $t = \rho(\mathscr{A}^s, f)(\bar{p})$. To do that, we analyze two cases:

A. $|\mathscr{A}^s(H) \cap T(F(r),t)| \geq c$, i.e., $T(F(r),t)$ contains at least c holes.

B. $|\mathscr{A}^s(H) \cap T(F(r),t)| < c$, i.e., $T(F(r),t)$ contains less than c holes.

For each of the two cases we present a solution procedure (*Procedure A* and *Procedure B*). The union of these procedures constitutes an algorithm to solve any instance of motion planning on trees, called the CATERPILLAR algorithm.

Case A. $T(F(r),t)$ contains at least c holes.

The main idea of the algorithm is to clear, piece by piece, the path that goes from r to t, allowing the pebble to reach the target. We identify intermediate junctions on π_{rt} (denoted by i_k), and "parking" positions (denoted by ℓ_k) which are neighbor nodes of i_k, but do not belong to π_{rt}. The pebble moves from one parking position to the next one, until it reaches the target. When the pebble is on ℓ_k, we move out of the way all the obstacles that are on $\pi_{i_k i_{k+1}}$, so that we can freely move the pebble from ℓ_k to ℓ_{k+1}. We identify a sequence of subsets of vertices on which the movement of the pebble from ℓ_k to ℓ_{k+1} will be defined: each of them will contain the path from i_k to i_{k+1} combined with the parking positions ℓ_k and ℓ_{k+1}. These subsets $(S_0,\ldots,S_m)$, called *caterpillar sets*, are such that:

- the restriction of T to S_k is a connected subtree for all $k = 0,\ldots,m$;
- $|S_k| = c+1$ for all $k = 0,\ldots,m-1$, and $|S_m| \leq c+1$;
- $s \in S_0$ and $t \in S_m$;
- $|S_k \cap S_{k+1}| \geq 2$ for all $k = 0,\ldots,m-1$, i.e., two consecutive sets have at least two nodes in common.
- $S_k \cap S_{k+1} \cap J \neq \emptyset$ for all $k = 0,\ldots,m-1$, i.e., two consecutive sets have at least one junction in common.

These properties guarantee that there are enough holes to clear the path and move the pebble from one parking position to the next one.

Construction of caterpillar sets

Along path π_{rt} we select the node triple (i_k, j_k, ℓ_k): i_k and j_k represent the ends of a caterpillar set, while ℓ_k is a parking position. We proceed as follows:

1. Let $\ell_0 = r$, i_0 be the neighbor of r that belongs to π_{rt}, and:
   - if $d(i_0,t) \leq c-1$, set $j_0 = t$, $m = 0$ and stop;
   - otherwise, let j_0 be the node on π_{rt} such that $d(i_0, j_0) = c-2$. Set $k = 0$, $j_{-1} = i_0$ and go to Step 2.
2. let i_{k+1} be the closest junction to j_k on $\pi_{j_{k-1}j_k} \setminus \{j_{k-1}\}$, and ℓ_{k+1} be one of the neighbors of i_{k+1} not belonging to π_{rt}:
   - if $d(i_{k+1},t) \leq c-1$, set $j_{k+1} = t$, $m = k+1$ and stop;
   - otherwise, let j_{k+1} be the node on $\pi_{i_{k+1}t}$ such that $d(i_{k+1}, j_{k+1}) = c-2$. Set $k = k+1$ and repeat Step 2.

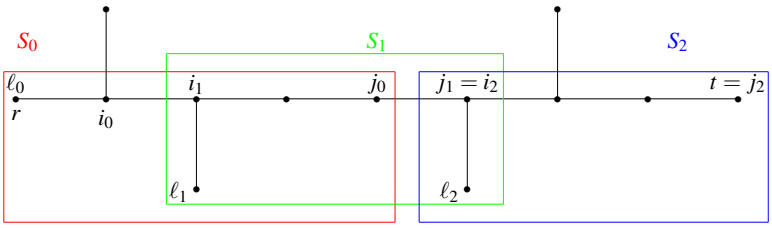


Figure 2.4: We consider the motion planning problem with source vertex r and target vertex t on a tree with $c = 5$. S_0, S_1 and S_2 are the *caterpillar sets* along path π_{rt}.

Remark 2.2.1. *Note that i_{k+1} is always well defined. Indeed, $d(i_k, j_k) = c-2 \geq c_2$ guarantees that there exists a junction on $\pi_{i_k j_k} \setminus \{i_k\}$ ($i_{k+1} \in \pi_{i_k j_k} \setminus \{i_k\}$). Since, by definition, i_k is the closest junction to j_{k-1} on $\pi_{j_{k-2} j_{k-1}} \setminus \{j_{k-2}\}$, then $i_{k+1} \notin \pi_{i_k j_{k-1}}$. Therefore, there exists $i_{k+1} \in \pi_{j_{k-1} j_k} \setminus \{j_{k-1}\}$ which is the closest junction to j_k.*

Observation 2.2.1. *Since $d(i_k, j_k) = c-2$, it follows that*

$$d(i_k, i_{k+2}) = d(i_k, j_k) + d(j_k, i_{k+2}) \geq c-1.$$

Now, let us find a lower bound for

$$d(i_0, i_m) = \sum_{k=0}^{m-1} d(i_k, i_{k+1}).$$

If m is even,

$$\sum_{k=0}^{m-1} d(i_k, i_{k+1}) = \sum_{k=0}^{\frac{m}{2}-1} [d(i_{2k}, i_{2k+1}) + d(i_{2k+1}, i_{2k+2})] =$$

$$= \sum_{k=0}^{\frac{m}{2}-1} d(i_{2k}, i_{2k+2}) \geq \frac{m}{2} \cdot (c-1).$$

Otherwise, if m is odd,

$$\sum_{k=0}^{m-1} d(i_k, i_{k+1}) = \left(\sum_{k=0}^{m-2} d(i_k, i_{k+1}) \right) + d(i_{m-1}, i_m)$$

$$\geq \frac{m-1}{2} \cdot (c-1).$$

Therefore,

$$m \leq 2 \cdot \frac{d(i_0, i_m)}{c-1} + 1 \leq 2 \cdot \frac{\delta}{c-1} + 1,$$

where δ is the diameter of the tree. If $c \geq 2$, it follows that $m = O(\frac{\delta}{c})$. Note that $c = 1$ only holds for the trivial case of the tree with 2 edges.

Nodes (i_k, j_k, ℓ_k) are used to delimit the borders of the caterpillar sets (see Figure 2.4), which are defined as follows:

$$S_k = \pi_{i_k j_k} \cup \{\ell_k\} \cup \{\ell_{k+1}\}, \quad \forall k = 0, \ldots, m-1,$$

$$S_m = \pi_{i_m j_m} \cup \{\ell_m\}.$$

We can easily note that the union of all caterpillar sets is a caterpillar tree (i.e., a tree in which all the vertices are within distance 1 from a central path). Moreover, all the properties of the caterpillar sets are verified:

- S_k is a connected component of T, and so it is a subtree;
- $|S_k| = |\pi_{i_k j_k}| + 2 = c + 1$ for all $k = 0, \ldots, m-1$ and $|S_m| = |\pi_{i_m j_m}| + 2 \leq c + 1$;
- $\{i_{k+1}, l_{k+1}\} \subset S_k \cap S_{k+1}$ and $i_{k+1} \in J$, so $|S_k \cap S_{k+1}| \geq 2$ and $S_k \cap S_{k+1} \cap J \neq \emptyset$ for $k = 0, \ldots, m-1$.

Moreover, it holds that

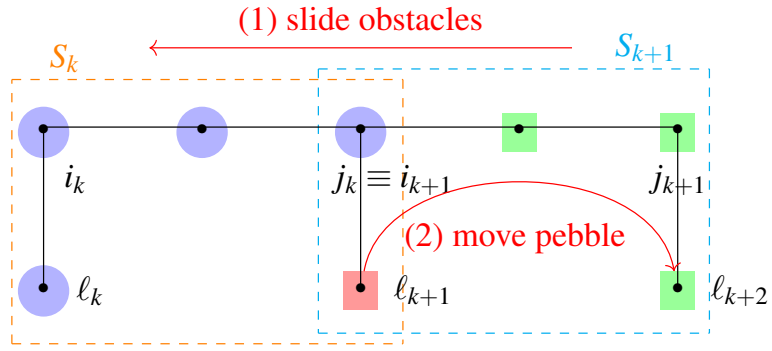
$$|S_{k+1} \cup S_k| = |S_{k+1}| + |S_k| - |S_{k+1} \cap S_k| \leq$$

$$\leq (2c+2) - 2 = 2c.$$

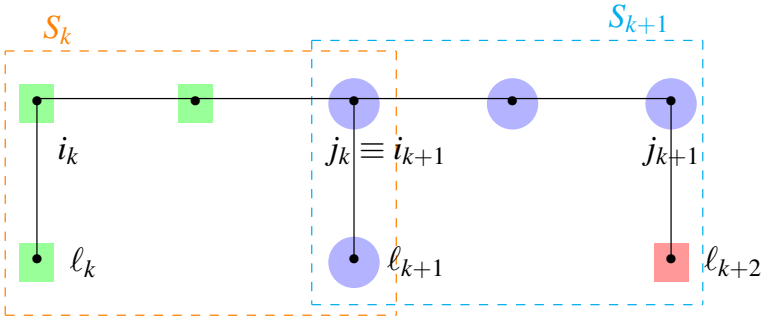
We are now ready to describe the procedure for solving the motion planning problem in case A.

Procedure A

1. Gather holes in $S_0 \setminus \{\ell_0\}$: $O(nc)$ moves in view of Proposition 2.2.1.
2. Move pebble from ℓ_0 to ℓ_1: c moves.
3. For all $k = 0, .., m-1$ moves the holes from $S_k \setminus \{\ell_{k+1}\}$ to $S_{k+1} \setminus \{\ell_{k+1}\}$ and move pebble p from ℓ_{k+1} to t or ℓ_{k+2} (see Figure 2.5). The former operation consists in sliding the obstacles from $S_{k+1} \setminus S_k$ to $S_k \setminus S_{k+1}$: this is equivalent to *gather holes in* S_{k+1} in the subtree obtained by the restriction of T to $S_{k+1} \cup S_k$, and so it requires at most $(|S_{k+1} \cup S_k| \cdot c) = O(c^2)$ moves. The last operation requires at most c moves. Overall the `for` cycle has length complexity $O(mc^2)$.



(a) Initial configuration. First we have slide the obstacles from $S_{k+1} \setminus S_k$ to $S_k \setminus S_{k+1}$, then we move the pebble from ℓ_{k+1} to ℓ_{k+2}.



(b) Final configuration.

Figure 2.5: Example of execution of an iteration of the `for` cycle of Step 3 of *Procedure A*. Blue circles are holes, green squares are obstacles and the red square is the marked pebble.

Note that the operations just described in *Procedure A* are all feasible because the properties of caterpillar sets hold.

Then, we proved the following result.

Proposition 2.2.2. *The length complexity of the solution provided by Procedure A is* $O(nc)$.

Proof. The solution provided by Procedure A requires

$$O(nc) + O(mc^2)$$

moves. Equivalently, the length complexity of the solution is

$$O(nc) + O(\delta c) = O(nc),$$

recalling that $m = O\left(\frac{\delta}{c}\right)$.

□

Case B. $T(F(s),t)$ contains less than c holes.

Suppose that q holes are missing to get to c. In this case, we create a neighborhood of s made up of q holes in $C(F(s),t)$, and then we move the pebble p to a node v such that $T(F(v),t)$ contains at least c holes. Let $z_1,\dots,z_k$ the neighbors of s which are not in $T(F(s),t)$. For all $j = 1,\dots,k$, we consider $T_j := T(F(s),z_j)$, the subtree containing z_j, and V_j the corresponding set of nodes. In this case the motion planning problem is solved through the following procedure.

Procedure B

1. Set $j = 1$;
2. Let $q_j = |\mathscr{A}^s(H) \cap V_j|$ be the number of holes in T_j:
   - if $q_j \leq q$, gather all the holes of T_j in H_j, which is a subset of V_j of cardinality q_j closest to the subset $\{s\}$:

$$H_j \in \arg \min_{\substack{W \in \mathscr{P}(V_j) \\ |W| = q_j}} d(W, \{s\});$$

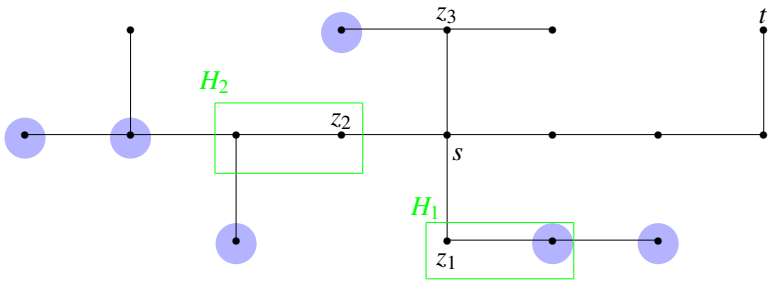
   Set $j = j+1$, $q = q - q_j$ and go back to Step 2;

- otherwise, gather q holes of T_j in H_j which, in this case, is defined as a subset of V_j of cardinality q closest to s.

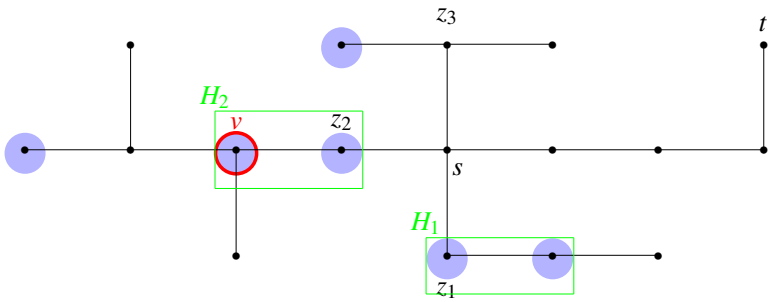
3. choose $v \in H_j$ which has the maximum distance from s and move pebble p on v;

4. reinitialize s with node v and apply *Procedure A*.

Note that Step 4 is feasible because in $T(F(v),t)$ there are certainly at least c holes.



(a) Initial configuration.



(b) Final configuration after *Procedure B*.

Figure 2.6: Example of situation of Case B with $q = 4$ missing holes.

Let n_j be the number of nodes of T_j. Then, recalling the length complexity of the gather holes procedure, *Procedure B* (without the final application of *Procedure A*) requires at most $\left(\sum_{j=1}^{k} n_j q_j\right)$ moves to bring the holes in the neighborhood of s, and at most n moves to bring pebble p on v. Therefore, the length complexity of this procedure is $O(nq)$ which, in the worst case, is $O(nc)$. The final application of *Procedure A* does not modify the complexity result. Thus, we proved the following result.

Proposition 2.2.3. *The complexity of Procedure B is $O(nc)$.*

2.2.3 Pebble motion on trees

Now we are ready to provide a procedure for the solution of PMT. We are given a tree $T = (V,E)$, a pebble set P, an initial valid configuration $\mathscr{A}^s$, and a final valid configuration $\mathscr{A}^t$. As already mentioned, the PMT problem consists in generating a plan f such that $\mathscr{A}^t(p) = \rho(\mathscr{A}^s, f)(p)\ \forall\, p \in P$. We will use the following strategy. First, we solve an unlabeled PMT on T to bring the pebbles to an ordered set of *intermediate targets*, $\bar{t}_1, \ldots, \bar{t}_{|P|}$. Later, we will discuss the choice of these targets. We call g the corresponding plan. Note that we do not apply this plan, but will apply the inverse plan g^{-1} at the end of the procedure. Then, we solve a set of $|P|$ motion planning problems on a sequence T_k of subtrees of T. Over each subtree, we use the CATERPILLAR algorithm to move each pebble $p \in P$ to one intermediate target t_k (that occupies a leaf of T_k). After the execution of the corresponding plan, we remove target v from tree T_k, obtaining tree T_{k+1}, and remove p from the set of pebbles. Finally, we apply the inverse plan g^{-1} on T.

We choose the *intermediate targets* such that trees T_k satisfies the following properties, for $k = 1, \ldots, |P| - 1$:

- T_k contains $\bar{t}_k, \bar{t}_{k+1}, \ldots, \bar{t}_{|P|}$ but does not contain $\bar{t}_1, \ldots, \bar{t}_{k-1}$;
- $c(T_k) \geq c(T_{k+1})$.

Let $L(T_k)$ be the set of all the leaves of the tree T_k. We denote by $V_1 = \{\bar{t}_k : k \in \{1,\dots,|P|\}\}$ the set of intermediate targets. We define them by the following procedure:

1. Set $k = 1$.
2. If $k > |P|$, stop.
3. Select $v \in L(T_k)$ such that, given the tree T_k^v obtained by removing v from T_k, it holds that $c(T_k^v) \leq c(T_k)$.
4. Define $\bar{t}_k = v$ and $T_{k+1} = T_k^v$. Set $k = k+1$ and to go Step 2.

The following proposition shows that Step 3 of the above procedure is well defined, i.e., it is always possible to find a leaf v of T_k such that $c(T_k^v) \leq c(T_k)$. Note that the result is valid for general trees and not only for the subtrees T_k generated by the above procedure.

Proposition 2.2.4. *For all $k \in \{1,\dots,|P|-1\}$, there exists $v \in L(T_k)$ such that $c(T_k^v) \leq c(T_k)$, where T_k^v is the tree obtained by removing v from T_k.*

Proof. For each $w \in L(T_k)$ we denote by n_w the unique neighbor of w, and by $deg(n_w)$ its degree. We define three subsets of leaves: $L_2(T_k) := \{w \in L(T_k) : deg(n_w) = 2\}$, $L_3(T_k) := \{w \in L(T_k) : deg(n_w) = 3\}$, $L_4(T_k) = \{w \in L(T_k) : deg(n_w) > 3\}$. Note that $L_2(T_k) \cup L_3(T_k) \cup L_4(T_k) = L(T_k)$. We choose v as follows (see also Figure 2.7):

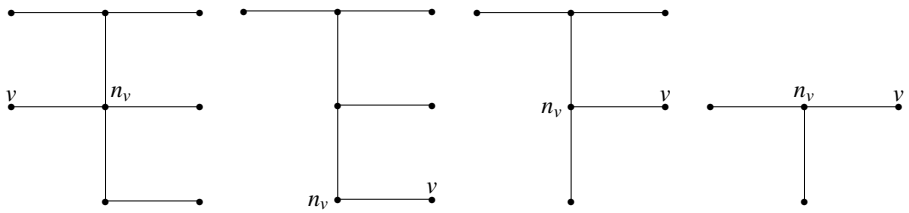
- if $L_4(T_k) \neq \emptyset$, we choose $v \in L_4(T_k)$: in this case $c(T_k^v) = c(T_k)$, since by removing v, we just remove a corridor of length 1 (see Figure 2.7a);
- else, if $L_2(T_k) \neq \emptyset$, we choose $v \in L_2(T_k)$: in this case $c_1(T_k^v) \leq c_1(T_k)$ and then $c(T_k^v) \leq c(T_k)$, since by removing v, we are reducing by one the length of one corridor in $C(T_k) \setminus \bar{C}(T_k)$ (see Figure 2.7b);
- otherwise, all corridors in $C(T_k) \setminus \bar{C}(T_k)$ have length one. We choose $v \in L_3(T_k)$. Two cases are possible:

1. $\bar{C}(T_k) = \emptyset$, so that T_k is a star graph with a central node and three neighboring leaves. It holds that $c(T_k) = 2$. Removing one leaf v, T_k^v is a path graph of length 2, therefore $c(T_k^v) = 2 = c(T_k)$ (see Figure 2.7d).
2. $\bar{C}(T_k) \neq \emptyset$: then, it holds that $c_1(T_k) = c_2(T_k)$ and, consequently, $c(T_k) = c_2(T_k) + 2$. Moreover, there exists at least one corridor in $\bar{C}(T_k)$ such that one of its endpoints is a junction connected to exactly two leaves, since $L_2(T_k) = L_4(T_k) = \emptyset$. We choose v between one of the two leaves (see Figure 2.7c). In this case, $c_1(T_k^v) \leq c_2(T_k) + 1$ and $c_2(T_k^v) \leq c_2(T_k)$, therefore

$$c(T_k^v) = \max\{c_1(T_k^v) + 1, c_2(T_k^v) + 2\} \leq$$

$$\leq \max\{(c_2(T_k) + 1) + 1, c_2(T_k) + 2\} = c(T_k).$$

□



(a) Case $v \in L_4(T_k)$. (b) Case $v \in L_2(T_k)$. (c) Case $v \in L_3(T_k)$, $\bar{C}(T_k) \neq \emptyset$. (d) Case $v \in L_3(T_k)$, $\bar{C}(T_k) = \emptyset$.

Figure 2.7: The four cases of Proposition 2.2.4.

We propose the following *Leaves procedure for PMT*, which breaks down the PMT problem into an unlabeled problem and a series of motion planning problems, and find plan f for any PMT instance.

Leaves procedure for PMT

1. Let V_1 be the set of intermediate targets found with the previous procedure. Find a plan g which solves the unlabeled PMT problem from the final configuration

$\mathscr{A}^t$ to V_1, i.e., such that

$$V_1 = \rho(\mathscr{A}^t, g)(P),$$

and let $\mathscr{A}^{\bar{t}} := \rho(\mathscr{A}^t, g)$ be the *intermediate configuration*. Note that for all $k \in \{1, \dots, |P|\}$, g would move a pebble from t_{i_k} to the intermediate target $\bar{t}_k$.

2. Set $k = 1$ and perform the following procedure:
   (a) if $k > |P|$, stop;
   (b) using CATERPILLAR algorithm, solve the motion planning for pebble p_{i_k} from $\mathscr{A}^s(p_{i_k})$ to $\bar{t}_k$, i.e., find a plan f_k, over the tree T_k obtained from T by removing nodes $\bar{t}_1, \dots, \bar{t}_{k-1}$, such that $\bar{t}_k = \rho(\mathscr{A}^s, f_k)(p_{i_k})$;
   (c) update $\mathscr{A}^s = \rho(\mathscr{A}^s, f_k)$;
   (d) set $k = k+1$ and go back to Step a).
3. Apply g^{-1}, the inverse plan of g, which, for all $k \in \{1, \dots, |P|\}$, moves pebble p_{i_k} from $\bar{t}_k$ to t_{i_k}.

Therefore, plan f which solves a given PMT instance is

$$f = f_1 f_2 \cdots f_{|P|}\, g^{-1}. \tag{2.9}$$

The complexity of the proposed procedure is stated in the following theorem.

Theorem 2.2.1. *The length complexity of the proposed procedure for the solution of the PMT problem is* $O(|P|nc + n^2)$.

Proof. Note that f_k requires at most $O(nc)$ moves for all $k = 1, \cdots, |P|$. Indeed, each f_k is the solution of a motion planning problem, which, in view of Propositions 2.2.2 and 2.2.3, is solved in $O(nc)$ moves. Moreover, g^{-1} requires at most n^2 moves. Indeed, g is the solution of an unlabeled PMT, which requires $O(n^2)$ moves, as seen in Section 2.2.1. Since g^{-1} is obtained from g by reversing its moves, it has the same length (see Observation 2.3.1). Then, the total number of moves is

$$O(|P|nc) + O(n^2).$$

□

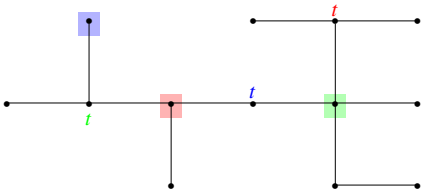
Figure 2.22 provides an example of application of the *Leaves procedure* for a PMT instance with three pebbles.

An interesting property of the *Leaves procedure for* PMT concerns the number of times each vertex is traversed by pebbles. As we further discuss in Section 2.2.4, the pebble motion problem on general graphs can be solved after converting it on a variant of PMT over trees. The bound on the number of times each vertex is crossed by the pebbles in the PMT problem over trees provided by the following proposition, allows deriving a complexity result also for the pebble motion problem over general graphs. Such complexity result will be used in the next Section.

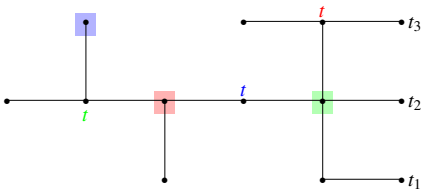
Proposition 2.2.5. *In any solution provided by the proposed procedure, each vertex is crossed $O(|P|c)$ times by the pebbles.*

Proof. Let us count how many times each vertex is crossed in the solutions of each problem:

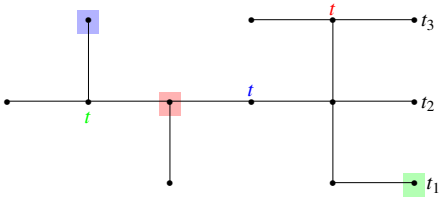
1. *Basic plans*.
   - BRING HOLE FROM w TO v (α_{vw}): each pebble along path π_{vw} moves forward one position, therefore each vertex is crossed at most once.
   - MOVE PEBBLE FROM v TO w (β_{vw}): one pebble moves on the path π_{vw}, therefore each vertex is crossed at most once.
2. *Unlabeled* PMT *problem*. In the solution of the *unlabeled* problem proposed in Section 2.2.1, each vertex is crossed at most once by each pebble for a total of $O(|P|)$ times in the overall procedure.
3. *Gather c holes*. In each iteration of Step 2 of the procedure described in Section 2.2.1, each vertex is crossed at most once because it performs α_{uv} or $\beta_{uw}\alpha_{wv}$. Since Step 2 is executed at most c times, each vertex is crossed $O(c)$ times.
4. CATERPILLAR algorithm. Each vertex is crossed $O(c)$ times, indeed:



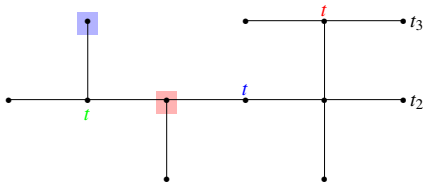
(a) Initial configuration.



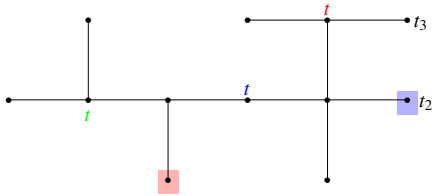
(b) Define intermediate targets t_1, t_2, t_3.



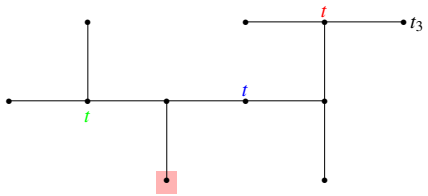
(c) Move green pebble to intermediate target.



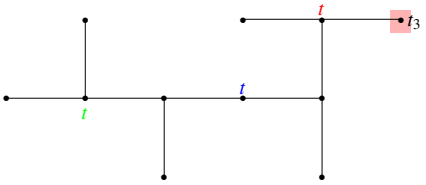
(d) Remove t_1.



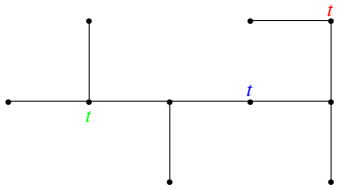
(e) Move blue pebble to intermediate target.



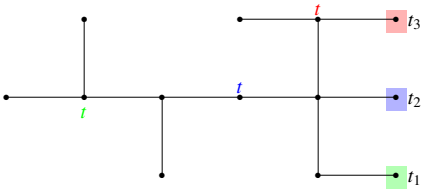
(f) Remove t_2.



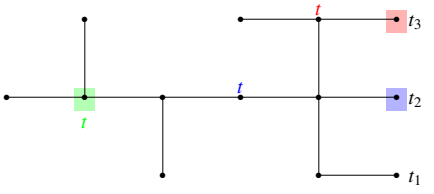
(g) Move red pebble to intermediate target.



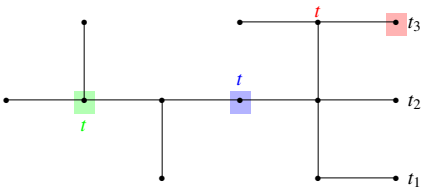
(h) Remove t_3



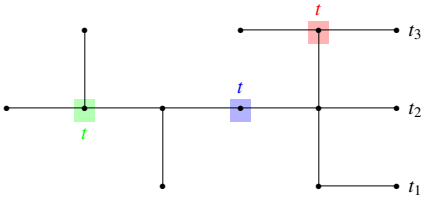
(i) Intermediate configuration.



(j) Move green pebble to final target.



(k) Move blue pebble to final target.



(l) Move red pebble to final target.

Figure 2.8: Example of application of the *Leaves procedure*.

(a) *Procedure A*. In Step 1 we solve a *gather hole problem*, therefore by point (2) each vertex is crossed $O(c)$ times. In Step 3 each vertex that belongs to a caterpillar set S_k is traversed once time by the pebble and once time by at most c obstacles that arrived from S_{k+1} and then moved to S_{k-1}. Therefore, in *Procedure A* each vertex is crossed $O(c)$ times.

(b) *Procedure B*. In Step 2 we solve a *gather hole problem* with q_j holes, therefore each vertex is crossed $O(q_j)$ times. Since $\sum_{j=1}^{k} q_j = q \leq c$, each vertex is crossed at most $O(c)$ times.

5. *Leaves Procedure.* A solution of PMT given by this procedure is $f = f_1 f_2 \cdots f_{|P|}\, g^{-1}$ (see (2.9)). For each $k \in \{1, \ldots, |P|\}$, f_k is the solution of a motion planning problem provided by the CATERPILLAR algorithm, therefore each vertex is crossed $O(c)$ times. Moreover, g^{-1} is the inverse of the solution of an unlabeled problem: therefore each vertex is crossed $O(|P|)$ times. We can conclude that in the solution plan of PMT each vertex is crossed

$$|P|O(c) + O(|P|) = O(|P|c)$$

times.

□

2.2.4 PMT with Trans-shipment vertices

The more general MAPF problem can always be reduced to a variant of the PMT (called *ts*-PMT), both in the case of undirected graphs [53, 54] and of directed graphs [**?**]. Given a graph G, it is possible to convert it into a tree (called *biconnected component tree*), adding a new type of vertex called *trans-shipment* [53, 54]. In particular, each biconnected component of the graph is converted into a star subgraph, whose internal vertex is a trans-shipment (see Figure 2.9). This way, the original problem on graph G is converted into a problem over a tree with trans-shipment vertices. Once a solution of the problem over the tree is obtained, this can be converted back into a solution for the original problem.

(a) Biconnected component.

(b) Star subgraph with a trans-shipment internal vertex (diamond shape).

Figure 2.9: Conversion of a biconnected component of a graph into a star subgraph.

For this reason, we need to study a variant of the PMT problem, the *pebble motion on trees with trans-shipment vertices* (*ts*-PMT), which is a PMT problem on a tree such that the vertex set is partitioned in trans-shipment and regular vertices.

Definition 2.2.2. *A trans-shipment vertex is a vertex with degree greater than one that cannot host a pebble: pebbles can cross this node, but cannot stop there. More formally, given a trans-shipment vertex s,*

1. $deg(s) \geq 2$*;*
2. $\rho(\mathscr{A}, (u \to s)(w \to v))!$ *if and only if* $w = s$, $(u,s),(s,v) \in E$, *and* $\mathscr{A}^{-1}(v) \in H$. *If* $\rho(\mathscr{A}, (u \to s)(s \to v))!$, *then* $\rho(\mathscr{A}, (u \to s)(s \to v)) = \mathscr{A}[u,v]$.

The second property means that, if a pebble is moved to a trans-shipment vertex, then it must be immediately moved to another node.

We denote by V_T the set of all the trans-shipment vertices and $V_R = V \setminus V_T$ the set of regular vertices. We require that V_T satisfies the following property

$$\forall\, v,w \in V_T \qquad d(v,w) > 1. \tag{2.10}$$

This assumption is motivated by the fact that trans-shipment vertices are the internal vertices of the stars, so they cannot be adjacent to each other.

Now we can formally define the PMT problem with trans-shipment vertices as follows.

Definition 2.2.3. *(**PMT problem with trans-shipment vertices**). Let $T = (V,E)$ be a tree with $V = V_R \cup V_T$, where the set of trans-shipment vertices V_T is such that (2.10) holds. Given a pebble set P, initial and final valid configurations $\tilde{\mathscr{A}^s}$, $\tilde{\mathscr{A}^t}$ such that $\tilde{\mathscr{A}^s}(P), \tilde{\mathscr{A}^t}(P) \subset V_R$, find a plan f such that $\tilde{\mathscr{A}^t} = \rho(\tilde{\mathscr{A}^s}, f)$.*

This problem can be solved with the same procedure described in Section 2.2.3. However, some changes need to be made to ensure that the second property of Definition 2.2.2 is fulfilled. In the next subsections we show the changes we need to introduce into the previous procedures to address the presence of trans-shipment vertices.

Basic plans

We generalize the definition of the plan BRING HOLE FROM w TO v to the case in which there is a trans-shipment vertex on the path π_{vw}. For instance, if $\pi_{vw} = v u_2 \cdots u_i \cdots u_{n-1} u_n \equiv w$ such that $u_i \in V_T$, then α_{vw} is defined as follows

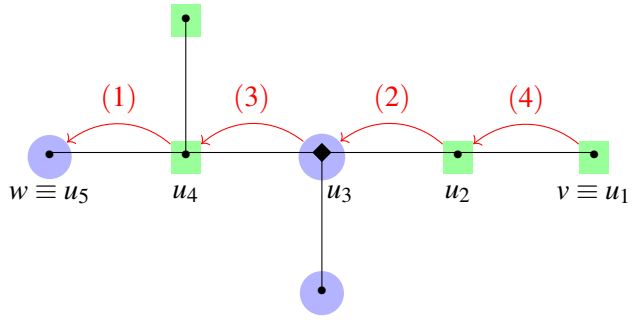
$$(u_{n-1} \to w, \ldots, u_{i-1} \to u_i, u_i \to u_{i+1}, \ldots, v \to u_2). \tag{2.11}$$

In other words, the only difference from the previous definition is that if a pebble move on u_i, then it immediately moves to u_{i+1}. For instance, see the example of Figure 2.10, where node u_3 is a trans-shipment vertex.

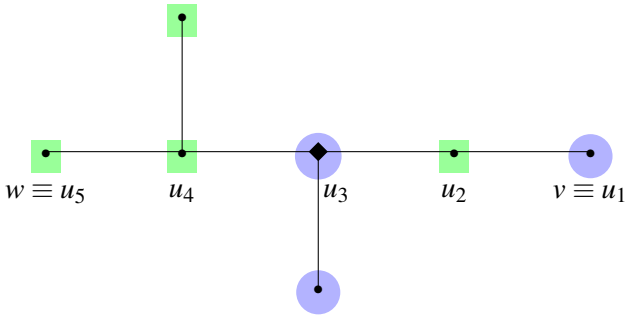
Observation 2.2.2. *Note that α_{vw} can be defined only if $w \in \mathscr{A}(H) \cap V_R$, which means that it is not allowed to bring hole from a trans-shipment vertex. Indeed, this could imply that in the final configuration a pebble lands on w. For the same reason plan* MOVE PEBBLE FROM v TO w *(i.e., β_{vw}), which in this case does not change, can be defined only if $w \in \mathscr{A}(H) \cap V_R$.*

Assumption

Observation 2.2.2 implies that the main difference of the new algorithm is that the holes on the trans-shipment vertices cannot be used in bring hole and gather hole operations, which are the basis for all the procedures that constitute the algorithm



(a) Initial configuration. The red edges represent plan $\alpha_{vw} = (u_4 \to w, u_2 \to u_3, u_3 \to u_4, v \to u_2)$. Node u_3 (identified by the diamond shape) is a trans-shipment vertex



(b) Final configuration after bringing the hole from w to v.

Figure 2.10: Example of BRING HOLE FROM w TO v. Vertex u_3 is a trans-shipment. Green squares represent pebbles, blue circles represent holes.

to solve PMT. For this reason, we define a new distance $\tilde{d}$ which does not take into account trans-shipment vertices. Given a path π_{uv}, $\tilde{d}(u,v)$ counts how many regular vertices belong to the path:

$$\tilde{d}(u,v) := |\pi_{uv} \cap V_R|.$$

Consequently, we also define $\tilde{c}_1$ and $\tilde{c}_2$ which count corridor lengths according to the new definition of distance:

$$\tilde{c}_1 := \max\{\tilde{d}(a,b) : \pi_{ab} \in C(T)\},$$

$$\tilde{c}_2 := \max\{\tilde{d}(a,b) : \pi_{ab} \in \bar{C}(T)\}.$$

Moreover, we define $\tilde{c} := \tilde{c}_1$ in the case of a path graph, $\tilde{c} := \max\{\tilde{c}_1 + 1, \tilde{c}_2 + 2\}$ otherwise. We note that on a tree with $V_T = \emptyset$, it holds that $d(u,v) = \tilde{d}(u,v) - 1$ and $c = \tilde{c} - 1$. Thus, to ensure the feasibility of any ts-PMT instance, at least $\tilde{c} - 1$ holes on regular vertices and $|V_T|$ holes for all trans-shipment vertices are needed. Therefore, Assumption (2.2) becomes

$$|H| \geq |V_T| + \tilde{c} - 1. \tag{2.12}$$

Unlabeled PMT with trans-shipment vertices

To solve this problem, we use the same procedure described in Section 2.2.1 to solve the classical *Unlabeled* PMT. The only difference is in Step 2 in the case vertex v is a source but not a target ($v \in S \setminus D$). Here, we need an unoccupied vertex u in order to move each pebble on the path π_{vu} towards it with plan α_{vu}. In this case u must be a regular vertex, so that we need to replace (2.7) with:

$$u \in \arg\min_{v' \in V_R \setminus S} d(v', v).$$

Gather holes problem with trans-shipment vertices

We use the same procedure described in Section 2.2.1. However, the choice of set M defined in (2.6) needs to be replaced by:

$$M \in \arg\min_{W \subset \mathscr{A}^s(H) \cap V_R:\ |W| = q} d(W, \bar{V}),$$

to guarantee that the holes in M are at regular vertices.

Motion planning problem with trans-shipment vertices

We must take into account the fact that trans-shipment vertices cannot host the marked pebble or the obstacles. Therefore, to ensure that the obstacle moves are feasible, we cannot only consider the cardinality of caterpillar sets, but the number of regular vertices they contain. To ensure this, we have to introduce the following modifications in the construction of the caterpillar sets:

1. replace d and c with $\tilde{d}$ and $\tilde{c}$;
2. the request on the size of the caterpillar sets concerns only the regular nodes: $|S_k \cap V_R| = \tilde{c}$ for all $k = 0, \dots, m-1$, and $|S_m \cap V_R| \leq \tilde{c}$;
3. parking positions ℓ_k cannot be trans-shipment vertices. At each step k, if the neighbors of i_k not belonging to π_{rt} are all trans-shipment vertices, then let ℓ_k be one of the 2-hop neighbors of i_k, which certainly exist in view of the first property of Definition 2.2.2 and are regular vertices because of assumption (2.10). Therefore, we can generalize the definition of *caterpillar sets* as follows:

$$S_k = \pi_{i_k j_k} \cup \pi_{i_k \ell_k} \cup \pi_{i_{k+1} \ell_{k+1}}, \quad \forall k = 0, \dots, m-1,$$

$$S_m = \pi_{i_m j_m} \cup \pi_{i_m \ell_m}.$$

For instance, see Figure 2.11.

To solve the motion planning problem, we use *Procedure A* and *Procedure B* with some small tweaks:

1. In *Procedure A*: when we slide the obstacles, we move them from $(S_{k+1} \setminus S_k) \cap V_R$ to $(S_k \setminus S_{k+1}) \cap V_R$. Indeed, we cannot bring holes from trans-shipment vertices.
2. In *Procedure B*: at each iteration we gather the holes that are on $V_j \cap V_R$ in H_j, which is a subset of $V_j \cap V_R$ of cardinality q_j closest to s:

$$H_j \in \arg \min_{\substack{W \in \mathscr{P}(V_j \cap V_R) \\ |W| = q_j}} d(W, \{s\}),$$

where $q_j = |\mathscr{A}^s(H) \cap V_j \cap V_R|$.

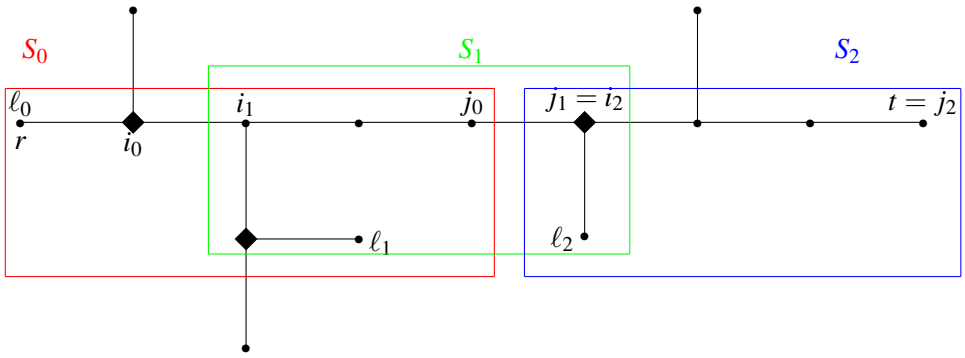


Figure 2.11: We consider the motion planning problem with source vertex r and target vertex t on a tree with $\tilde{c} = 5$. Diamond shapes represent trans-shipment vertices. S_0, S_1 and S_2 are the *caterpillar sets* along path π_{rt}.

2.3 MAPF on graphs

2.3.1 Solving MAPF on undirected graphs

In this section, we recall the planning method for a connected undirected graph presented in [54]. The main idea is to trasform the graph $G = (V, E)$ into a *biconnected component tree* $T := \mathscr{T}(G)$, and the MAPF problem into a *ts*-PMT problem. It is possible to prove that the MAPF problem is solvable on G if and only if the corresponding *ts*-PMT problem is solvable on T [53]. Moreover, the solution of MAPF can be obtained from the solution of the corresponding *ts*-PMT [54].

Convert MAPF into ts-PMT

Given a connected graph $G = (V, E)$, we construct the biconnected component tree $\mathscr{T}(G) = (V_T, E_T)$ as follows. We initialize $V_T = V$, $E_T = E$, and we convert each maximal non-trivial (i.e., with at least three vertices) biconnected component $S = (V_S, E_S) \subset G$ into a star subgraph. The nodes in V_S are the leaves of the star. The internal node of the star is a newly added trans-shipment vertex.

The conversion of S into a star involves the following steps:

1. add a trans-shipment vertex s,
2. remove every edge $e \in E_S$,
3. add the edges $\{(u,s) | u \in V_T\}$.

Note that $V_T = V \cup V^t$, where V^t is the set of all trans-shipment vertices and V represents the set of regular vertices of tree T. Note that trans-shipment vertices of the biconnected component tree has the following properties:

- $|V^t| = K$, where K is the number of non-trivial biconnected component of G;
- $\forall\, v \in V^t$ $\deg(v) \geq 3$;
- $\forall\, v, w \in V^t$ $d(v,w) > 1$, where $d(v,w)$ is the edge-distance between v and w.

Note that the latter property, which means that two trans-shipment vertices cannot be adjacent, is considered an assumption in the definition of ts-PMT (see Section 2.2.4).

Note that G and $\mathscr{T}(G)$ have a similar structure. Biconnected components of G correspond to star subgraphs in $\mathscr{T}(G)$, with trans-shipment vertices as internal nodes. Figure 2.12 shows an undirected graph and its corresponding biconnected component tree. Building $\mathscr{T}(G)$ from G takes a linear time with respect to $|E|$ [62].

G and $\mathscr{T}(G)$ have the same number of pebbles but a different number of holes. Denoting with H_T the set of holes of the tree, it holds that $H_T = H \cup H^t$, where H^t is a new set of holes added for trans-shipment vertices ($|H_T| = |V_T|$).

Let $\mathscr{A} : P \cup H \to V$ be a configuration on G. We associate it to a configuration on $\mathscr{T}(G)$, $\mathscr{A}_T : P \cup H_T \to V_T$ such that $(\forall q \in P \cup H)\, \mathscr{A}_T(q) = \mathscr{A}(q) \subset V$ and

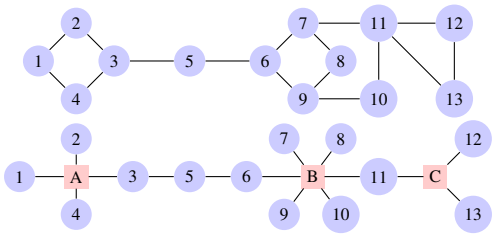


Figure 2.12: Undirected graph and corresponding biconnected component tree. A, B, and C are the trans-shipment vertices.

$\mathscr{A}_T(H_t) = V_T$. Note that $\tilde{\mathscr{A}_T}|_{P\cup H} = \tilde{\mathscr{A}}$, since pebbles are on the same vertices. In this way, we associate every MAPF instance $< G, \tilde{\mathscr{A}^s}, \tilde{\mathscr{A}^t} >$ to a ts-PMT instance $< \mathscr{T}(G), \tilde{\mathscr{A}^s_T}, \tilde{\mathscr{A}^t_T} >$. Reference [53] proves the following important result.

Lemma 2.3.1. *[53] Let $G = (V,E)$ be a connected undirected graph, which is not a cycle, and let $\mathscr{T}(G)$ be the corresponding biconnected component tree. Let $\mathscr{A}$ be an initial configuration on G and $\mathscr{A}_T$ the corresponding configuration on $\mathscr{T}(G)$. Let $a,b \in V$. Then, if $|H| \geq 2$, there is a plan f_{ab} such that $\mathscr{A}[a,b] = \rho(\mathscr{A}, f_{ab})$ if and only if there is a plan $f'_{a'b'}$ such that $\mathscr{A}_T[a',b'] = \rho(\mathscr{A}_T, f'_{a'b'})$.*

As a consequence of this Lemma, it follows that:

Theorem 2.3.1. *[53] Let G be a connected undirected graph which is not a cycle, with at least two holes. Then, MAPF on graph G is feasible if and only if ts-PMT on tree $\mathscr{T}(G)$ is feasible.*

Since feasibility of PMT on a tree $T = (V_T, E_T)$ is decidable in $O(|V_T|)$ time (see [5]), it follows that also MAPF on an undirected graph, which is not a cycle and with at least two holes, is decidable in linear time.

Actually, MAPF is decidable in linear time on undirected graphs in the general case, even on cycles and on graphs with only one hole.

Indeed, in a cycle, a MAPF instance is feasible if and only if in the final configuration the pebbles are in the same order as in the initial one.

In the case of graphs with only one hole, pebbles cannot move between different biconnected components (see [53]). Therefore, a MAPF problem on a graph with one

hole is feasible only if the final position of each pebble is in the same biconnected component as the initial position. Moreover, the feasibility on a biconnected graph with one hole is decidable in linear time (see Remark 8.8 of [53]).

In conclusion, for *any* undirected graph $G = (V_G, E_G)$ it holds that:

Theorem 2.3.2. *[53] The feasibility of a MAPF instance on G is decidable in $O(|V_G|)$ time.*

Summary of algorithm presented in [54] to solve MAPF on graphs

The algorithm presented in [54] to solve MAPF problem on a graph has the following structure:

1. Convert G into the biconnected component tree $\mathscr{T}(G)$ and convert MAPF into ts-PMT;

2. Solve ts-PMT problem on $\mathscr{T}(G)$;

3. Convert the solution of ts-PMT on $\mathscr{T}(G)$ into solutions of MAPF on G.

In particular, [54] presents a function *CONVERT-PATH* that converts the solutions of ts-PMT on $\mathscr{T}(G)$ into solutions of MAPF on G. When a pebble moves from v to u via a trans-shipment vertex s, this function first checks if there is a pebble-free path between v and u on G. If there is, this movement can be achieved. Otherwise, a more complex process is implied by the feasibility algorithm for graphs, presented more in detail in [52].

The algorithm that we will present in Section 2.3.5 to solve MAPF problem on strongly connected digraphs has the same structure as the one just described for undirect graphs. The only difference consists in the *CONVERT-PATH* function: while the one presented in [54] converts a plan on a tree into a plan on the undirect graph, we will study a function that provides a plan on the strongly connected digraph.

2.3.2 Strongly connected digraphs

As said, we consider MAPF for *strongly connected digraphs*. In this section we define this type of directed graph and we go into the detail of their structure. This will be useful to present an algorithm to solve MAPF.

Definition 2.3.1. *A digraph $D = (V, E)$ is* strongly connected *if for each $v, w \in V$, $v \neq w$, there exist a directed path from v to w, and a directed path from w to v in D.*

Given a digraph D, we indicate with $\mathscr{G}(D)$ its *underlying graph*, that is the undirected graph obtained by ignoring the orientations of the edges, and $\mathscr{T}(\mathscr{G}(D))$ the corresponding biconnected component tree. Note that D is strongly connected only if $\mathscr{G}(D)$ is connected (see Figures 2.12 and 2.13).

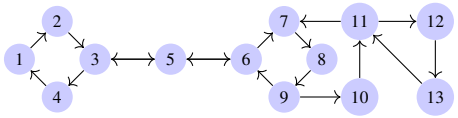


Figure 2.13: Example of strongly connected digraph: the corresponding underlying graph and biconnected component tree are shown in Figure 2.12.

An important property of strongly connected digraphs is that they can be decomposed in strongly biconnected components.

Definition 2.3.2. *A digraph D is said to be strongly biconnected if D is strongly connected and $\mathscr{G}(D)$ is biconnected.*

We recall that an undirected graph G is biconnected if it is connected and there are no cut vertices, i.e., the graph remains connected after removing any single vertex. The *partially-bidirectional cycle* is a simple example of a strongly biconnected digraph:

Definition 2.3.3. *A digraph is a partially-bidirectional cycle if it consists of a simple cycle C, plus zero or more edges of the type (u, v), where $(v, u) \in C$ (i.e., edges obtained by swapping the direction of an edge from C).*

Reference [56] shows that strongly biconnected (respectively, strongly connected) digraphs have an open (respectively, closed) ear decompositions.We recall the definitions of open and closed ear decompositions. Given a graph $D = (V_D, E_D)$ and a

sub-digraph $M = (V_M, E_M)$, a path π in D is a M-path if it is such that its startpoint and its endpoint are in V_M, no internal vertex is in V_M, and no edge of the path is in E_M. Moreover, a cycle C in D is a M-cycle if there is exactly one vertex of C in V_M.

Definition 2.3.4. *Let $D = (V_D, E_D)$ be a digraph and $L = [L_0, L_1, \ldots, L_r]$ an ordered sequence of sub-digraphs of D, where $L_i = (V_{L_i}, E_{L_i})$. We say that L is:*

1. *a closed ear decomposition, if:*
   - *L_0 is a cycle,*
   - *for all $0 < i \leq r$, L_i is a D_i-path or a D_i-cycle, where $D_i = (V_{D_i}, E_{D_i})$ with $V_{D_i} = \bigcup_{0 \leq j < i} V_{L_j}$ and $E_{D_i} = \bigcup_{0 \leq j < i} E_{L_j}$,*
   - *$V_D = \bigcup_{0 \leq j \leq r} V_{L_j}$, $E_D = \bigcup_{0 \leq j \leq r} E_{L_j}$*
2. *an open ear decomposition (oed), if it is a closed ear decomposition such that for all $0 < i \leq r$, L_i is a D_i-path, (i.e., it is not a D_i-cycle).*

In Definition 2.3.4, each L_i is called an *ear*. In particular, L_0 is the basic cycle and the other ears are derived ears. An ear is *trivial* if it has only one edge.

Definition 2.3.5. *We say that an open ear decomposition of a strongly biconnected digraph is regular (r-oed) if the basic cycle L_0 has three or more vertices, and there exists a non-trivial derived ear with both ends attached to the basic cycle.*

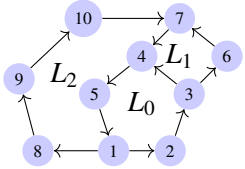


Figure 2.14: Digraph with an open ear decomposition.

Observation 2.3.1. *Let $D = (V, E)$ be a digraph with an oed $L = [L_0, L_1, \ldots, L_n]$. For each pair $v, w \in V$, there exists a sequence of cycles $C = [C_1, \ldots, C_n]$ such that:*

- *$v \in V_{C_1}$ and $w \in V_{C_n}$;*

- *for all* $j = 1, \ldots, n-1$, $\exists a_j, b_j \in V_{C_j} \cap V_{C_{j+1}}$ *such that* $(a_j, b_j) \in E$.

Figure 2.14 shows a digraph with an oed $[L_0, L_1, L_2]$. *The sequence of cycles associated to pair* $v = 2$, $w = 10$ *is* $C = [C_0, C_2]$, *where* $C_0 = L_0$ *and* C_2 *is the subgraph induced by* $\{1,8,9,10,7,4,5\}$. *Note that* $(4,5) \in C_0 \cap C_2$. *The sequence associate to pair* $v = 1$, $w = 6$ *is simply* $C = [C_1]$, *where* C_1 *is the subgraph induced by* $\{1,2,3,6,7,4,5\}$. *In fact, nodes* 1 *and* 6 *belong to the same cycle.*

Proof. Let $\pi = u_1 = v, u_2, \ldots, u_{n-1}, u_n = w$ be a shortest path from v to w. Let L_i be an ear such that $v, u_2 \in V_{L_i}$. Let n_1 and m_1 be the startpoint and endpoint of L_i. Then, there exists a path π_1 from m_1 to n_1 and $C_1 = \pi_1 \cup L_i$ is the first cycle of the sequence. We initialize $C = [C_1]$ and we set $k = 1$. Now, if $n > 2$, for $j = 3, \ldots, n$:

- if $u_j \in V_{C_k}$ we go to next iteration;
- otherwise, let π_{j-1} be a path from u_j to u_{j-2}, $C_{k+1} = (V_{C_{k+1}}, E_{C_{k+1}}) = \pi_{j-1} \cup (u_{j-2}, u_{j-1}) \cup (u_{j-1}, u_j)$ (note that $u_{j-2}, u_{j-1} \in V_{C_k} \cap V_{C_{k+1}}$); we add C_k to C and set $k = k+1$, then we go to the next iteration.

□

We recall the following results, that characterize strongly biconnected and strongly connected digraphs:

Theorem 2.3.3. *Let D be a non-trivial digraph.*

- *D is strongly biconnected if and only if D has an oed. Any cycle can be the starting point of an oed [63].*
- *D is strongly biconnected if and only if exactly one of the following holds [6]:*
  1. *D is a partially-bidirectional cycle;*
  2. *D has a r-oed.*

Theorem 2.3.4. *[64] Let D be a non-trivial digraph. D is strongly connected if and only if D has a closed ear decomposition.*

Observation 2.3.2. *Roughly speaking, this last result means that a strongly connected digraph is composed of non-trivial strongly biconnected components connected by corridors, or articulation points. A corridor on a digraph $D = (V,E)$ is a sequence of adjacent vertices $u_1, \dots, u_n$ such that*

- $(u_i, u_{i+1}), (u_{i+1}, u_i) \in E$ *for each* $i = 1, \dots, n-1$*;*
- *all the vertices of the sequence except u_1 and u_n (called endpoints) have exactly two neighbors.*

For example, in Fig. 2.13 the subgraph induced by nodes 3,5 and 6 is a corridor. We indicate with $\mathscr{C}(D)$ the set of all corridors of D. Vertex $v \in V$ is an articulation point if its removal increases the number of connected components of the underlying graph $\mathscr{G}(D)$. In Fig. 2.13 nodes 3, 6 and 11 are articulation points. We define the subclass of corridors $\bar{\mathscr{C}}(D) \subset \mathscr{C}(D)$ that have only articulation points as endpoints.

Note that each star subgraph of $\mathscr{T}(\mathscr{G}(D))$ represents a biconnected component of $\mathscr{G}(D)$, which corresponds to a strongly biconnected component of D. Indeed, Theorem 9 of [56] defines a one-to-one correspondence between strongly biconnected components of D and biconnected components of $\mathscr{G}(D)$.

Finally, we recall the following definition about strongly biconnected digraphs adapted from [53], that will be useful in Section 2.3.5:

Definition 2.3.6. *Let $B = (V,E)$ be a strongly biconnected digraph and $v \notin V$ be an external node. We consider a digraph $G = (V \cup \{v\}, \bar{E})$ with $E \subset \bar{E}$. We say that G is:*

- *a strongly biconnected digraph with an entry-attached edge, if there exists $z \in V$ such that $\bar{E} = \{(v,z)\} \cup E$;*
- *a strongly biconnected digraph with an attached edge, if there exists $z \in V$ such that $\bar{E} = \{(v,z),(z,v)\} \cup E$.*

2.3.3 Feasibility of MAPF on strongly connected digraphs

In this section, we focus on feasibility of MAPF problem on strongly connected digraphs.

As shown in Proposition 13 of [56], in strongly connected digraphs each move is reversible. From this, a more general result follows:

Proposition 2.3.1. *In a strongly connected digraph each plan has a reverse plan.*

Proposition 2.3.1 leads to the following result about the feasibility of MAPF on digraphs:

Theorem 2.3.5. *Let $D = (V_D, E_D)$ be a strongly connected digraph. Then,*

1. *any MAPF instance on D is feasible if and only if it is feasible on the underlying graph $G = \mathscr{G}(D)$;*
2. *feasibility of any MAPF instance on D is decidable in linear time with respect to $|V_D|$.*

Proof. 1. The necessity is obvious. To prove sufficiency, let f' be a plan which solves a MAPF instance on $\mathscr{G}(D)$. Then we can define a plan f on D in the following way. For each pebble move $u \to v$ in f', if $(u,v) \in E_D$, we perform move $u \to v$ on D. Otherwise, since $(v,u) \in E_D$, we execute a reverse plan for $v \to u$, $(v \to u)^{-1}$, that exists by Proposition 2.3.1.

2. It follows from Theorem 2.3.2.

□

A direct consequence of Theorem 2.3.5 and Theorem 2.3.1 is the following important result:

Corollary 2.3.1. *Let D be a strongly connected digraph with at least two holes and such that the corresponding underlying graph $\mathscr{G}(D)$ is not a cycle. Then, MAPF on D is feasible if and only if ts-PMT on tree $\mathscr{T}(\mathscr{G}(D))$ (i.e., the biconnected component tree of $\mathscr{G}(D)$) is feasible.*

The proof of Theorem 2.3.5 leverages the reversibility of each pebble motion in strongly connected digraphs. It presents a simple algorithm that reduces MAPF for strongly connected digraphs to the case of undirected graphs. However, this approach leads to redundant solutions, since it does not exploit the directed graph structure. This fact is illustrated in Fig. 2.15, that shows a digraph D and its associated underlying graph $\mathscr{G}(D)$.

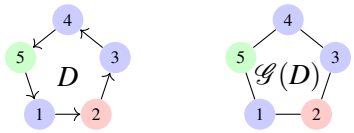


Figure 2.15: A digraph D and its underlying graph $\mathscr{G}(D)$.

Example 2.3.1. *A pebble p is placed at node 2, while all other nodes are free. We want to move p to 5. Plan $f' = (2 \to 1)(1 \to 5)$ is a solution of the corresponding problem on $\mathscr{G}(D)$. We convert this to a plan on D by applying the method in Theorem 2.3.5. Since $(2,1) \notin E_D$, move $(2 \to 1)$ is converted into plan $(2,3)(3,4)(4,5)(5,1)$. Similarly, move $(1 \to 5)$ is converted into $(1,2)(2,3)(3,4)(4,5)$. This solution is redundant, since the shorter plan $f = (2 \to 3)(3 \to 4)(4 \to 5)$ solves the overall problem.*

Definition 2.3.7. *We say that a MAPF solution algorithm has length complexity $O(n^k)$ if there exists a positive real constant $\gamma \in \mathbb{R}$ such that, for any MAPF instance on a graph with n nodes, the algorithm is able to provide a solution plan f of length $|f| \leq \gamma \cdot n^k$.*

Given a strongly directed graph $D = (V,E)$, and denoting by $\mathscr{C}(D)$ the set of all the cycles contained in D, we define

$$N = \max_{e \in E} \quad \min\{|C| : C \in \mathscr{C}(D),\ e \in E_C\}. \tag{2.13}$$

Proposition 2.3.2. *The length complexity of the algorithm described in Theorem 2.3.5 is $O(n^3 N^2)$.*

Proof. On undirected graphs, Kornhauser ([8]) proposes a MAPF algorithm that computes solutions of length γn^3, where $\gamma \in \mathbb{R}$ is a positive real constant. In the worst

case, each pebble move $u \to v$ on the undirected graph must be converted into a reverse plan (as in Example 3.2b). A reverse plan consists of a complete rotation of a cycle, which takes $O(n_c^2)$ moves, where n_c is the number of nodes of the cycle that contains arc $u \to v$. Each edge $e_i \in E$, crossed by the pebbles in the solution, belongs to a collection of cycles $\{C_i^j\}_j$ of the graph, among which we can choose the one with minimum length $n_i \leq N$, denoted with $C_i = \arg\min\{|C| : C \in \mathscr{C}(D),\ e_i \in E_C\}$. Let $[C_1, C_2, ..., C_m]$ be the sequence of cycles of minimum length, chosen for each edge, and $(n_1, n_2, ..., n_m)$ the corresponding number of nodes. Then, the overall solution length is bounded by

$$\gamma n^3 \cdot \sum_{i=1}^{m} O(n_i^2) \leq \gamma n^3 N^2.$$

□

2.3.4 Necessary and Sufficient condition for feasibility

Let us consider a strongly connected digraph $D = (V, E)$ and the corresponding biconnected component tree $T = (V_T, E_T)$ with $V_T = V \cup V^t$. In this section we want to derive a necessary ans sufficient condition (n.s.c.) for feasibility of MAPF on D from the n.s.c. of the ts-PMT on the corresponding tree T.

First of all, we remind that the n.s.c (2.12) for the feasibility of any ts-PMT provided in Section 2.2.4 is

$$|H| \geq |V^t| + \tilde{c}^T,$$

where $\tilde{c}_T := \max\{\tilde{c}_T^1, \tilde{c}_T^2 + 1\}$ with

$$\tilde{c}_T^1 := \max\{\tilde{d}_T(a,b) : \pi_{ab} \in \mathscr{C}(T)\},$$

$$\tilde{c}_T^2 := \max\{\tilde{d}_T(a,b) : \pi_{ab} \in \bar{\mathscr{C}}(T)\},$$

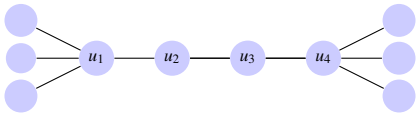
and $\tilde{d}_T(a,b)$ is the number of regular vertices belonging to the shortest path from a to b. In the same way we can define $\tilde{d}_D$, $\tilde{c}_D^1$, $\tilde{c}_D^2$ and $\tilde{c}_D$ on the digraph D.

Since establishing feasibility of MAPF on a digraph is equivalent to establishing the feasibility on the corresponding biconnected component tree (see Corollary 2.3.1),

we can easily deduce that (2.12) is the n.s.c. for solvability of MAPF on a strongly connected digraph which is not a partially-bidirectional cycle and which has at least two holes. Anyway, we want to translate (2.12) into a condition directly verifiable on the digraph.

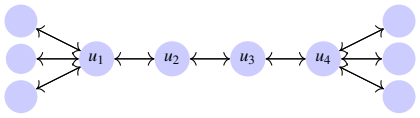
To do that, we have to find the relation between $\tilde{c}_T$ and $\tilde{c}_D$.

We consider a corridor $u_1, u_2 \dots, u_{n-1}, u_n$ on the tree.

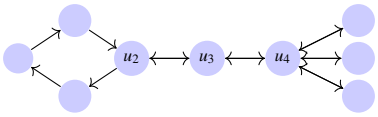


There are three possibilities:

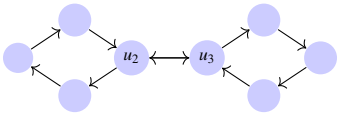
1. $u_1, u_n \notin V^t$: the corresponding corridor on D is exactly the same. Note that $\tilde{d}_T(u_1, u_n) = \tilde{d}_D(u_1, u_n)$.



2. $u_1 \in V^t, u_n \notin V^t$ (or vice versa): the corresponding corridor on D is $u_2 \dots, u_{n-1}, u_n$. Note that $\tilde{d}_T(u_1, u_n) = \tilde{d}_D(u_2, u_n)$.



3. $u_1, u_n \in V^t$: the corresponding corridor on D is $u_2, \dots, u_{n-1}$. Note that $\tilde{d}_T(u_1, u_n) = \tilde{d}_D(u_2, u_{n-1})$.



Since the vertex-length of the corridors on the digraph is equal to the vertex-length of the corresponding corridors on the tree, it follows that $\tilde{c}_D^1 = \tilde{c}_T^1$, $\tilde{c}_D^2 = \tilde{c}_T^2$ and $\tilde{c}_D = \tilde{c}_T$. Since the number of trans-shipment vertices is equal to the number of non trivial biconnected components (i.e., $|V^t| = K$) condition (2.12) is equivalent to

$$|H| \geq K + \tilde{c}_D. \tag{2.14}$$

2.3.5 A path planner for strongly connected digraph: diSC Algorithm

The first possible approach to solve MAPF on strongly connected digraphs is described in Theorem 2.3.5. This algorithm solves MAPF on the corresponding underlying graph and then converts the solution into a solution on the original digraph. However, as just widely discussed in Section 2.3.3, in some cases solutions found with this procedure may be too long.

To find shorter solutions in polynomial time, we present diSC algorithm, that better exploits the structure of the directed graph. In particular, we will exploit the fact that strongly connected digraphs can be decomposed in strongly biconnected components (see Observation 2.3.2).

In Section 2.3.3 we proved that MAPF on D is feasible if and only if ts-PMT on $\mathscr{T}(\mathscr{G}(D))$ is feasible. Therefore, as in the case of undirected graphs (see Section 2.3.1), we can reduce MAPF to ts-PMT, solve it, and then convert the solution on the tree into a solution on the original directed graph.

Thus, diSC strategy is the same presented in Section 2.3.1 to solve MAPF on undirected graphs. The main steps are the following ones:

1. Convert the digraph D into a tree $T = \mathscr{T}(\mathscr{G}(D))$ and consider the corresponding ts-PMT problem.

2. Solve the ts-PMT problem on tree T.

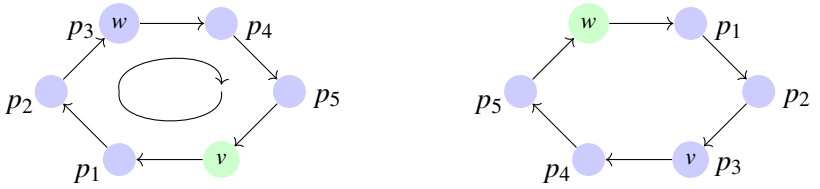
3. Convert the solution plan on T into a plan on D by a suitable algorithm *CONVERT-PATH*.

The novelty in diSC algorithm lies in the definition of algorithm *CONVERT-PATH*, which allows converting a plan f', that solves the ts-PMT, into a plan f, that solves the MAPF on the digraph. In Theorem 3.4.1, we will describe in detail algorithm *CONVERT-PATH*.

Basic plans for *CONVERT-PATH* algorithm

Algorithm *CONVERT-PATH* uses a number of basic plans, that perform simple tasks:

1. k-**CYCLE ROTATION**: this plan rotates all pebbles on a cycle, moving each one by k positions. In particular, if $k = 1$ each pebble moves forward by one position. If $k = n$ (where n is the cycle length) it peforms a *complete* rotation, so that each agent returns to its initial position. We denote this plan with r_k^C.

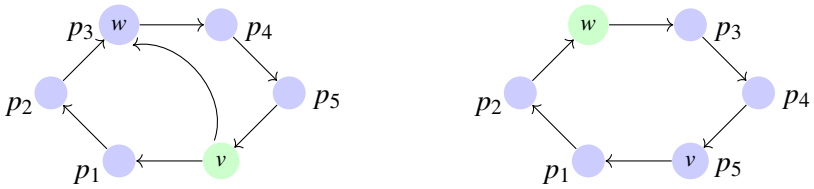


(a) Initial configuration. (b) Final configuration.

Figure 2.16: Example of 3-CYCLE ROTATION.

2. **BRING HOLE FROM** v **TO** w: holes are needed to execute any move or plan. For example, a k-**CYCLE ROTATION** is not possible unless there is at least one hole in the cycle. For this reason, it is useful to define this plan, which allows moving a hole from one position of the graph to another. Plan **BRING BACK HOLE FROM** w **TO** v is a reverse of BRING HOLE FROM v TO w. Hence, it returns the hole to its initial position, and moves back all agents to their initial positions. We denote these plans with $h_{v,w}$ and $h_{v,w}^{-1}$.

We formally define the plans just described in the Appendix.



(a) Initial configuration. (b) Final configuration.

Figure 2.17: Example of BRING HOLE FROM v TO w.

Observation 2.3.3. *Let $\mathscr{A}$ be an initial configuration, $h_{v,w}$ a plan* BRING HOLE FROM v TO w*, and $\bar{\mathscr{A}} = \rho(\mathscr{A}, h_{v,w})$ the corresponding final configuration. Given $a, b \in V$ with $b \neq v$, $p = \mathscr{A}^{-1}(a)$ and $q = \mathscr{A}^{-1}(b)$ are the pebbles or holes that occupy a, b. Then, $\bar{\mathscr{A}}[\bar{\mathscr{A}}(p), \bar{\mathscr{A}}(q)] = \overline{\mathscr{A}[a,b]}$. That is, the configuration obtained performing $h_{v,w}$ on $\mathscr{A}[a,b]$ is equal to the one obtained by exchanging $\bar{\mathscr{A}}(p)$ and $\bar{\mathscr{A}}(q)$ on $\bar{\mathscr{A}}$.*

Plans for movements through biconnected components

As said, *CONVERT-PATH* algorithm converts a PMT solution into a MAPF solution. The most complex subtask is the conversion of a movement in a 'star' into a sequence of movements within the associated biconnected component. In particular, each pebble motion to a different spike of the same star requires crossing the star transshipment vertex. We need to convert this motion into a plan on the digraph that allows moving the pebble to the same final position, without altering the positions of all other pebbles.

Observation 2.3.4. *In Observation 2.3.2 we noted that a strongly connected digraph $D = (V, E)$ is composed of non-trivial biconnected components and corridors. Following the classification presented in [53] for undirected graphs, in strongly connected directed graphs we defined three different motion tasks. Following [53], for each task we present one or more Lemmas, that describe a possible plan:*

1. *a pebble moves within the same strongly biconnected component: Stay in Lemma 2.3.5;*

2. *a pebble moves from a corridor to a strongly biconnected component (or vice-versa): Entry Lemma 2.3.2, Go out Lemma 2.3.3, Attached-Edge Lemma 2.3.4;*

3. *a pebble moves from a strongly biconnected component to another one, connected by an articulation point: Two Biconnected Components Lemma 2.3.6.*

All these Lemmas define plans that require two holes, with the following roles:

- the *destination* hole h_1 is in the final position of pebble p. In the final configuration, h_1 and p will be exchanged;
- the *transport* hole h_2 is used to move p to the position of h_1. Indeed, each pebble motion requires the presence of an hole. Each plan, at the end, brings back h_2 to its initial position, together with all pebbles with the exception of p.

Lemma 2.3.2. ***Entry.*** *Let P be a set of pebbles and H, with $|H| \geq 2$, a set of holes on $D = (V \cup \{v\}, \bar{E})$, where D is a strongly biconnected digraph with an entry-attached edge (v,y) (see Definition 2.3.6). Let $\mathscr{A}$ be a configuration, $p \in P$ such that $\mathscr{A}(p) = v$, and $w \in \mathscr{A}(H)$. Let $\mathscr{A}[v,w]$ be the configuration defined in (4.1). Then, there exists a plan f_{vw} such that $\mathscr{A}[v,w] = \rho(\mathscr{A}, f_{vw})$, i.e., that moves p from v to w, without altering the locations of the other pebbles.*

Figure 2.18 shows an example of this situation.

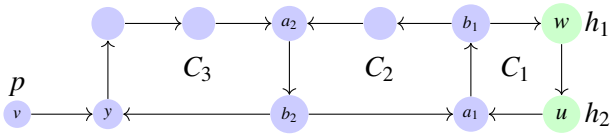
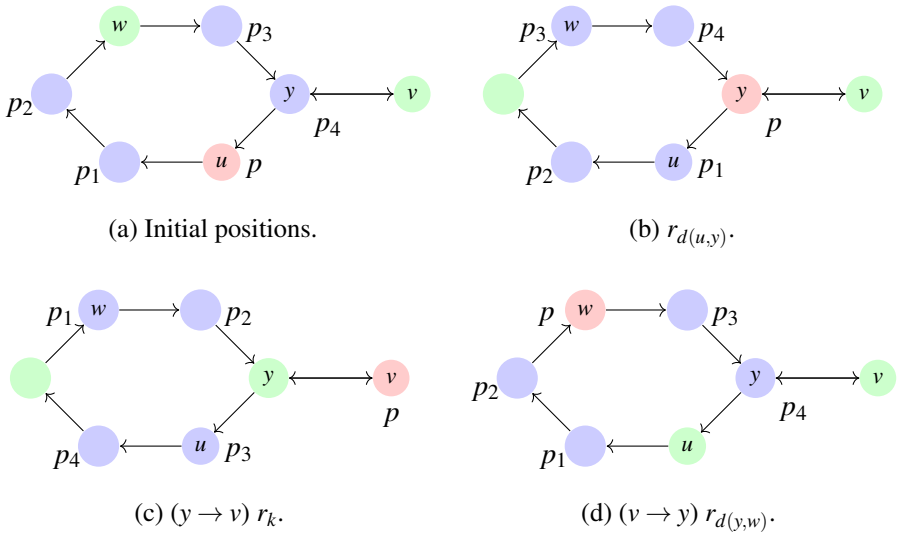


Figure 2.18: An example of situation of Entry Lemma 2.3.2. Pebble p has to move on w without altering the final location of the other pebbles. h_1 and h_2 are, respectively, the destination and transport holes. The solution plan is $f_{vw} = r_2^{C_1} r_3^{C_2} r_2^{C_3} (v \to y) r_3^{C_3} r_2^{C_2} r_2^{C_1}$.

Lemma 2.3.3. ***Go out.*** *Let P be a set of pebbles and H, with $|H| \geq 2$, a set of holes on $D = (V \cup \{w\}, \bar{E})$, where D is a strongly biconnected digraph with an attached edge (y,w) (see Definition 2.3.6). Let $\mathscr{A}$ be a configuration, $p \in P$ such that $\mathscr{A}(p) = v \in V$,*

and $w \in \mathscr{A}(H)$. Let $\mathscr{A}[v,w]$ be the configuration defined in (4.1). Then, there exists a plan f_{vw} such that $\mathscr{A}[v,w] = \rho(\mathscr{A}, f_{vw})$, i.e., that moves p from v to w, without altering the locations of the other pebbles.

Lemma 2.3.4. ***Attached-Edge.*** *Let P be a set of pebbles and H a set of holes on $D = (V \cup \{v\}, \bar{E})$, a strongly biconnected digraph with an attached edge such that $|H| \geq 2$. Let $\mathscr{A}$ be a configuration, $p \in P$ such that $\mathscr{A}(p) = u$, and $w \in \mathscr{A}(H)$. Let $\mathscr{A}[u,w]$ be a final configuration defined as in (4.1), then there exists a plan f_{uw} such that $\mathscr{A}[u,w] = \rho(\mathscr{A}, f_{uw})$. Figure 2.19 shows an example of this situation.*



(a) Initial positions.

(b) $r_{d(u,y)}$.

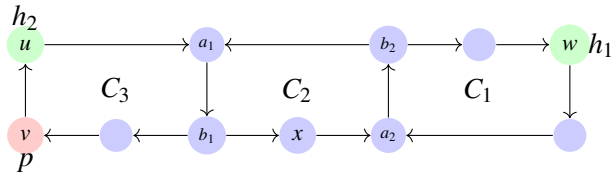
(c) $(y \to v)\ r_k$.

(d) $(v \to y)\ r_{d(y,w)}$.

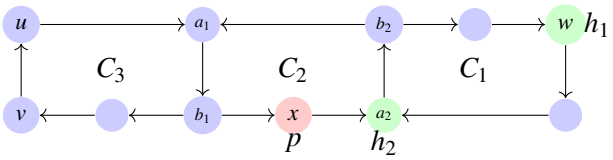
Figure 2.19: An example of situation of Attached-edge Lemma 2.3.4. Pebble p has to move on w without altering the final location of the other pebbles. The solution plan is $f_{vw} = r_{d(u,y)}\,(y \to v)\,r_k\,(v \to y)\,r_{d(y,w)}$.

Lemma 2.3.5. ***Stay in.*** *Let P be a set of pebbles and H, with $|H| \geq 2$, be a set of holes on $D = (V,E)$, a strongly biconnected digraph with a r-oed. Let $\mathscr{A}$ be a configuration, $p \in P$ such that $\mathscr{A}(p) = v$, and $w \in \mathscr{A}(H)$. Let $\mathscr{A}[v,w]$ be a configuration defined as in (4.1), then there exists a plan f_{vw} such that $\mathscr{A}[v,w] = \rho(\mathscr{A}, f_{vw})$.*

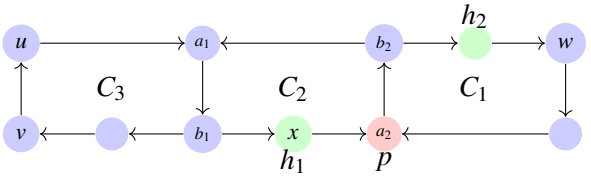
Figure 2.20 shows an example of this situation.



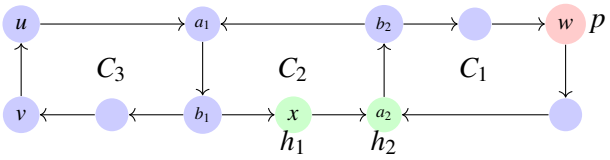
(a) Initial configuration.



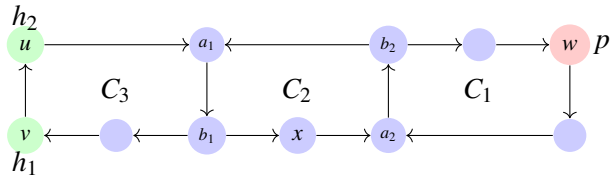
(b) $r_2^{C_3} r_2^{C_2}$



(c) $r_2^{C_1}(x \to a_2)$



(d) $r_3^{C_1}$



(e) $r_3^{C_2} r_3^{C_3}$

Figure 2.20: An example of situation of Stay in Lemma 2.3.5. Pebble p has to move on w without altering the final location of the other pebbles. h_1 and h_2 are, respectively, the destination and transport holes. The solution plan is $f_{vw} = r_2^{C_3} r_2^{C_2} r_2^{C_1}(x \to a_2) r_3^{C_1} r_3^{C_2} r_3^{C_3}$.

The next lemma deals with the case of two biconnected components joined by an articulation point (e.g., like $\{6,7,8,9,10,11\}$ and $\{11,12,13\}$ in Figure 2.13, where the articulation point is node 11).

Lemma 2.3.6. ***Two Biconnected Components.*** *Let P be a set of pebbles and H, with $|H| \geq 2$, a set of holes on $D = (V,E)$, a strongly connected digraph, composed of two biconnected components joined by an articulation point. Let $\mathscr{A}$ be a configuration, $p \in P$ be such that $\mathscr{A}(p) = a$, and $b \in \mathscr{A}(H)$. Let $\mathscr{A}[a,b]$ be a final configuration defined as in (4.1). Then, there exists a plan f_{ab} such that $\mathscr{A}[a,b] = \rho(\mathscr{A}, f_{ab})$.*

CONVERT-PATH algorithm

The following theorem shows that each plan on the biconnected components tree can be converted into a plan on the original digraph. The proof (which can be found in the Appendix) uses the Lemmas defined above to explicitly show how to convert a plan f' on the tree into a plan f on the digraph. This allows explicitly defining *CONVERT-PATH* algorithm.

Theorem 2.3.6. *Let P be a set of pebbles and H a set of holes on a strongly connected digraph $D = (V,E)$, which is not a partially-bidirectional cycle, and let $T = (V_T, E_T)$ be the corresponding biconnected component tree. Let $\mathscr{A}$ be an initial configuration on D and $\tilde{\mathscr{A}}$ the corresponding configuration on T. Let $a,b \in V$ and $p \in P$ be a pebble on a. Then, if $|H| \geq 2$, there is a plan f_{ab} on D such that $\mathscr{A}[a,b] = \rho(\mathscr{A}, f_{ab})$ if and only if there is a plan f'_{ab} on T such that $\mathscr{A}_T[a,b] = \rho(\mathscr{A}_T, f'_{a'b'})$.*

2.3.6 Upper bound for solutions length and time complexity

In this section, we provide an upper bound for the length of MAPF solutions found by diSC Algorithm. For a MAPF instance on a graph G composed by K non-trivial biconnected components, this length depends on:

A. the solution length of the ts-PMT problem on the associated biconnected components tree,

B. the number of trans-shipment vertex crossings in the ts-PMT solution,

C. the length complexity of the conversion of a movement through a trans-shipment vertex on the tree into a plan on the digraph,

D. the length complexity of plan BRING HOLE FROM v TO w and the number of times this plan is used.

We can bound these quantities as follows.

A. The best upper bound for PMT solutions is $O(|P|nc+n^2)$, provided by Theorem 2.2.1.

B. In the *Leaves Procedure* described in the previous Section, any vertex is crossed at most $O(|P|c)$ times by the pebbles in the overall solution (see Proposition 2.2.5).

C. To calculate the cost of the conversion, first we have to compute the length of a k-CYCLE ROTATION. Let C_i a cycle with n_i nodes and $k \leq n_i$. Then a 1-CYCLE ROTATION takes $O(n_i)$, while a k-CYCLE ROTATION takes $O(k \cdot n_i)$ which, in the worst case, is $O(n_i^2)$. If $C = [C_1, ..., C_m]$ is an ordered sequence of cycles with n nodes, and $k = (k_1, \ldots, k_m) \in \mathbb{N}^m$, the length complexity of the plan R_k^C, obtained by concatenating a k_1-CYCLE ROTATION over C_1, a k_2-CYCLE ROTATION over C_2, and analogous rotations over the remaining cycles of C, is given by

$$O\left(\sum_{i=1}^{m} k_i \cdot n_i\right)$$

Therefore, the complexity of a *complete* k-CYCLE ROTATION of C is $O(n^2)$. Note that also the plan BRING HOLE FROM v TO w has quadratic length, since it is a k-CYCLE ROTATION.

The plans described in *Stay in Lemma* 2.3.5, *Entry Lemma* 2.3.2, *Go out Lemma* 2.3.3, *Attached-Edge Lemma* 2.3.4 and *Two Biconnected Components Lemma* 2.3.6 are all combination of plans of the type k-CYCLE ROTATION and BRING HOLE FROM v TO w. Therefore, on a biconnected component with n_j nodes, they have $O(n_j^2)$ length.

D. To calculate the total cost of the function *CONVERT-PATH*, in addition to the cost of the plans presented in the lemmas, we must also consider how many times

we use function BRING HOLE FROM v TO w. Indeed, in the Proof of Theorem 3.4.1, when a pebble has to move within a biconnected component but there are not enough holes to perform this movement, we use BRING HOLE FROM v TO w to add a hole. In the worst case, BRING HOLE FROM v TO w is used once for each traversal of a trans-shipment vertex. Since the complexity of this plan is $O(n^2)$ and from point B. we know that each vertex is crossed $O(|P|c)$ times, we conclude that the overall complexity due to BRING HOLE FROM v TO w is $O(|P|n^2c)$.

If graph G is composed by K non-trivial biconnected components, the cost of the conversion performed by function *CONVERT-PATH* (see the proof of Theorem 3.4.1 in the Appendix) is

$$\underbrace{O(|P|n^2c)}_{D}+\sum_{i=1}^{K}\left(\underbrace{O(|P|c)}_{B}\cdot\underbrace{O(n_i^2)}_{C}\right)=$$

$$O(|P|n^2c)+O(|P|c)\cdot O(n^2+K)=O(|P|n^2c).$$

where n_i is the number of nodes of the i-th biconnected component, and $\sum_{i=1}^{K} n_i \leq n+K$, since some nodes can belong to more than one biconnected component.

Since the length complexity of PMT solution is $O(|P|nc+n^2)$ (see point A.), the overall complexity is dominated by the conversion cost and the following results holds:

Theorem 2.3.7. *The length complexity of diSC algorithm is $O(|P|n^2c)$.*

Note that with diSC Algorithm we find a better complexity result than the one obtained through the method of Theorem 2.3.5 (see Proposition 2.3.2).

Remark 2.3.1. *Note that the choice between the diSC algorithm and the procedure described in Theorem 2.3.5 depends on the structure of the digraph and on the number of pebbles. If cycles of the graph have small length, the latter procedure might be more convenient. For example, if $N=3$ the complexity is $O(9n^3)=O(n^3)$. Instead,*

if the corridors have a delimited length with respect to the dimensions of the graph (for example $c = 6$), the diSC algorithm is usually preferable. In any case, we note that in the worst case diSC has a complexity of $O(n^4)$, while the other procedure has complexity $O(n^5)$ in the worst case .

2.4 Experimental Results

We performed three distinct set of experiments, the first regarding only the motion planning algorithm on trees, the second for the whole PMT algorithm, and the third for the MAPF problem on strongly connected digraphs. The algorithms have been implemented in `Matlab`.

2.4.1 Motion planning

In the first set of experiments, we generated random trees with a number of nodes $|V|$ ranging from 20 to 200 by 20 using the *NetworkX* [65] graph generator for random trees (function `random_tree()`). The number of agents $|P|$ ranges from 2 to $|V|-2$, while $\mathscr{A}^s$ and $\mathscr{A}^t$ are randomly generated. Only instances that fulfill Assumption (2.2) are taken into account. For every combination of number of nodes and number of agents, we generated 100 instances, each instance refers to a different graph. In Figure 2.21 we display the average number of moves of the solutions found on nc.

According to Propositions 2.2.2 and 2.2.3, the motion planning algorithm returns solutions with length complexity $O(nc)$. In Figure 2.21 we can see a linear upper bound for the average number of moves, that supports the complexity result. We can also see how the number of moves is often much lower than the upper bound found. We remark that instances with number of moves closer to the upper bound line are those for which the number of pebbles is very high (as expected, these are more tricky instances).

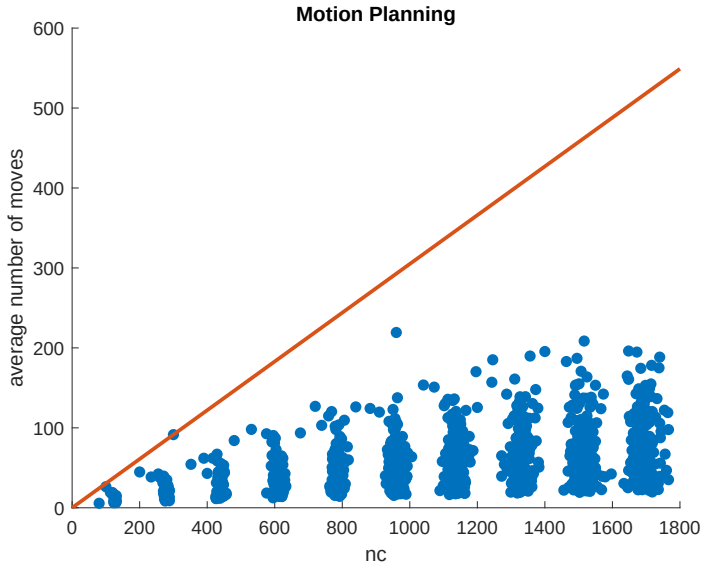


Figure 2.21: Average number of moves for the algorithm motion planning on *nc*.

2.4.2 PMT

In the second set of experiments, we generated random trees with a number of nodes $|V|$ ranging from 20 to 200 by 20 using the same procedure used for the first set of experiments. The number of agents $|P|$ ranges from 5 to $(3/4)|V|$ by 5, while $\mathscr{A}^s$ and $\mathscr{A}^t$ are randomly generated. As for the first set of experiments, only instances that fulfill Assumption (2.2) are taken into account. For every combination of number of nodes and number of agents, we generated 20 instances. In Figure 2.22 we display the average number of moves of the solutions found on $n|P|c+n^2$.

As stated in Theorem 2.2.1, the length complexity of the PMT algorithm is $O(n|P|c+n^2)$. Figure 2.22 displays a linear upper bound on the average number of moves, therefore confirming the complexity result.

The implementation of CATERPILLAR algorithm and the *Leaves Procedure* can be downloaded at `https://github.com/auroralab-unipr/PMT`.

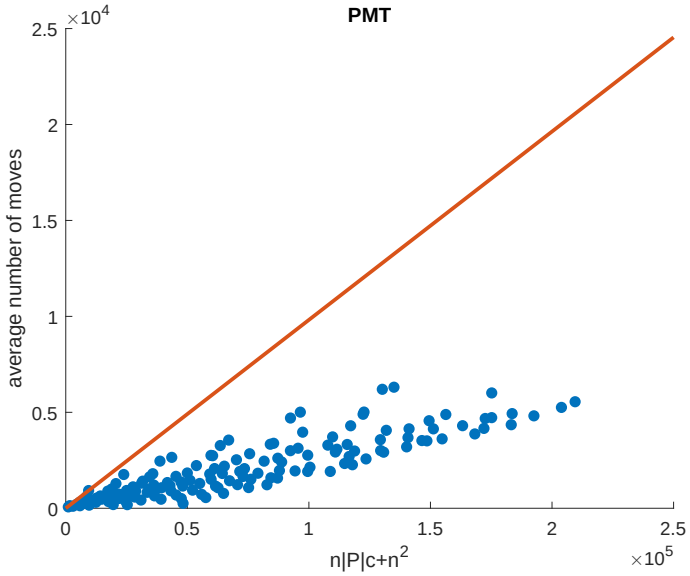


Figure 2.22: Average number of moves for the algorithm PMT on $n|P|c+n^2$.

2.4.3 MAPF

In the last set of experiments, we tested the diSC algorithm. To evaluate its behaviour, we generated random graphs with a number of nodes that ranges from 20 to 100, with increments of 5 nodes. For every number of nodes, we generated a set of 200 graphs. In order to generate test graphs with multiple biconnected components, we used the following procedure. First, we create a random connected undirected graph with function *networkx.connected_watts_strogatz_graph()*, contained in the Networkx library[1]. Then, we construct a maximum spanning rooted tree. We process the tree nodes with a breadth-first order. Every node that has a number n of children higher than 1 is converted into a biconnected component, together with its children, with the following method. We substitute the parent node and its children with a directed cycle with a random number of nodes lower or equal than $n+1$. Then, we add directed ears of random length (but sufficiently small, not to exceed the total number of $n+1$ nodes assigned to the biconnected component) and random initial and final nodes,

[1] https://networkx.org/

until the number of nodes in the resulting biconnected component equals $n+1$. After processing the tree, every remaining undirected edge $\{u,v\}$ is converted into two directed edges (u,v), (v,u).

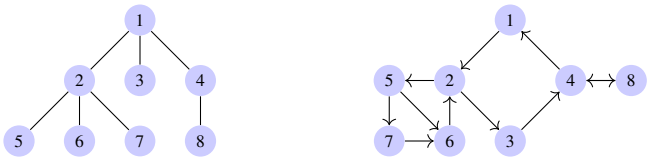


Figure 2.23: From tree to strongly connected digraph.

First, we ran the algorithm varying the number of nodes: for every generated graph (200 for every different number of nodes), we created a MAPF problem instance, with 10 agents and random initial and final positions. Then, we ran the algorithm on the set of 200 60-nodes random graphs, with a number of agents varying from 2 to 10. We used a *Intel(R) Core(TM) i7-4510U CPU @ 2.60 GHz* processor with 16 GB of RAM. For each obtained solution, we recorded the overall number of moves and the computation time.

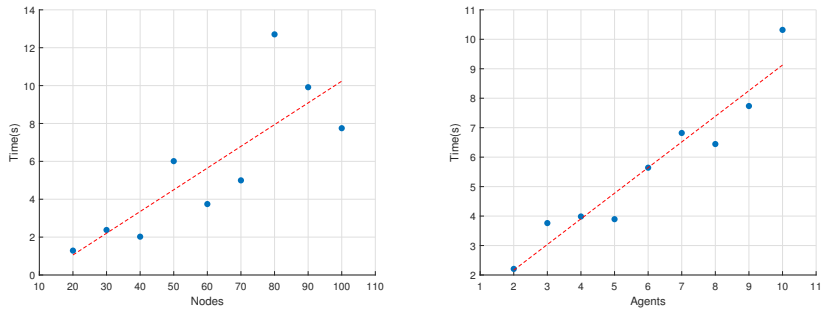


Figure 2.24: Median of running times per number of nodes and agents.

Fig. 2.24 shows the medians of the computational time as a function of the number of nodes and, respectively, the number of agents. Roughly, in both cases, the computational time increases linearly. In these figures, the trendlines are the least squares approximations with first order polynomials. Fig. 2.25 shows the medians of the overall number of moves as a function of the number of agents and, respectively,

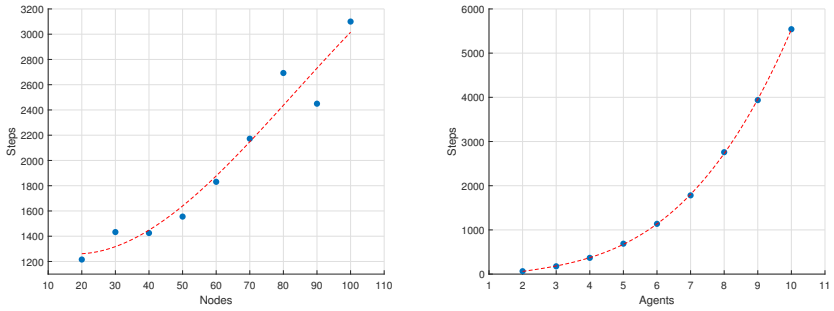


Figure 2.25: Median of moves per number of nodes and agents.

the number of nodes. Roughly, the overall number of nodes is a cubic function of the number of agents and a quadratic function of the number of nodes.

Then, we ran the algorithm for MAPF problem instances on a 397 nodes graph associated to the layout of a real warehouse (Fig. 2.27). We ran the algorithm varying the number of agents from 2 to 10. Also in this case both the number of steps and the running time seem to be increasing in a polynomial way.

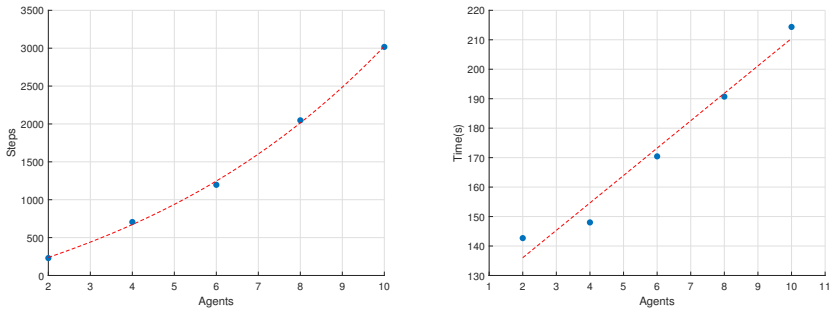


Figure 2.26: Median of moves and times per number of agents on warehouse.

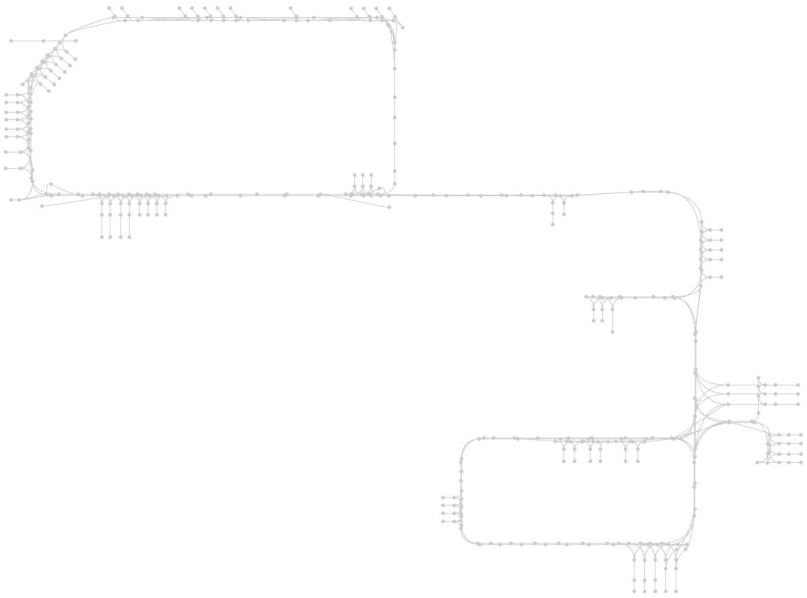


Figure 2.27: Graph representing the warehouse.

Chapter 3

Constrained Multi-Agent Path Finding on Directed Graphs

We focus on a variant of MP and MAPF, where we introduce additional constraints. Given directed graph $G = (V, E)$, we represent the problem constraints as a family of pairs

$$\mathscr{Q} = \{(S,k) : S \subset V,\, k \in \mathbb{N}\}, \tag{3.1}$$

where each pair (S,k) represents a constraint. Indeed, at any solution step, and for each pair $(S,k) \in \mathscr{Q}$, we require that the vertices in S contain at most k agents. For instance, consider the digraphs G and G' in Figures 3.2a-3.2b, where the constraints are the two pairs $(\{2,3\},1)$ and $(\{1,4\},1)$. At any time, we require that at most one agent is present in vertex subsets $\{2,3\}$ and $\{1,4\}$. For instance, this constraint need be satisfied if vertices $\{2,3\}$ and $\{1,4\}$ are too close to be occupied by two agents at the same time. In what follows, we denote by $\mathscr{C}$-MP and $\mathscr{C}$-MAPF the variants of MP and MAPF obtained by imposing constraints of this type. Namely, $\mathscr{C}$-MP (respectively, $\mathscr{C}$-MAPF) consists in finding a plan such that a single marked agent reaches a desired target vertex (respectively, all the agents reach their target vertices), avoiding collisions and *respecting the constraint that at each time step the number of agents on S is at most k, for each pair* $(S,k) \in \mathscr{Q}$. As in Chapter 2, when defining these problems we assume that the agents can move one at a time.

For example, let us consider the $\mathscr{C}$-MAPF instance on graph G' of Figure 3.2b, where we want to bring agent 1 from vertex 1 to vertex 3, and agent 2 from vertex 3 to vertex 1, respecting the constraints defined above. In Figure 3.1a we show a plan which solves the classical MAPF instance, but does not respect one of the constraints, since, at time step two, agents 1 and 2 are at vertices 1 and 4, violating constraint $(\{1,4\},1)$. Instead, Figure 3.1b presents a plan that solves the problem fulfilling the constraints.

	positions at each time step					
agent 1	1	1	2	2	2	3
agent 2	3	4	4	5	1	1

(a) Example of a plan that does not respect the constraints.

	positions at each time step				
agent 1	1	1	2	2	3
agent 2	3	5	5	1	1

(b) Example of a plan that respects the constraints.

Figure 3.1: Examples of plans on graph G' of Figure 3.2b .

Constraints defined by a suitable choice of pairs (S,k) allow:

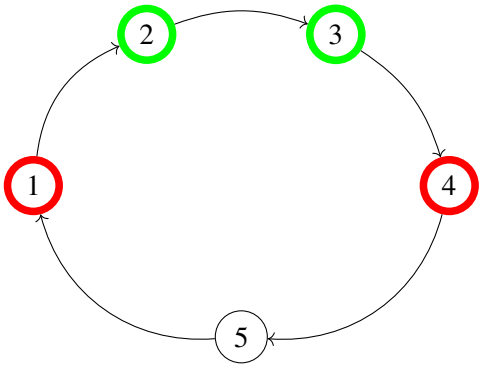
- taking into account the area occupied by each agent. Even if it is common to assume that each agent occupies a single vertex, this is often not true. For instance, industrial AGVs are often quite large, and close vertices cannot be occupied at the same time (see also the discussion of the previous example);
- maintaining a safety distance between agents;
- fulfilling some traffic rules within a warehouse, imposing a maximum number of agents in given areas. This can be due to constraints on floor load capacity, or to restrict the number of agents in areas shared with human workers.

We present three main new contributions.

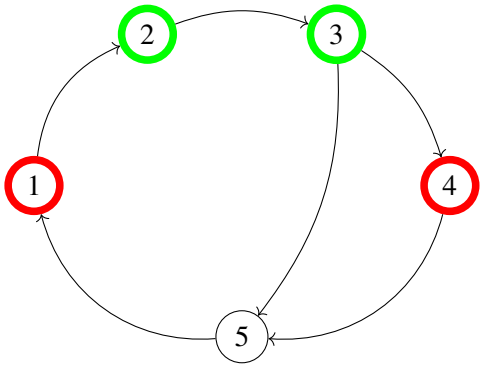
1) Complexity of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF

In Propositions 3.3.1 and 3.3.2 of Section 3.3 we prove the following result.

Finding a feasible solution for $\mathscr{C}$-MP and $\mathscr{C}$-MAPF is NP-hard.



(a) $G = (V,E)$, $\mathscr{Q} = \{(\{1,4\},1),(\{2,3\},1)\}$.



(b) $G' = (V,E')$, $\mathscr{Q} = \{(\{1,4\},1),(\{2,3\},1)\}$.

Figure 3.2: Examples of $\mathscr{C}$-MAPF instances.

We prove this by a reduction from 3-SAT. Note that, differently from $\mathscr{C}$-MP, deciding feasbility of (unconstrained) MP has polynomial time-complexity (see [57]). Instead, the NP-hardness of $\mathscr{C}$-MAPF is not surprising, since also (unconstrained) MAPF is NP-hard (see [45]).

2) Conversion of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF to unconstrained MP and MAPF

We propose to strengthen the constraints of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF in such a way

that the resulting problems are equivalent to (unconstrained) MP and MAPF over a suitably defined reduced graph. Such problems can be solved in polynomial time (Section 3.4).

We partition vertex set V in W and $V \setminus W$. Then, we disregard constraints in $\mathscr{Q}$, but we enforce the policy that at most one agent can stay in $V \setminus W$. Further, if agent a is in $V \setminus W$, no other agent is allowed to move until a re-enters in W. We choose W such that this policy guarantees that constraints in $\mathscr{Q}$ are satisfied. We define a *reduced graph* $G_W = (W, E_W)$. The edge set E_W is such that $(s,d) \in E_W$ if and only if an agent can move from s to d respecting the constraints in $\mathscr{Q}$, assuming that all vertices in $W \setminus \{s,d\}$ are occupied. We select a set W such that the resulting graph G_W is strongly connected. Then, instead of solving $\mathscr{C}$-MP or $\mathscr{C}$-MAPF on G, we solve the (unconstrained) MP or MAPF on G_W. As we will see, the obtained solution can be simply converted to a solution of the original $\mathscr{C}$-MP or $\mathscr{C}$-MAPF problem. Note that we require that G_W be strongly connected since, as already mentioned, feasibility of any MAPF instance over strongly connected digraphs can be decided in polynomial time (see Section 2.3.5, which focuses on the particular case of strongly connected digraphs). However, for $\mathscr{C}$-MP we could relax this requirement, since for MP feasibility can be decided in polynomial time under weaker assumptions (see, e.g., [57]). Our approach allows finding a feasible solution of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF in polynomial time. We do not address the problem of finding an optimal (or, at least, suboptimal) solution to these problems. However, in Chapter 4 we propose local search procedures for improving the quality of a known feasible solution of MAPF and MP. The proposed method could be also applied to $\mathscr{C}$-MAPF and $\mathscr{C}$-MP.

Figures 3.4a and 3.4b show possible reduced graphs G_{W_1} and G'_{W_2} for the examples introduced above. In particular, the problem presented in Figure 3.2a is reduced to the graph (W_1, E_{W_1}) of Figure 3.4a. A move from vertex 2 to vertex 4 over the reduced graph corresponds to path $2-3-4$ in the original graph, while a move from vertex 4 to vertex 2 corresponds to path $4-5-1-2$ in the original graph. Unfortunately, since the reduced graph has just two vertices, it does not allow solving $\mathscr{C}$-MP and $\mathscr{C}$-MAPF instances with two agents. Indeed, the agents block each other. This fact suggests a limitation of the proposed approach that will be further commented below.

The problem in Figure 3.2b corresponds to the reduced graph (W_2, E_{W_2}) of Figure 3.4b. This reduced graphs allows finding a feasible solution of the $\mathscr{C}$-MAPF instance introduced above, where we want to bring agent 1 from vertex 1 to vertex 3, and agent 2 from vertex 3 to vertex 1. Namely, instead of the initial $\mathscr{C}$-MAPF problem, we consider the same (unconstrained) MAPF on G'_{W_2}. Figure 3.3 shows a feasible solution. Then, we easily convert this solution into the solution of the $\mathscr{C}$-MAPF instance displayed in Figure 3.1b.

	positions at each time step			
agent 1	1	1	3	3
agent 2	3	5	5	1

Figure 3.3: Solution of MAPF problem on G'_{W_2}.

The proposed strengthening method has one significant advantage and one important limitation. The advantage is the reduction of complexity of the resulting planning problem. Namely, while the original $\mathscr{C}$-MP and $\mathscr{C}$-MAPF are NP-hard, the strengthened problems are simple MP and MAPF on a strongly connected digraph, and, thus, feasible solutions can be found with polynomial complexity with respect to the number of vertices and agents ([57]). The disadvantage is that this strengthening process is not exact. Namely, there exists feasible $\mathscr{C}$-MP and $\mathscr{C}$-MAPF instances that correspond to unfeasible MP or MAPF problems in the reduced graph. This fact is apparent if we get back to the digraph in Figure 3.2a with the related constraints. We consider a $\mathscr{C}$-MP instance with two agents: agent 1 should move from vertex 2 to vertex 4, while agent 2 is at vertex 4. As already commented, its strengthening to the MP over G_{W_1} (see Figure 3.4a), has no feasible solution because the two agents block each other. On the other hand, it is obvious that the original $\mathscr{C}$-MP instance admits the following feasible solution: agent 2 moves from vertex 4 to vertex 5; agent 1 first moves from vertex 2 to vertex 3, and then from vertex 3 to vertex 4.

3) Maximisation of the cardinality of W

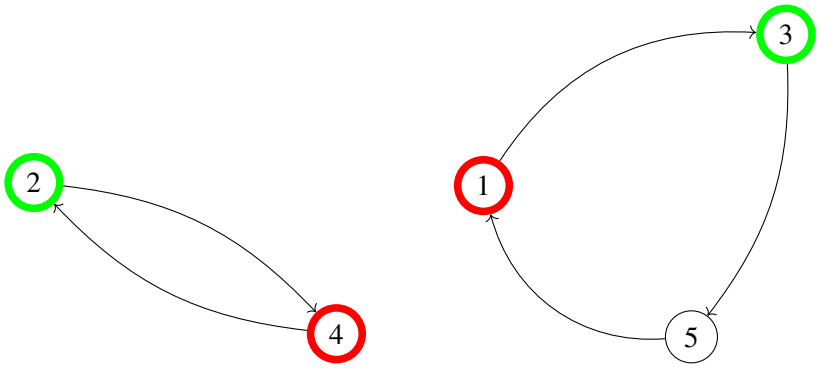
We introduce the $\mathscr{C}$-MIS problem, where we search for a reduced graph with vertex set W of maximum cardinality. We prove that this problem is strongly NP-hard

and propose a heuristic approach for its solution (Section 3.4.1).

It is convenient to choose a set W of largest possible cardinality. Indeed, the larger is the cardinality of W, the larger is the number of agents for which we can solve MAPF on the reduced graph. This is due to the fact that the feasibility of MAPF is related to the number of agents and of unoccupied vertices. In the case of trees, undirected graphs and strongly connected digraphs, the number of unoccupied vertices must be higher than a certain threshold which depends on the structure of the graph [8, 52]. In the case of strongly biconnected digraphs, two unoccupied vertices are always enough to ensure that a solution exists [6]. In Theorem 3.4.2, we show that the problem of finding W of maximum cardinality (called $\mathscr{C}$-MIS problem) is strongly NP-hard. For this reason, we propose two heuristic algorithms that allow finding a suboptimal set W in polynomial time.

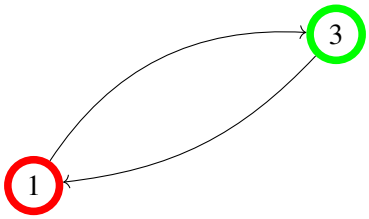
For instance, let us consider G'_{W_2} (Fig. 3.4b) and G'_{W_3} (Fig. 3.4c), which are both reduced graphs of G'. In particular, G'_{W_3} is not maximal. Between the two, it is convenient to choose G'_{W_2}, since it has a higher cardinality. This is evident with the following example. Let us consider the usual $\mathscr{C}$-MAPF instance over graph G' where agent 1 should go from vertex 1 to vertex 3, and agent 2 from vertex 3 to vertex 1. As already seen, the solution displayed in Figure 3.3 is feasible for the corresponding MAPF on G'_{W_2}, and can be converted into the solution displayed in Figure 3.1b over the original graph G'. Conversely, there does not exist a feasible solution on G'_{W_3}, since there are only two vertices and the two agents are blocked.

Comparison with existing literature. To our knowledge, $\mathscr{C}$-MP and $\mathscr{C}$-MAPF, in the formulation discussed here, have not been considered in literature. In particular, we could not find any reference that addresses our main topic: the conversion, through a strengtheing of the constraints, of constrained MP and MAPF problems into unconstrained ones. Anyway, some papers do consider MP and MAPF with additional constraints. For instance, [66] studies Multi-Agent Path Finding for Large Agents (LA-MAPF), which takes into account the occupancy of the agents, enforcing a safety distance to avoid collisions. Differently from this work, [66] does not propose



(a) Reduced graph G_{W_1} for G (Figure 3.2a), where $W_1 = \{2,4\}$.

(b) Reduced graph G'_{W_2} for G' (Figure 3.2b), where $W_2 = \{1,3,5\}$.



(c) Reduced graph G'_{W_3} for G' (Figure 3.2b), where $W_3 = \{1,3\}$.

Figure 3.4: Reduced graphs associated with the examples of Figure 1.

a conversion of this problem into a classical MAPF, but solves it directly, using an optimal algorithm, the Multi-Constraints CBS, a generalization of the Conflict-Based Search algorithm (see [49]). Also, [67] introduces additional constraints to MAPF. Namely, it tightens standard collision avoidance constraints, to avoid collisions even in case each agent experiences delays, or in the presence of localization errors. Finally, a somehow related problem is the path avoiding forbidden pairs (PAFP) problem, a generalization of the classical shortest path problem (see, e.g., [68]): given a set F of vertex pairs, this problem consists in finding a path for a single agent from a source to a target that contains at most one vertex from each pair in F.

3.1 Background

We recall the definitions of an *abstract simplicial complex* and a *matroid*.

Definition 3.1.1. *Given a set V, an abstract simplicial complex (ASC in the following) $\mathscr{F}$ is a family of subsets of V such that the following axioms hold:*

1. *trivial axiom: $\emptyset \in \mathscr{F}$;*
2. *hereditarian axiom: if $X \in \mathscr{F}$ and $Y \subset X$ then $Y \in \mathscr{F}$.*

Definition 3.1.2. *Given a set V, a matroid $\mathscr{F}$ is an abstract simplicial complex such that the following axiom holds:*

(3) exchange axiom: if $X, Y \in \mathscr{F}$ with $|X| = |Y| + 1$ then $\exists i \in X \setminus Y$ such that $Y \cup \{i\} \in \mathscr{F}$.

Roughly speaking, an ASC is a family of subsets of V, that contains all subsets of each family element. A matroid is an ASC with the additional property that every member of the family of non-maximal cardinality is strictly included in another member.

3.2 Problem Definition

We first introduce the following definition.

Definition 3.2.1. *An admissible collection $\mathscr{C}$ is a family of subsets of V. For each $U \in \mathscr{C}$ we say that U is admissible.*

We say that a configuration $\mathscr{A} : P \to V$ is admissible if $\mathscr{A}(P) \in \mathscr{C}$. Moreover, for a given initial valid configuration $\mathscr{A}^s : P \to V$ and a plan $f \in E^*$, we say that the pair $(f, \mathscr{A}^s)$ is **consistent with** an admissible collection $\mathscr{C}$ if every pebble configuration visited along the plan belongs to $\mathscr{C}$, that is

$$(\forall r \in \mathrm{Pref}(f)) \quad \rho(\mathscr{A}^s, r) \in \mathscr{C}.$$

This definition implicitly assumes that $\rho(\mathscr{A}^s, r)$ is well-defined for every prefix r of f. That is, all moves of f move a pebble to an empty vertex. Now, we are ready to introduce the definitions of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF. They are simple extensions of Definitions 2.1.2 and 2.1.1.

Definition 3.2.2. *($\mathscr{C}$-MP). Let $G = (V,E)$ be a digraph, P a set of pebbles, and $\mathscr{C}$ an admissible collection. Let $p \in P$, $s,t \in V$, $\mathscr{O} \subset V$. Let $\mathscr{A}^s$ be an initial configuration such that $\mathscr{A}^s(p) = s$ and $\mathscr{A}^s(P \setminus \{p\}) = \mathscr{O}$. The $\mathscr{C}$-MP defined by $< G, (s,t), \mathscr{O}, \mathscr{C} >$ consists in finding a plan f such that*

1. $t = \rho(\mathscr{A}^s, f)(p)$;
2. $(f, \mathscr{A}^s)$ *is consistent with* $\mathscr{C}$.

In Definition 3.2.2, s and t are the source and target vertices for pebble p, respectively. Set $\mathscr{O}$ contains the initial positions of all other pebbles.

Definition 3.2.3. *($\mathscr{C}$-MAPF problem). Let $G = (V,E)$ be a digraph, P a pebble set, and $\mathscr{C}$ an admissible collection. Given an initial valid configuration $\mathscr{A}^s$, and a final valid configuration $\mathscr{A}^t$, the $\mathscr{C}$-MAPF problem defined by $< G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} >$ consists in finding a plan f such that*

1. $\mathscr{A}^t = \rho(\mathscr{A}^s, f)$;
2. $(f, \mathscr{A}^s)$ *is consistent with* $\mathscr{C}$.

According to Definition 3.1.1, an admissible collection $\mathscr{C}$ is an ASC if every subset of an admissible configuration is admissible. In other words, any pebble configuration obtained by removing one or more pebbles from an admissible configuration is still admissible. Obviously, not all admissible collections are ASC. For instance, the admissible collection defined by requiring that one subset of V contains *at most* one pebble is an ASC. Instead, the admissible collection obtained by requiring that this subset contains *exactly* one pebble is not. Anyway, the family of admissible collections that are ASC is sufficiently broad to include many cases encountered in applications.

For instance, to define $\mathscr{C}$, we may choose a family $\mathscr{Q}$ of constraints (S,k) defined as in (3.1). For each pair (S,k), we require that, at any time, no more than k agents occupy the vertices in S. In this way, we define the following admissible collection $\mathscr{C}_{\mathscr{Q}}$, associated to $\mathscr{Q}$.

$$\mathscr{C}_{\mathscr{Q}} = \{U \subset V : (\forall (S,k) \in \mathscr{Q}) \quad |S \cap U| \leq k\}. \tag{3.2}$$

Note that (3.2) corresponds to the definition of constraint given in the Introduction, which is also the most natural one for the applications with AGVs. It turns out that form (3.2) is not restrictive, that is, an admissible collection is an ASC if and only if it can be written in this form.

Proposition 3.2.1. *An admissible collection $\mathscr{C}$ is an ASC if and only if there exists a family $\mathscr{Q}$ of constraints (S,k) such that $\mathscr{C} = \mathscr{C}_{\mathscr{Q}}$.*

Proof. ($\Leftarrow$) First note that $\emptyset \in \mathscr{C}_{\mathscr{Q}}$, so that 1), of Definition 3.1.1, holds. To show that also 2) holds, let $A \in \mathscr{C}_{\mathscr{Q}}$ and $B \subset A$. Then, $(\forall (S,k) \in \mathscr{Q})\, |A \cap S| \leq k$, and, since $B \subset A$, also $(\forall (S,k) \in \mathscr{Q})\, |B \cap S| \leq k$, which implies that $B \in \mathscr{C}_{\mathscr{Q}}$.

($\Rightarrow$) Let $\mathscr{C}$ be an ASC. For each $A \in \mathscr{P}(V)$, set $v_A = \max\{|B|, B \subset A, B \in \mathscr{C}\}$ and define the family $\mathscr{Q}$ composed of pairs (A, v_A), for all $A \in \mathscr{P}(V)$. For every $A \in \mathscr{P}(V)$, we have that $(\forall C \in \mathscr{C})\, |C \cap A| \leq v_A$. This implies that $\mathscr{C} \subseteq \mathscr{C}_{\mathscr{Q}}$. To prove that also $\mathscr{C}_{\mathscr{Q}} \subseteq \mathscr{C}$, by contradiction, assume that there exists $M \in \mathscr{C}$ such that $M \notin \mathscr{C}_{\mathscr{Q}}$. Then, there exists $N \in \mathscr{P}(V)$ such that $|M \cap N| > v_N$. However, since $\mathscr{C}$ is an ASC and $M \in \mathscr{C}$, necessarily $M \cap N \in \mathscr{C}$, which implies that $v_N \geq |M \cap N|$. □

Throughout this Chapter, we make the following assumption.

Assumption 3.2.1. *The admissible collection $\mathscr{C}$ is an ASC or, equivalently, $\mathscr{C} = \mathscr{C}_{\mathscr{Q}}$ for some family $\mathscr{Q}$ of constraints (S,k).*

As a consequence of Proposition 3.2.1, we will usually define the admissible collection $\mathscr{C}$ by choosing a family $\mathscr{Q}$ of constraints (S,k), with the understanding that the actual admissible collection is $\mathscr{C}_{\mathscr{Q}}$.

Remark 3.2.1. *An equivalent alternative to the definition of a admissible collection through a family $\mathscr{Q}$ of constraints (S,k) is its definition through a family $\mathscr{Q}'$ of triples (S,k,w^S), where w^S is a vector of positive integer weights. In this case the admissible collection is defined as follows*

$$\mathscr{C}_{\mathscr{Q}'} = \{U \subset V : (\forall (S,k,w^S) \in \mathscr{Q}') \quad \sum_{i \in S \cap U} w_i^S \leq k\}. \tag{3.3}$$

Basically, each constraint can be viewed as a knapsack constraint*, where w^S is the weight vector of the objects in S and k is the weight capacity of the knapsack. Definitions (3.2) and (3.3) are equivalent. It is obviously true that each admissible collection defined as in (3.2) can be represented as an admissible collection defined as in (3.3) (just set $w_i^S = 1$ for all $i \in S$ and for all S). But also the opposite is true. Indeed, each knapsack constraint $\sum_{i \in S \cap U} w_i^S \leq k$ can be replaced by the collection of all its cover inequalities (see, e.g., [**?**]), i.e.,*

$$\forall C \subseteq S : \sum_{i \in C} w_i^S > k \;\; \rightarrow \;\; |C \cap U| \leq |C| - 1,$$

i.e., each triple (S,k,w^S) can be replaced by the collection of pairs $(C, |C|-1)$ for all $C \subseteq S$ such that $\sum_{i \in C} w_i^S > k$. Note that the collection of cover inequalities associated to a knapsack constraint may contain an exponential number of elements. Thus, while equivalent, representation (3.3) is usually more compact than representation (3.2).

3.3 Complexity of the motion planning problem with additional constraints

In this section we will prove that, differently from MP, even establishing the feasibility of $\mathscr{C}$-MP is NP-hard.

The proof of NP-hardness is by a reduction from 3-SAT. In particular, it uses a construction similar to the one used in [68] to prove NP-hardness of the PAFP problem, even if the two problems are different. As an example, we consider the 3-SAT problem with variables $V = \{x_1, x_2, x_3\}$ and conjunctive normal form $(x_1 \vee \bar{x}_2) \wedge x_3 \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$. This problem can be reduced to a $\mathscr{C}$-MP instance $< G, (s,t), \mathscr{O}, \mathscr{C} >$, where G is

the digraph displayed in Figure 3.5, $(s,t) = (0,18)$, $\mathscr{O} = \{9,12,15\}$, and $\mathscr{C}$ is defined as in (3.2) where, for each pair $(S,k) \in \mathscr{Q}$, $k = 1$ and S belongs to the collection:

$$\mathscr{S} = \{\{11,5\},\{10,2\},\{3,14\},\{6,13\},\{4,16\},$$
$$\{7,17\},\{1,9\},\{1,12\},\{1,15\}\}.$$

Indeed, solving $< G,(0,18),\{9,12,15\},\mathscr{C} >$ means moving the obstacles so that there is a feasible path for a pebble p from vertex 0 to vertex 18. Since p must necessarily pass through vertex 1, the three obstacles must leave from their initial positions (since $\{1,9\},\{1,12\},\{1,15\} \in \mathscr{S}$). Therefore, they must move so that at least one between $\{2,3\}$ ($\{x_1,\bar{x}_2\}$), at least one between $\{5,6,7\}$ ($\{\bar{x}_1,x_2,\bar{x}_3\}$), and 4 ($\{x_3\}$) remains free, which is equivalent to solve the 3-SAT associated to form $(x_1 \vee \bar{x}_2) \wedge x_3 \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$.

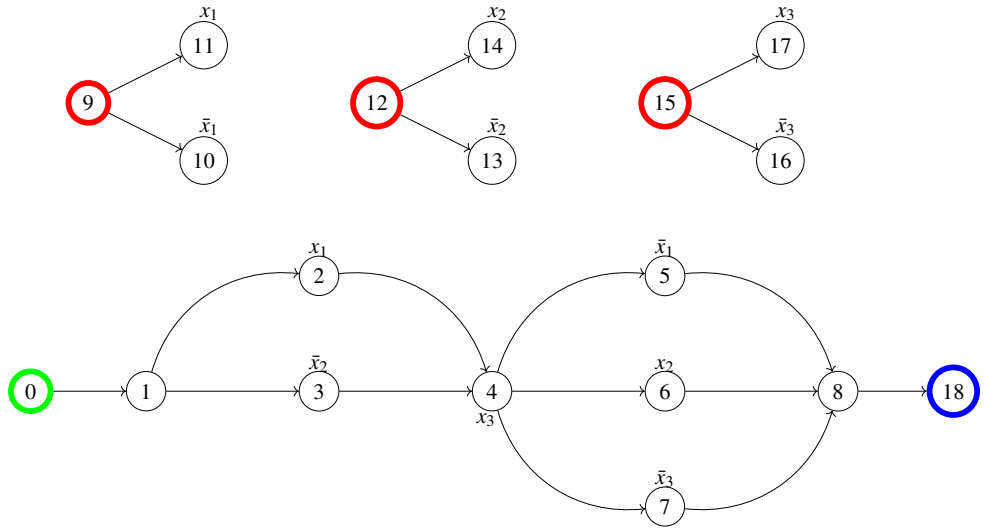


Figure 3.5: Reduction of the 3-SAT instance $(x_1 \vee \bar{x}_2) \wedge x_3 \wedge (\bar{x}_1 \vee x_2 \vee \bar{x}_3)$ to an $\mathscr{C}$-MP instance.

The above procedure can be extended to reduce any 3-SAT instance to a $\mathscr{C}$-MP instance on a digraph, so that we can prove the following.

Proposition 3.3.1. *$\mathscr{C}$-MP is NP-hard.*

Proof. Let us consider a 3-SAT instance, consisting of variables $x_1,\dots,x_n$ and k clauses, with at most 3 literals each. Formally, the j-th clause is $C_j = c_j^1 \vee c_j^2 \vee c_j^3$, $j \in \{1,\dots,k\}$, with $c_j^m = x_i$ or $c_j^m = \bar{x}_i$, $m \in \{1,2,3\}$, for some $i \in \{1,\dots,n\}$. We define a $\mathscr{C}$-MP instance on a digraph G as follows. Let $\mathscr{O} = \{o_1, o_2, \dots, o_n\}$ be the set of vertices that correspond to initial obstacles positions. For $i = 1,\dots,n$, we define a weakly-connected component of graph G, which connects vertex o_i to two vertices, representing literals x_i and $\bar{x}_i$ (see Figure 3.6). Moreover, we add to G a weakly-connected component, which represents the sequence of k clauses. This component contains vertices $s,t,d_1,\dots,d_k,d_{k+1}$ and c_j^m, $j = 1,\dots,k$, $m = 1,2,3$ (see Figure 3.7). Note that c_j^m represents both a literal and some vertex of the graph.

Vertices s and t are the source and the target of the marked pebble, respectively. Vertex c_k^i is associated to the literal containing variable x_i or $\bar{x}_i$ in the k-th clause. Vertices $d_1,\dots,d_k,d_{k+1}$ connect the subcomponents $C_1,\dots,C_k$, associated to each clause. We define the admissible collection $\mathscr{C}$ in form (3.2), where, for each pair

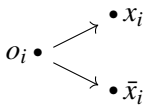


Figure 3.6: Component $\{o_i, x_i, \bar{x}_i\}$.

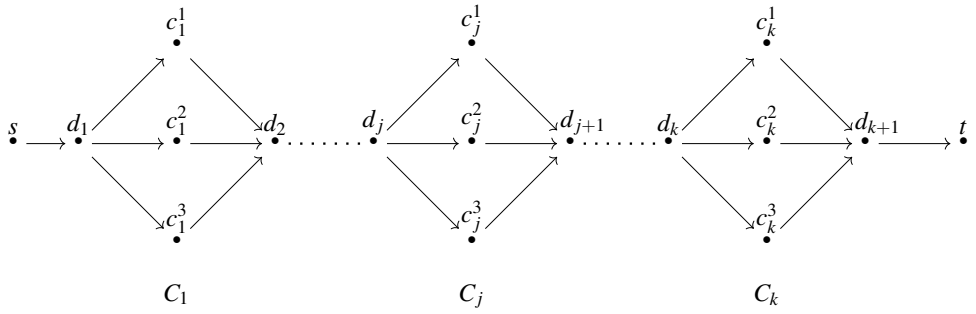


Figure 3.7: Component associated to the k clauses.

$(S,k) \in \mathscr{Q}$, $k = 1$ and S is the union of two families of sets. The first family is:

$$\{\{d_1, o_i\}, i = 1,\dots,n\}.$$

Since the pebble must pass through d_1 to reach target t, this admissible collection forces the obstacles to move away from their initial positions and to choose one of the two vertices x_i or $\bar{x}_i$. The second family is:

$$\left\{ \begin{array}{ll} \{c_j^m, x_i\} & \text{if } c_j^m = x_i, \\ \{c_j^m, \bar{x}_i\} & \text{if } c_j^m = \bar{x}_i, \end{array} \quad j = 1, \ldots, k,\ m = 1,2,3 \right\}.$$

In this way, an obstacle placed at vertex x_i disables all vertices associated to literal x_i, that is, belonging to set $\{c_j^m = x_i, j = 1, \ldots, k, m = 1,2,3\}$. The marked pebble is able to reach vertex t if and only if the obstacles are placed in such a way that each subcomponent C_j contains at least one vertex that has not been disabled. This is equivalent to find an assignment of truth values to the variables x_i that makes all clauses true in the original 3-SAT problem. Note that, if the $\mathscr{C}$-MP problem is feasible, the negated literal associated to the vertices of the final obstacles positions are a feasible solution of the 3-SAT problem. □

As shown in [45], (unconstrained) MAPF is NP-hard. Hence, it is obvious that also $\mathscr{C}$-MAPF is NP-hard. Anyway, for completeness, we show that the argument of Proposition 3.3.1 can be adapted to show NP-hardness of $\mathscr{C}$-MAPF.

Proposition 3.3.2. *$\mathscr{C}$-MAPF is NP-hard.*

Proof. Since the proof is a slight variant of the one of Proposition 3.3.1, we only briefly sketch it. With respect to the previous proof, we need to replace the components $\{o_i, x_i, \bar{x}_i\}$, $i = 1, \ldots, n$, displayed in Figure 3.6, with the components $\{o_i, x_i, \bar{x}_i, p_i\}$ displayed in Figure 3.8. Next, we consider the $\mathscr{C}$-MAPF instance with $n+1$ agents, initial configuration $\mathscr{A}^s = (s, o_1, \ldots, o_n)$, and final configuration $\mathscr{A}^t = (t, p_1, \ldots, p_n)$, with the same families of constraints presented in Proposition 3.3.1 and the additional family of constraints:

$$\{(\{d_{k+1}, p_i\}, 1), i = 1, \ldots, n\}. \tag{3.4}$$

The proof is based on the fact that this $\mathscr{C}$-MAPF instance admits a solution if and only if the $\mathscr{C}$-MP instance discussed in Proposition 3.3.1 admits a solution. Indeed, the family of constraints (3.4) imposes that vertices $p_1, \ldots, p_n$ can be occupied by

an agent only if the first agent is able to go through vertex d_{k+1} and reach the target vertex t, which is possible if and only if the $\mathscr{C}$-MP instance discussed in Proposition 3.3.1 admits a solution. □

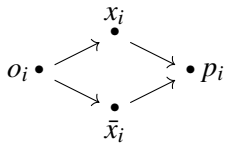


Figure 3.8: Component $\{o_i, x_i, \bar{x}_i, p_i\}$.

3.4 Strengthening of $\mathscr{C}$-MAPF into MAPF

Our main idea for solving a $\mathscr{C}$-MAPF instance $< G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} >$ (similarly for a $\mathscr{C}$-MP instance) is to derive a MAPF instance $< G', \mathscr{A}^s, \mathscr{A}^t >$ over a new graph G' (the *reduced graph*), in general with a lower number of vertices, where the constraints can be ignored (in the sense that they are satisfied by construction). Next, a feasible solution of the classical MAPF (or MP) on the reduced graph (computable in polynomial time, if it exists) can be converted into a solution of $\mathscr{C}$-MAPF (or $\mathscr{C}$-MP) on the original graph. In what follows, we describe how we build the reduced graph.

For a vertex set V, an ASC $\mathscr{C}$, and $Z \subset V$, we define $V_Z^{\mathscr{C}} = \{x \in V : Z \cup \{x\} \in \mathscr{C}\}$ as the set of vertices which can be added to Z, keeping it in the admissible collection $\mathscr{C}$. The following lemma shows that if $Z \subset W$, then the reverse inclusion holds for the corresponding sets $V_Z^{\mathscr{C}}, V_W^{\mathscr{C}}$.

Lemma 3.4.1. *Let $G = (V, E)$ be a digraph, $\mathscr{C}$ an ASC, and let $Z \subset W \subset V$. Then, $V_Z^{\mathscr{C}} \supset V_W^{\mathscr{C}}$.*

Proof. If $x \in V_W^{\mathscr{C}}$, then $Z \subset W$ implies that $\mathscr{C} \ni W \cup \{x\} \supset Z \cup \{x\}$. Since $\mathscr{C}$ is an ASC, this implies that $Z \cup \{x\} \in \mathscr{C}$. □

The reduced graph associated to a vertex subset $W \subset V$ is defined as follows.

Definition 3.4.1. *Let $G = (V, E)$ be a digraph, $\mathscr{C}$ an admissible collection, and $W \subset V$ a subset of vertices with $W \in \mathscr{C}$. The reduced digraph $G_W = (W, E_W)$ is such that*

$$\forall v_1, v_2 \in W \ (v_1, v_2) \in E_W \text{ if and only if}$$
$$\text{there exists a directed path on } G_W^{v_1 v_2} \text{ from } v_1 \text{ to } v_2,$$

where $G_W^{v_1 v_2} = (V_W^{v_1, v_2}, E_W^{v_1, v_2})$ is obtained from G erasing:

- *the vertices in $W \setminus \{v_1, v_2\}$;*
- *all the vertices $v \in V \setminus W$ such that $\{v\} \cup [W \setminus \{v_1, v_2\}] \notin \mathscr{C}$ (i.e., $v \notin V^{\mathscr{C}}_{W \setminus \{v_1, v_2\}}$).*

In other words, $G_W^{v_1 v_2}$ is the subgraph of G obtained by removing all vertices in W (apart from v_1 and v_2) and all vertices $v \in V \setminus W$, such that the subset obtained by adding v to $W \setminus \{v_1, v_2\}$ does not belong to admissible collection $\mathscr{C}$. This definition is justified by the fact that a pebble placed at v_1 can move to v_2 on subgraph $G_W^{v_1 v_2}$, in such a way that all subsets of vertices occupied by pebbles are in the admissible collection, even if all vertices in $W \setminus \{v_1, v_2\}$ are occupied. Indeed, $G_W^{v_1 v_2}$ contains only those vertices that can be safely added to $W \setminus \{v_1, v_2\}$, obtaining a set belonging to the admissible collection.

Definition 3.4.2. *Let $G = (V, E)$ be a digraph and let $\mathscr{C}$ be an admissible collection. A non-empty subset $W \subset V$ is **independent** on $(G, \mathscr{C})$ if G_W is strongly connected. We denote by $\mathscr{F}_G^{\mathscr{C}}$ the family of all independent subsets of V. The empty subset $\emptyset$ is always independent on $(G, \mathscr{C})$.*

For instance, consider graphs G and G' with admissible collection $\mathscr{C} := \mathscr{C}_{\mathscr{Q}}$ in Figures 3.2a and 3.2b, respectively. Subset $W_1 = \{2, 4\}$ is independent both on $(G, \mathscr{C})$ and on $(G', \mathscr{C})$ (see Figure 3.4a), while subset $W_2 = \{1, 3, 5\}$ is independent only on $(G', \mathscr{C})$ (see Figure 3.4b). Moreover, subset $W_3 = \{2, 4, 5\}$ is independent neither on $(G, \mathscr{C})$ nor on $(G', \mathscr{C})$. In particular, on G', $(5, 2) \notin E_{W_3}$, since on G' there does not exist a path from 5 to 2, after the removal of vertex $\{4\} = \{2, 4, 5\} \setminus \{2, 5\}$ and of vertex 1, since $\{1, 4\} \notin \mathscr{C}$ (see Figure 3.9).

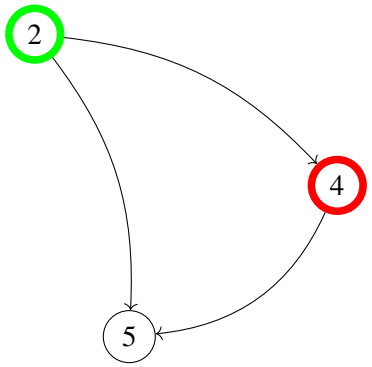


Figure 3.9: Reduced graph G_{W_3} for G' (Figure 3.2b), where $W_3 = \{2,4,5\}$ is not independent.

In what follows, we show that the family $\mathscr{F}_G^{\mathscr{C}}$ of all independent subsets of V is an ASC. Given a digraph G, the contraction of a vertex v is the digraph obtained from G by eliminating v and merging all incoming and outgoing edges of v.

Definition 3.4.3. *Let $G = (V,E)$ be a digraph and $v \in V$. The graph obtained from G by contracting vertex v is defined as $G/v = (V \setminus \{v\}, E')$, where E' contains*

i) all $e \in E$ such that $e \cap \{v\} = \emptyset$,

ii) all (u,w) such that $\{(u,v)(v,w)\} \subset E$.

For completeness, we state the well-known fact that strong connectedness is maintained after contracting a vertex.

Lemma 3.4.2. *Let $G_H = (V_H, E_H)$ be a strongly connected digraph and $v \in V_H$. Then, G_H/v is strongly connected.*

Proof. We need to show that there exists a directed path in G_H/v between any couple of vertices u,w. Since G_H is strongly connected, there is a directed path from u to w on G_H. If this path does not contain v, then this is also a path in G_H/v. If this path contains v, it can be written as $p = p_1 v^- v v^+ p_2$, where v^- and v^+ are the predecessor and the successor of v in this path. By definition, G_H/v contains edge (v^-, v^+) so that $p_1 v^- v^+ p_2$ is a path on G_H/v. □

The following proposition shows that the graph obtained by contracting a vertex w of the reduced graph G_W, where W is an independent set, is a subset of the reduced graph $G_{W\setminus\{w\}}$.

Proposition 3.4.1. *Let $G=(V,E)$ be a digraph and let $\mathscr{C}$ be an admissible collection. Let $W\subset V$ be an independent subset on $(G,\mathscr{C})$, and $w\in W$. Then, $G_{W\setminus\{w\}}\supset G_W/w$.*

Proof. Let (u,v) be an edge of G_W/w and let $G_W=(W,E_W)$. By definition of G_W/w, one of the following holds:

i) $(u,v)\in E_W$,

ii) $(u,w),(w,v)\in E_W$.

In case i), since $(u,v)\in E_W$, G contains a directed path p from u to v that belongs to $V^{\mathscr{C}}_{W\setminus\{u,v\}}$. Set $Z=W\setminus\{w\}$. Since $Z\subset W$, by Lemma 3.4.1, $V^{\mathscr{C}}_{Z\setminus\{u,v\}}\supset V^{\mathscr{C}}_{W\setminus\{u,v\}}$. Hence, p belongs to $V^{\mathscr{C}}_{Z\setminus\{u,v\}}$, and (u,v) is a directed edge of G_Z. In case ii), G contains a directed path p_1 from u to w that belongs to $V^{\mathscr{C}}_{W\setminus\{u,w\}}$, and a directed path p_2 from w to z that belongs to $V^{\mathscr{C}}_{W\setminus\{w,z\}}$. Since $Z=W\setminus\{w\}$, both paths belong to $V^{\mathscr{C}}_{Z\setminus\{u,v\}}$, so that their concatenation $p=p_1p_2$ is a path from u to v that belongs to $V^{\mathscr{C}}_{Z\setminus\{u,v\}}$. This implies that (u,v) is a directed edge of G_Z. □

The following proposition shows that each subset of an independent set is also independent.

Proposition 3.4.2. *Let $G=(V,E)$ be a digraph and $\mathscr{C}$ an admissible collection. Let $W\subset V$ be independent on $(G,\mathscr{C})$, and let $Z\subset W$. Then, Z is independent on $(G,\mathscr{C})$.*

Proof. Let $\{v_1,v_2,\dots,v_m\}=W\setminus Z$. Since W is independent on $(G,\mathscr{C})$, G_W is strongly connected. By Lemma 3.4.2, G_W/v_1 is also strongly connected. By Proposition 3.4.1, G_W/v_1 is a subgraph of $G_{W\setminus\{v_1\}}$, which implies that $G_{W\setminus\{v_1\}}$ is also strongly connected. By reiterating the same argument, we obtain that $G_{W\setminus\{v_1,\dots,v_m\}}$ is strongly connected. Hence, Z is independent on $(G,\mathscr{C})$. □

An immediate consequence of Proposition 3.4.2 is that the family of independent subsets is an ASC.

Proposition 3.4.3. *Let $G=(V,E)$ be a digraph and let $\mathscr{C}$ be an admissible collection. Let $\mathscr{F}_G^{\mathscr{C}}$ be a family of subsets of V, defined as in Definition 3.4.2. Then, $\mathscr{F}_G^{\mathscr{C}}$ is an ASC.*

Proof. In Definition 3.1.1, 1) is satisfied since, by definition of $\mathscr{F}_G^{\mathscr{C}}$, $\emptyset \in \mathscr{F}_G^{\mathscr{C}}$, 2) is a direct consequence of Proposition 3.4.2. □

Observation 3.4.1. *In the general case, $\mathscr{F}_G^{\mathscr{C}}$ is not a matroid. For instance, consider the following counterexample.*

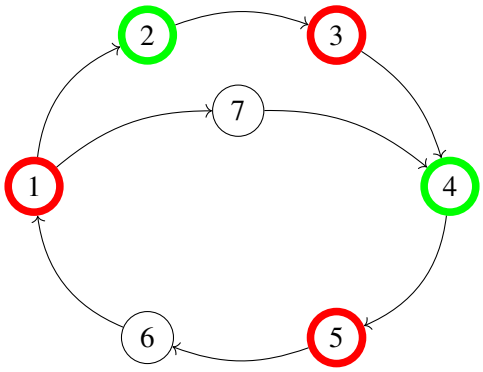


Figure 3.10: Counterexample

Let G be the digraph in Figure 3.10 with admissible collection $\mathscr{C}$ defined as in (3.2) where, for each pair $(S,k) \in \mathscr{Q}$, $k=1$ and S belongs to the collection:

$$\mathscr{S} = \{\{1,2\},\{2,3\},\{3,4\},\{4,5\},\{5,6\},\{6,1\}\}.$$

Let $W=\{1,3,5\} \in \mathscr{F}_G^{\mathscr{C}}$, $U=\{2,4\} \in \mathscr{F}_G^{\mathscr{C}}$ be subsets of V such that $|W|=|U|+1$. However, $\nexists v \in W \setminus U$ such that $U \cup \{v\} \in \mathscr{F}_G^{\mathscr{C}}$. Indeed, $\forall v \in W \setminus U$ $\exists u \in U$, $S \in \mathscr{S}$ such that $u,v \in S$. This means that the exchange axiom does not hold.

The following proposition shows that every edge (v,u) of the reduced graph G_W corresponds to a plan in G that brings a pebble from v to u, and is consistent with $\mathscr{C}$.

Proposition 3.4.4. *Let $G=(V,E)$ be a digraph, P a set of pebbles, $\mathscr{C}$ an admissible collection, and let $W \subset V$ be independent on $(G,\mathscr{C})$. Let $G_W=(W,E_W)$ be the*

reduced graph. For any $e = (u,v) \in E_W$, and any pebble configuration $\mathscr{A} \subset W$, with $u \in \mathscr{A}$ and $v \notin \mathscr{A}$, there exists a directed path p in G, from u to v, such that $(\forall r \in Pref(p))\, \rho(\mathscr{A}, r) \in \mathscr{C}$. We call such path p a **lift** *of edge e.*

Proof. By definition of an independent set, graph G_W is strongly connected. Since $\mathscr{A} \setminus \{u\} \subset W$ and since the family of independent sets $\mathscr{F}_G^{\mathscr{C}}$ is an ASC (see Proposition 3.4.3), then also graph $G_{\mathscr{A}\setminus\{u\}}$ is strongly connected. Hence, there is a directed path from u to v that visits only vertices that belong to set $\{x \in V : \{x\} \cup (\mathscr{A} \setminus \{u\}) \in \mathscr{C}\}$, which implies the thesis. □

We can define a transition function on the reduced graph as follows.

Definition 3.4.4. *Let $G = (V,E)$ be a digraph, $W \subset V$, and P a set of pebbles. Let $\rho : ((P \to V) \times E) \to (P \to V)$ be the corresponding transition function. Let $G_W = (W,E)$ be the reduced graph. The reduced transition function ρ_W is the transition function on G_W. Namely, $\rho_W : ((P \to W) \times E_W) \to (P \to W)$ is such that, for any valid configuration $\mathscr{A}_W \in P \to W$ and any edge (u,w) with $u \in \mathscr{A}_W$ and $w \notin \mathscr{A}_W$, $\rho_W(\mathscr{A}_W, e)$ is the configuration obtained from $\mathscr{A}_W$ by moving the pebble on vertex u to vertex v.*

Remark 3.4.1. *By Proposition 3.4.4, for any valid configuration $\mathscr{A}_W \in P \to W$ and $e \in E_W$, there exists a lift p such that $\rho_W(\mathscr{A}_W, e) = \rho(\mathscr{A}_W, p)$.*

The following theorem states that a feasible solution of MP or MAPF over a reduced graph can be converted into a solution of $\mathscr{C}$-MP or $\mathscr{C}$-MAPF over the original graph.

Theorem 3.4.1. *Let $G = (V,E)$ be a digraph and let $< G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} >$ be an instance of $\mathscr{C}$-MAPF. Let $W \subset V$ be independent on $(G,\mathscr{C})$ (i.e., $W \in \mathscr{F}_G^{\mathscr{C}}$), and assume that $\mathscr{A}^s, \mathscr{A}^t \subset W$. Let $p = e_1 \dots e_m \in E_W^*$ be a plan and define plan $\hat{p} = P_1 \dots P_m \in E^*$, where, for $i = 1, \dots, m$, P_i is a lift of e_i. If p solves MAPF $< G, \mathscr{A}^s, \mathscr{A}^t >$, then $\hat{p}$ solves $\mathscr{C}$-MAPF $< G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} >$, so that:*

$$< G_W, \mathscr{A}^s, \mathscr{A}^t > \text{ feasible } \Rightarrow < G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} > \text{ feasible.}$$

Proof. Plan $\hat{p}$ is consistent with $\mathscr{C}$ by Proposition 3.4.4. By Remark 3.4.1, $\rho(\mathscr{A}^s, \hat{p}) = \mathscr{A}^t$. □

Theorem 3.4.1 provides a strategy for finding a solution of $\mathscr{C}$-MAPF. Namely, we look for an independent set of vertices W that contains those associated to the initial and final positions, plus additional ones used for maneuvering, and we solve a standard MAPF problem on the reduced graph. Then, we *lift* the obtained solution to a solution of the original $\mathscr{C}$-MAPF.

As already mentioned, the converse implication

$$< G_W, \mathscr{A}^s, \mathscr{A}^t > \text{ feasible } \Longleftarrow < G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} > \text{feasible},$$

is not true in the general case.

In Theorem 3.4.1, Assumption $\mathscr{A}^s, \mathscr{A}^t \subset W$, $W \in \mathscr{F}_G^{\mathscr{C}}$, ensures the existence of an independent set containing both source and target vertices. If this assumption is not satisfied, it may still be possible to find a solution of $\mathscr{C}$-MAPF by solving a sequence of two MAPF problems. Indeed, assume that there exist two sets $W_1 \in \mathscr{F}_G^{\mathscr{C}}$ and $W_2 \in \mathscr{F}_G^{\mathscr{C}}$ such that $\mathscr{A}^s \subset W_1$, $\mathscr{A}^t \subset W_2$, and $|W_1 \cap W_2| \geq |P|$, (i.e., their intersection has cardinality greater than the number of pebbles). Let U be a subset of $W_1 \cap W_2$ such that $|U| = |P|$ and let $\mathscr{A}^i : P \to U$ be a valid configuration. Then, we replace the original $\mathscr{C}$-MAPF problem $< G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} >$, with a sequence of two $\mathscr{C}$-MAPF problems: $< G_{W_1}, \mathscr{A}^s, \mathscr{A}^i >$, which moves pebbles from the source positions to the intermediate positions, and $< G_{W_1}, \mathscr{A}^i, \mathscr{A}^t >$ which moves pebbles from the intermediate positions to the targets. The lift of the concatenation of the solutions of these two MAPF problems is a solution of the original $\mathscr{C}$-MAPF problem. This remark leads to the following Corollary of Theorem 3.4.1.

Corollary 3.4.1. *Let $G = (V, E)$ be a digraph and $\mathscr{F}_G^{\mathscr{C}}$ the family of subsets defined in Definition 3.4.2. Let $\mathscr{A}^s$ and $\mathscr{A}^t$ be initial and final configurations such that $\mathscr{A}^s \in \mathscr{F}_G^{\mathscr{C}}$ and $\mathscr{A}^t \in \mathscr{F}_G^{\mathscr{C}}$. Then, for all sets $W_1, W_2 \in \mathscr{F}_G^{\mathscr{C}}$ and for all configurations $\mathscr{A}^i$ such*

that $\mathscr{A}^i \subset W_1 \cap W_2$*:*

$$< G_{W_1}, \mathscr{A}^s, \mathscr{A}^i > \text{ and } < G_{W_2}, \mathscr{A}^i, \mathscr{A}^t > \text{ are feasible}$$
$$\Rightarrow < G, \mathscr{A}^s, \mathscr{A}^t, \mathscr{C} > \text{ is feasible.}$$

3.4.1 Finding an independent set of vertices of largest cardinality

In view of Theorem 3.4.1, we need to find an independent set of vertices W that contains $\mathscr{A}^s \cup \mathscr{A}^t$ (if it does not exist, we can proceed as suggested in Corollary 3.4.1). However, it is convenient that W contains the largest number of additional vertices. The number of holes (i.e., unoccupied vertices) in the reduced MAPF problem $< G_W, \mathscr{A}^s, \mathscr{A}^t >$ is given by $|W| - |\mathscr{A}_s|$, that is the difference between the number of available vertices and the number of pebbles. Various results in literature (see, for instance, [6, 52]) show that the feasibility of a MAPF problem depends on the number of holes. Intuitively, the larger the number of holes, the easier it is to find a feasible solution.

This observation leads us to consider the problem of maximizing the cardinality of W. We first introduce the notion of a maximal independent set. Intuitively, an independent set is maximal if it is not strictly included into any other independent set.

Definition 3.4.5. *An independent set* $W \in \mathscr{F}_D^{\mathscr{C}}$ *is* ***maximal*** *if* $\forall U \in \mathscr{F}_D^{\mathscr{C}}$ *such that* $W \subset U$*,* $U = W$*.*

Given a digraph $G = (V,E)$ and an admissible collection $\mathscr{C}$, we may find various maximal independent sets, of different cardinality. For example, in digraph G' of Figure 3.2b, both $W_1 = \{2,4\}$ and $W_2 = \{1,3,5\}$ are maximal independent sets. For this reason, it is natural to consider the following problem.

Definition 3.4.6. *Given a graph* $G = (V,E)$*, an admissible collection* $\mathscr{C}$*, and a subset* $W \subset V$ *such that* $W \in \mathscr{C}$*, the Maximum Independent Set with Additional Constraints (*$\mathscr{C}$*-MIS)* $< G, \mathscr{C}, W >$ *consists in finding an independent set that contains* W*, of the largest cardinality.*

In [69], it is proved that finding the largest independent set within a generic ASC is NP-hard. In what follows, we prove that NP-hardness (in fact, strong NP-hardness) holds also for $\mathscr{C}$-MIS (i.e., for the specific ASC considered in this Chapter). We prove the result by a reduction from the Maximum Independent Set (MIS) over graphs. Given a graph $G = (V,E)$, the MIS problem consists in searching for the largest independent subset U of V, where U is independent (in the classical sense) if $(\forall i, j \in U : i \neq j)\ (i, j) \notin E$.

Theorem 3.4.2. *$\mathscr{C}$-MIS is strongly NP-hard.*

Proof. The proof is based on a polynomial reduction of the classical Maximum Independent Set (MIS) problem on a graph $G = (V,E)$ to $\mathscr{C}$-MIS.
Starting from G, we define a new (undirected) graph, adding a new vertex q connected to all vertices of G. Formally, the new graph is $G' = (V',E')$, where $V' = V \cup \{q\}$ and $E' = E \cup \{(i,q) :\ i \in V\}$. We define an admissible collection $\mathscr{C}$ on G' as in (3.3) where we introduce:

- for each $(i, j) \in E$ a triple $(\{i, j\}, 1, (1,1)) \in \mathscr{Q}'$;
- the single triple $(V', k, w^{V'}) \in \mathscr{Q}'$, where $w_i^{V'} = 1$ for all $i \in V$, while $w_q^{V'} = 2$.

Then, we prove the equivalence of the following two problems:

a) G has a set of k independent vertices (in the classical sense);

b) there exists a subset of k vertices of V' that is $(G', \mathscr{C})$ independent.

Assume that $U \subset V$ is a solution of a). Since U is independent, $|U \cap \{i, j\}| \leq 1$ for all $(i, j) \in E$. Moreover, $q \notin U$, so that $\sum_{i \in U \cap V'} w_i^{V'} = k$. Moreover, each pair $v_1, v_2 \in U$ is connected by path $v_1 q v_2$ (since, $q \notin U$ and $\sum_{i \in [U \setminus \{v_1, v_2\}] \cup \{q\}} w_i^{V'} = k$). Hence, U is also a solution of b).

Conversely, assume that $U \subset V'$ is a solution of b). Note that $q \notin U$. Otherwise, we would have $\sum_{i \in V' \cap U} w_i^{V'} = k+1$, which would imply that $U \notin \mathscr{C}$. Hence, $U \subset V$ and U is a solution of a) since $|U \cap \{i, j\}| \leq 1$ for all $(i, j) \in E$.

□

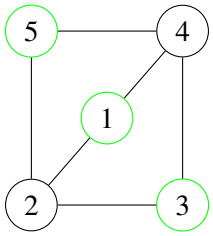


Figure 3.11: Solution of MIS on graph G

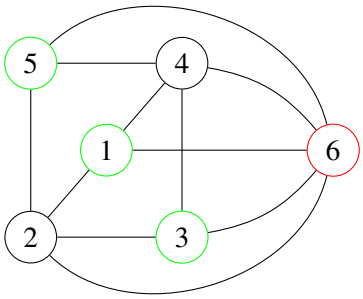


Figure 3.12: $\mathscr{C}$-MIS on G'

In order to illustrate the result, consider the MIS problem on G (Figure 3.11) and the $\mathscr{C}$-MIS problem on G' (Figure 3.12) with $\mathscr{S} = \{\{i, j\} \ : \ (i, j) \in E'\}$. It is easily observed that $S = \{1, 3, 5\}$ is the largest solution of MIS on G. We note that one of the maximum solutions of $\mathscr{C}$-MIS on G would be $T = \{1, 3\} \subset S$. If we consider graph G' of Figure 3.12, vertex 5 can also be added to the independent set T, as it is connected to both 1 and 3 by a free path through 6. Therefore, the solutions of the two problems on the two different graphs are equivalent.

3.4.2 $\mathscr{C}$-MP and $\mathscr{C}$-MAPF as supervisory control problems

Our approach for $\mathscr{C}$-MP and $\mathscr{C}$-MAPF is based on a limitation on the set of allowed moves. Namely, at any step, only one agent can occupy a vertex that does not belong to W. Further, if one agent occupies a vertex that is not in W, then no other agent can move until the first agent re-enters in W. Finally, all agents can move only to those

vertices from which there exists a directed path that leads to an unoccupied vertex in W. If we represent the motions of the agents in the digraph as a discrete event system, these limitations can be implemented by the definition of a supervisor (see [70, 71]). In our contest, a supervisor is a system that can disable some agents moves, if these can lead to deadlock configurations, that is configurations from which the desired final configuration cannot be reached. In the following, we discuss this fact in more detail for $\mathscr{C}$-MAPF. The case of $\mathscr{C}$-MP is similar. To frame $\mathscr{C}$-MAPF as a supervisory control problem we need some definitions. Language $L = \{f \in E^* : \rho(\mathscr{A}^s, f)!\}$ represents the subset of E^* consisting of all well-defined plans. The *specification* language $S = \{f \in L : (\forall r \in \text{Pref}(f))\ \rho(\mathscr{A}^s, r) \in \mathscr{C}\}$ represents the subset of L that satisfies the constraints. Finally, the marked language

$$L_m = \{f \in L : \rho(\mathscr{A}^s, f) = \mathscr{A}^t\},$$

consists on all plans that lead to the desired final configuration $\mathscr{A}^t$ from the initial one $\mathscr{A}^s$.

Using supervisory control terms, we refer to edge $u \to v$ as an *event*. It represents the motion of an agent from vertex u to vertex v. We assume that all events are *controllable*. That is, at any step, we can decide to disable an arbitrary subset of events. Following the definition in [70] and [71], a supervisor is a function $\Sigma : L \to \mathscr{P}(E)$ (where $\mathscr{P}(E)$ is the power set of E, that is the family of all subsets of E). For a plan $p \in L$, $\Sigma(p)$ denotes the set of all directed edges that the supervisor enables after the execution of p. We call $L(\Sigma), L_m(\Sigma)$ the subsets of L, L_m resulting from the intervention of Σ. That is $L(\Sigma)$ is the smallest set such that

1) $\varepsilon \in L(\Sigma)$

2) $(\forall p \in L, e \in E)\ p \in L(\Sigma), e \in \Sigma(p) \to pe \in L(S)$.

That is, we can add an edge e to a plan p only if the supervisor enables e after the execution of p. Moreover, $L_m(\Sigma) = L(\Sigma) \cap L_m$. The supervisor Σ is *nonblocking* if we can extend any plan in $L(\Sigma)$ to a plan in $L_m(\Sigma)$. In other words, Σ is nonblocking if it prevents deadlocks, that is, if we can extend any $p \in L(\Sigma)$ to a plan that brings all agents to the desired final positions.

A fundamental problem in supervisory control is finding a nonblocking supervisor

Σ that satisfies the specification language, that is, $L(\Sigma) \subset S$. Applied to $\mathscr{C}$-MAPF, this consists in finding a supervisor that avoids all configurations that do not belong to $\mathscr{C}$, and configurations from which the $\mathscr{C}$-MAPF instance cannot be solved.

In general, the decision problem of checking if such supervisor exists is *NP*-complete (see [72]). Essentially, this is due to the fact that, if we want to find the supervisor or check if it exists, we need to enumerate all states (or a significant portion of them). However, the cardinality of the states set grows exponentially with respect to the number of agents.

Our approach defines a nonblocking supervisor Σ such that $L(\Sigma) \subset S$. At every step, the supervisor ensures that at most one agent be outside the vertices in W. Moreover, it only allows those moves that can be completed to a directed path that ends in an unoccupied vertex of W. In more detail, given a vertex set $Z \subset W$, we define a directed graph G^v_Z similarly as graph G^{v_1,v_2}_W in Definition 3.4.1. Namely, we obtain G^v_Z from G by erasing:

- the vertices in $Z \setminus \{v\}$;
- all the vertices $w \in V \setminus Z$ such that $\{w\} \cup [Z \setminus \{v\}] \notin \mathscr{C}$ (i.e., $w \notin V^{\mathscr{C}}_{Z\setminus\{v\}}$).

Then, we define $G^v_{W,Z} = (V^v_{W,Z}, E^v_{W_Z})$ as the subgraph of G^v_Z containing only the vertices from which there exists a directed path to a vertex in $W \setminus Z$. In other words, $G^v_{W,Z}$ contains only those nodes from which there is a direct path to one of the currently unoccupied nodes in W (i.e., set $W \setminus Z$), assuming that all nodes in $Z \setminus \{v\}$ are occupied by a pebble. Moreover, the visited pebble configurations must belong to the admissible collection $\mathscr{C}$.

The supervisor is such that

$$\Sigma(p) = \mathscr{E}(\rho(\mathscr{A}^s, p)(P)).$$

Here, $\mathscr{E} : \mathscr{P}(V) \to \mathscr{P}(E)$ is a function such that, for $Z \subset V$, $\mathscr{E}(Z)$ is the subset of E consisting of those directed edgess that the supervisor enables if the pebbles are located at vertices Z. Note that $\rho(\mathscr{A}^s, p)(P) \subset V$ is the subset of vertices that are occupied by a pebble after the execution of plan p, from initial condition $\mathscr{A}^s$.

Set E_Z is defined in the following way. If $|Z \setminus W| = 1$ (that is, one agent is outside W), let $v = Z \setminus W$ be the vertex outside W. Then, set $E_Z = \{(s,t) \in E_Z^v : s = v\}$. That is, if one agent is outside W, only this agent can move, and the supervisor only allows moves along those edges that can be completed to a directed path that terminates at an unoccupied vertex in W. If $|Z \setminus W| = 0$ (that is, all agents are in W), set $E_Z = \{(s,t) \in E : (s,t) \in E_Z^s\}$. That is, all agents can move, but only along those edges that can be completed to a directed path that ends at an unoccupied vertex of W.

This supervisor is nonblocking if the resulting MAPF $< G_W, \mathscr{A}^s, \mathscr{A}^t >$ satifies one of the sufficient conditions for (unconstrained) MAPF appearing in literature. For instance, if G_W is strongly biconnected and has at least two unoccupied vertices (see [6]). This corresponds to the following.

Proposition 3.4.5. *If $W \supset \mathscr{A}^s \cup \mathscr{A}^t$, G_W is strongly biconnected, and if the cardinality of W is greater or equal to the number of agents plus two, there exists a nonblocking supervisor Σ such that $L(\Sigma) \subset S$.*

3.4.3 Heuristics for the *C-MIS* problem

Paull and Unger's procedure, presented in [69], finds the maximal independent sets, and, in particular, the largest one, within a general ASC. As already mentioned, reference [69] also proves that detecting the largest maximal independent set is NP-hard. Hence, we can apply Paull and Unger's procedure to solve $\mathscr{C}$-MIS, but only for very small instances.

In Theorem 3.4.2, we proved that also $\mathscr{C}$-MIS is (strongly) NP-hard. In the following, we introduce polynomial-time heuristic algorithms for obtaining a good quality, sub-optimal solution of $\mathscr{C}$-MIS. We describe an iterative procedure that finds a maximal independent set M over a graph $G = (V,E)$. In the procedure, described in Algorithm 3.4.1, we first set $M = W$, where W is some initial set belonging to $\mathscr{C}$. Set W can be equal to $\mathscr{A}^s \cup A^t$ in view of Theorem 3.4.1 or to $\mathscr{A}^s \cup A^i$ (or $\mathscr{A}^i \cup A^t$) in view of Corollary 3.4.1. Then, at each iteration, it searches for a vertex v such that $M \cup \{v\}$ is independent, among all the vertices of set $\Delta := V \setminus \mathscr{E}$. Set $\mathscr{E}$ contains the vertices that can no longer be added. In particular, $\mathscr{E} = M \cup \mathscr{L}(M)$ where

$$\mathscr{L}(M) = \{w \in V : \ M \cup \{w\} \notin \mathscr{C}\}.$$

If such vertex does not exist, M is a maximal independent set and the procedure terminates. Otherwise, v is added to set M, $\mathscr{L}(M)$ is updated, and a new iteration is performed. To avoid repetition in the following iterations, all selected vertices in this iteration are added to $\mathscr{E}$.

Observation 3.4.2. *Let $G = (V,E)$ be a directed graph, $M \subset V$ a subset of vertices, and $\mathscr{C}$ defined as in (3.3) the admissible collection. Then, building the reduced graph $G_M = (M, E_M)$ has worst time complexity $O(n^2(n^2+h))$, where $n = |V|$ and $h = |\mathscr{Q}|$.*

Proof. The procedure involves two steps for each couple of vertices $(u,v) \in V \times V$:

- constructing $G_M^{u,v} = (V_M^{u,v}, E_M^{u,v})$;
- checking whether there exists a path from u to v on $G_M^{u,v}$.

The latter step can be done by depth-first search, which takes $O(|V_M^{u,v}| + |E_M^{u,v}|)$. However, in the worst case of a complete graph, it has time complexity $O(n^2)$. As for the former, if $\mathscr{C}$ is defined as in (3.3), it takes $O(h)$, where $h = |\mathscr{Q}|$. Indeed, to build this graph, for each $(S_i, k_i, w_i^S) \in \mathscr{Q}$ we have to check which vertices $w \in V$ are such that $\sum_{j \in S_i \cap M \setminus \{u,v\}} w_j^S \leq k_i$.

To do that, we propose the following procedure. For each vertex $x \in V$ we define a vector $\bar{x}$ of length h where, for each $i = 1, \dots, h$,

$$\bar{x}_i = \begin{cases} w_x^S & \text{if } x \in S_i, \\ 0 & \text{otherwise.} \end{cases} \tag{3.5}$$

Moreover, we define a vector m of length h, which represents at each iteration which is the occupied capacity of the vertices that stay in the intersection between M and S_i:

$$m_i = \sum_{j \in S_i \cap M} w_j^S.$$

This vector is initialized with the null vector, and it is updated iteratively each time a vertex x is added to M:

$$m_i = m_i + \bar{x}_i, \quad \forall i = 1, \ldots, h.$$

Therefore, checking whether $\sum_{j \in S_i \cap M \setminus \{u,v\}} w_j^S \leq k_i$ is equivalent to check whether $(m_i - \bar{u}_i - \bar{v}_i + \bar{x}_i) \leq k_i$, which takes $O(h)$.

□

Proposition 3.4.6. *If $\mathscr{C}$ is defined as in (3.3), Algorithm 3.4.1 has worst time complexity $O(n^3(n^2+h))$, where $n = |V|$ and $h = |\mathscr{Q}|$.*

Proof. By Observation 3.4.2, at each iteration of Algorithm 3.4.1, building the reduced graph $G_M = (M, E_M)$ has worst time complexity $O(n^2(n^2+h))$. Moreover, checking whether G_M is strongly connected is dominated by its construction and takes $O(|M| + |E_M|)$. However, in the worst case of a complete graph, it has time complexity $O(n^2)$. Therefore, checking whether M belongs to $\mathscr{F}_G^{\mathscr{C}}$ takes $O(n^2(n^2+h))$. Since each vertex is selected at most once, then the operation of checking whether a set is independent is performed at most n times. It follows that Algorithm 3.4.1 has worst time complexity $O(n^3(n^2+h))$. □

Algorithm 3.4.1. ***Finding maximal independent set.***

Let M be a (possibly empty) independent set;
$MaxSetFound = false;$
$\mathscr{E} = M \cup \mathscr{L}(M);$
$\Delta = V \setminus \mathscr{E};$
while $MaxSetFound = false$ ***do***
    $NewVertexFound = false;$
    while $\Delta \neq \emptyset$ & $NewVertexFound = false$ ***do***
        Select $v \in \Delta$;
        $N = M \cup \{v\};$
        if $N \in \mathscr{F}_G^{\mathscr{C}}$ ***then***
            $M = N$
            $NewVertexFound = true;$
            Update $\mathscr{L}(M)$;

```
14:            E = E ∪ L(M);
15:        end if
16:        E = E ∪ {v};
17:        Δ = Δ \ E;
18:    end while
19:    if NewVertexFound = false then
20:        MaxSetFound = true;
21:    end if
22: end while
```

Different heuristics can be obtained by changing the rule to select vertex v. The first one is the *Random* approach, which, at each step, selects the next vertex randomly. The second one is the *Greedy* approach, where vertex v is selected through a selection criterion $\psi : V \times \mathscr{C} \to \mathbb{R}$:

$$v = \arg\max_{w \in V \setminus M} \psi(w, M).$$

The selection criterion can be defined in different ways. If the admissible collection $\mathscr{C}$ is defined as in (3.2), an idea is to give precedence to vertices that are less constrained by other vertices. In this case, we can define the selection criterion as follows

$$\psi(v, M) := |\{w \in V : M \cup \{v, w\} \in \mathscr{C}\}|. \tag{3.6}$$

The selection criterion may also take into account the form of $\mathscr{C}$. For example, let $\mathscr{C}$ be defined as in (3.2) where, for each pair $(S,k) \in \mathscr{Q}$, $k = 1$ and S belongs to the collection:

$$\mathscr{S} = \{\{i, j\} : (i, j) \in E\}. \tag{3.7}$$

Then, if we indicate with $G \setminus \mathscr{E}$ the subgraph of G obtained by erasing the vertices in $\mathscr{E}$, a possible selection criterion is the inverse of the degree on $G \setminus \mathscr{E}$:

$$\psi(v, M) := \frac{1}{\deg_{G \setminus \mathscr{E}}(v)}. \tag{3.8}$$

Indeed, vertices with a lower degree are vertices bounded to a smaller number of other vertices. Therefore, adding a vertex with a lower degree to the independent set usually means having more choices in the following steps.

3.5 Experimental results

3.5.1 Real-life applications examples of $\mathscr{C}$-MAPF

Consider the digraph $G = (V,E)$ in Figure 3.13. It represents a small warehouse with three agents. Agent 1 has to move from s_1 to t_1, agent 2 from s_2 to t_2, and agent 3 from s_3 to t_3. To maintain a safety distance, we require that agents cannot occupy adjacent vertices at the same time. This is equivalent to consider an admissible collection $\mathscr{C}$ defined as in (3.2), where, for each pair $(S,k) \in Q$, $k = 1$ and $S \in \mathscr{S}$ (where $\mathscr{S}$ is defined as in (3.7)). Note that the union of the sources $S = \{s_1, s_2, s_3\}$ and of the targets $T = \{t_1, t_2, t_3\}$ is an independent set on $(G, \mathscr{C})$. To reduce $\mathscr{C}$-MAPF to MAPF, we use Algorithm 3.4.1, setting $W = S \cup T$, to find a maximal independent set. In this way, we are able to obtain a reduced graph G'. We can solve the (unconstrained) MAPF problem on G'. Here, we do not discuss the solution of MAPF, since it is extensively discussed in literature. Finally, we lift the obtained MAPF solution to a solution of the original $\mathscr{C}$-MAPF. In Figure 3.13, we show the best maximal independent set on G, containing W, obtained after 100 iterations of Algorithm 3.4.1 with the *Random* approach, and the correponding reduced graph G'.

As a second example, we considered a real-life warehouse layout, provided by packaging company Ocme S.r.l., based in Parma, Italy. The admissible collection $\mathscr{C}$ takes into account the dimensions of the AGV vehicles. The set $\mathscr{C}$ is defined as in (3.2), where for each pair $(S,k) \in Q$, $k = 1$ and $S \in \mathscr{S}$, with $\mathscr{S}$ defined as follows. For each vertex $v \in V$ on the graph, the company provides a set of forbidden edges $\tilde{E}_v$. That is, if an agent occupies v, the moves corresponding to the elements $\tilde{E}_v$ cannot be made by any agent. We define $\mathscr{S}$ as the set of triplets of vertices (v,i,j) such that $(i,j) \in \tilde{E}_v$:

$$\mathscr{S} = \{(v,i,j) : \ v,i,j \in V, \ (i,j) \in \tilde{E}_v\}.$$

We ran 100 times the random variant of Algorithm 3.4.1, and we computed a maximal independent set of cardinality equal to 20. The red vertices of Figure 3.14 show this independent set.

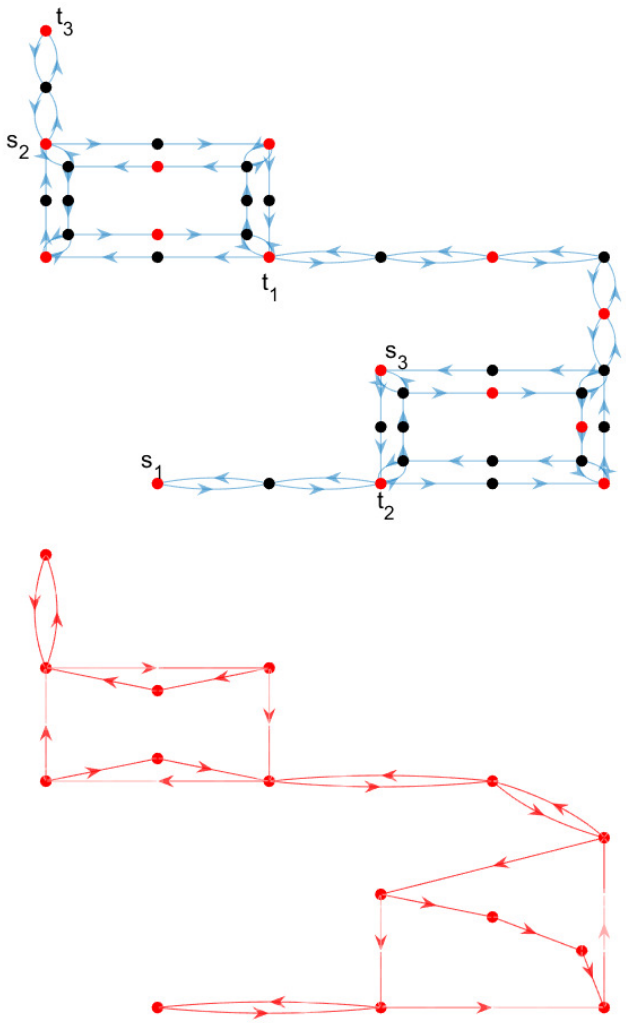


Figure 3.13: The upper figure shows a graph representing a simple industrial $\mathscr{C}$-MAPF scenario. Labels s_1, s_2, s_3 mark the initial vertices, and t_1, t_2, t_3 the final ones. Red vertices belong to the chosen independent set W. The lower figure shows the corresponding reduced graph.

These results suggest that on a real-life industrial scenario, the number of independent vertices is sufficiently large to solve $\mathscr{C}$-MAPF problems with a significant

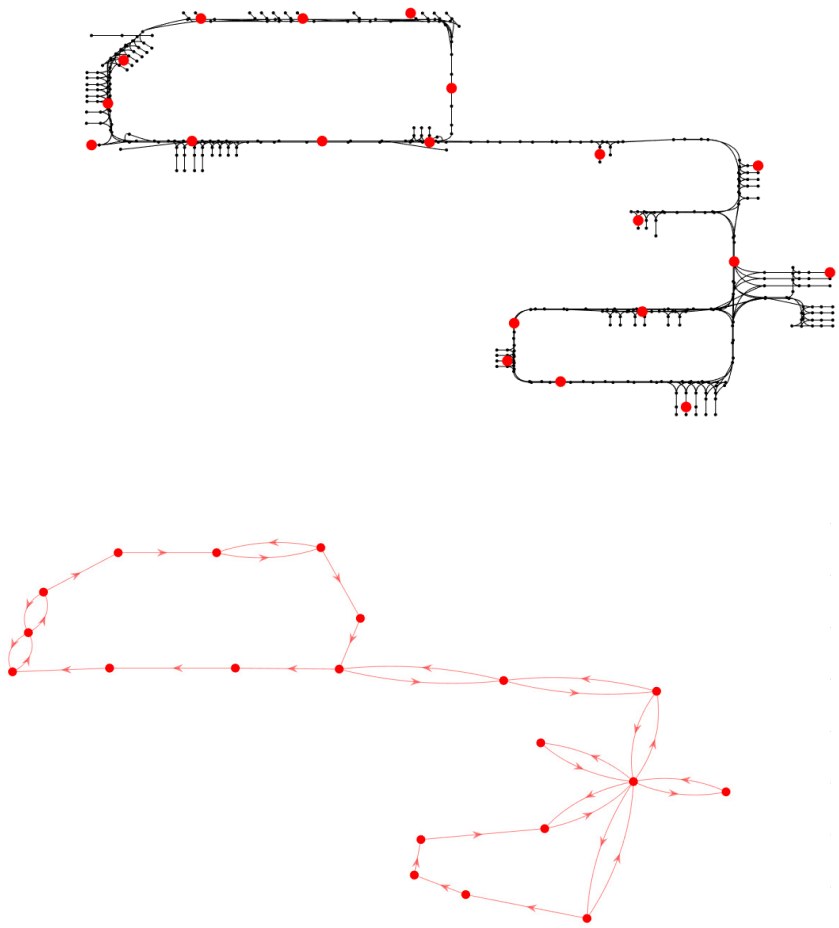


Figure 3.14: The upper figure shows a graph associated to a real-life warehouse layout. The red vertices are the ones selected in the independent set W. The lower figure shows the corresponding reduced graph.

number of agents. In particular, taking into account the number of vertices of the two graphs and the length of the corridors, and using the results of [52, 54], it is possible to show that, in the reduced graphs of Figure 3.13, we can solve all MAPF instances with up to 9 agents. Instead, in the reduced graph of Figure 3.14, we can solve all

MAPF instances with up to 14 agents.

Figure 3.15 shows a $\mathscr{C}$-MAPF instance with 5 agents on this digraph. Note that five is a significant number of agents for this graph, since the size is relatively small and the constraints are quite tight. We represent each agent with a different color. Circle marks show initial agents positions, and diamonds shows final ones. Using the random variant of Algorithm 3.4.1, we computed a maximal independent set of cardinality 19, containing initial and final agents positions. Figure 3.16 represents the computed reduced digraph (red), overlaying the original digraph (black).

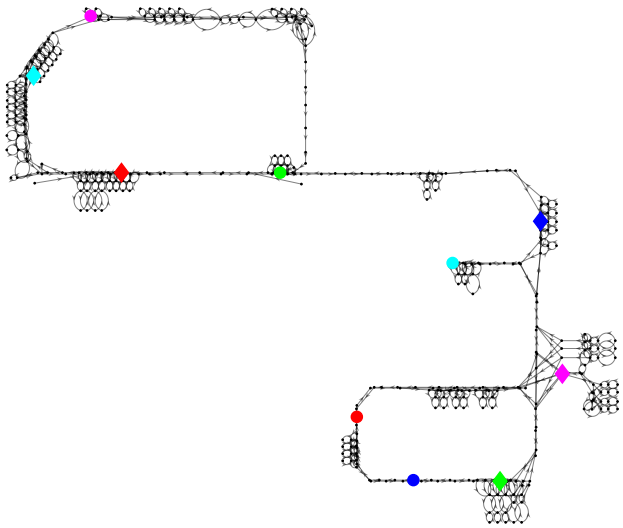


Figure 3.15: A $\mathscr{C}$-MAPF instance on the graph associated to the warehouse layout. We represent each agent with a different color. Circle marks represent initial positions and diamonds final ones.

Figure 3.17 represents the (unconstrained) MAPF problem on the reduced graph.

We solved the MAPF problem on the reduced graph, using the algorithm presented in Section 2.3.5. The computed plan consists of 98 moves, without synchronous moves. To reduce the number of time-steps, we used the local search procedure presented in Chapter 4 on the reduced graph. This method allows finding solutions

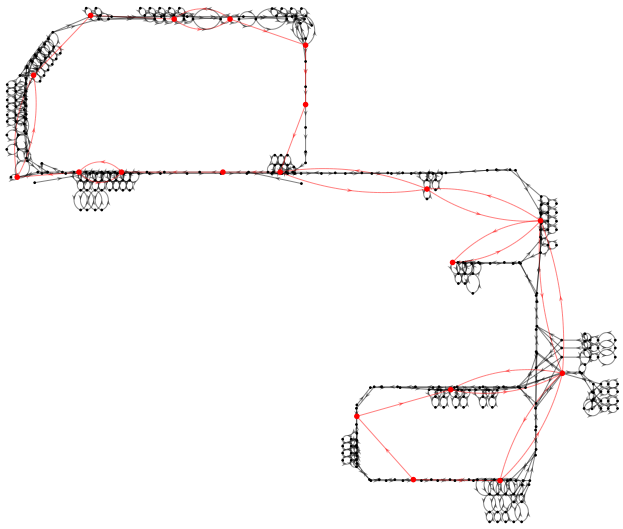


Figure 3.16: The reduced graph (red), overlaying the original graph (black).

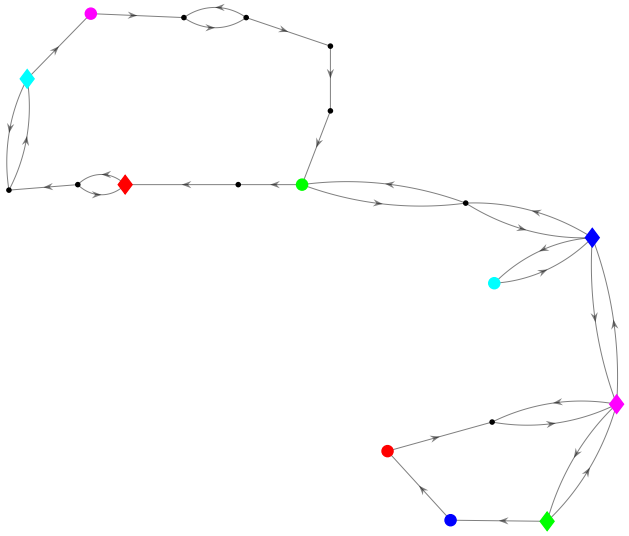


Figure 3.17: The (unconstrained) MAPF instance on the reduced graph.

with synchronous moves. We obtained a solution with a time-length of 29 time-steps. Alternatively, we could first lift the initial solution to a feasible solution of the original $\mathscr{C}$-MAPF problem, and then perform a local search for the $\mathscr{C}$-MAPF problem, starting from the lifted solution.

3.5.2 $\mathscr{C}$-MIS problem on square grid graphs

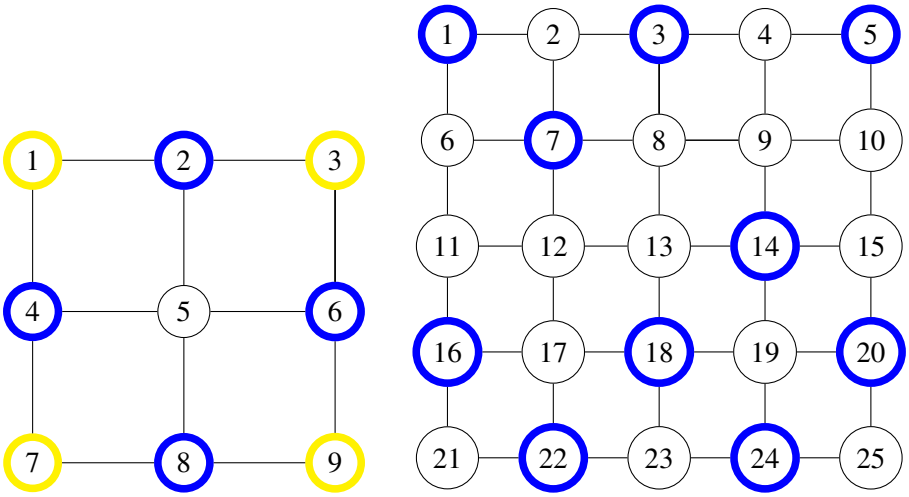
We focus on the problem of finding the independent set of maximum cardinality. In particular, we consider square grid graphs, as in Figures 3.18a, 3.18b. The admissible collection $\mathscr{C}$ is defined as in (3.2), where, for each pair $(S,k) \in Q$, $k = 1$ and $S \in \mathscr{S}$ (where $\mathscr{S}$ is defined as in (3.7)). In other words, at any time, at most one agent can be positioned in the two vertices associated to each edge.

First, we found the independent sets of maximal cardinality by Paull and Unger's algorithm. Due to the computational complexity of this algorithm (recall that $\mathscr{C}$-MIS is NP-hard), we could not solve instances on grids larger than 5×5 vertices in a reasonable time. Indeed, solving the 5×5 required more than 20 hours.

Figure 3.18a shows the two optimal solutions on the 3×3 grid, while Figure 3.18b shows one optimal solution W_1 for the 5×5 grid. The optimal solution of the latter is unique up to the isometries belonging to the dihedral group D_4. Indeed, all the other seven optimal solutions can be found with the composition of $k\pi/4$ rotations and symmetries with respect to the axes.

We also tested the heuristic algorithms presented in Section 3.4.3 to find independent vertices (without guarantee of optimality) on square grid graphs with $n \times n$ vertices, for $n = 2, \dots, 12$. In particular, we implemented Algorithm 3.4.1 both with the purely random selection rule and with the greedy rule (defined as in (3.8)).

For each graph, we ran the random algorithm 100 times, while we ran the greedy version only once, since the greedy rule is deterministic. Figure 3.20 presents the distribution of the cardinality of the independent sets returned by the random approach, compared with that of the greedy algorithm (the blue square is the cardinality of the greedy solution). Note that the single solution returned by the greedy approach is usually better than the average solution returned by the random one. However, the best solution over the 100 runs of the first approach is usually better than the single solution



(a) Grid Graph of 3×3. $W_1 = \{1,3,7,9\}$ and $W_2 = \{2,4,6,8\}$ are the two optimal solutions of $\mathscr{C}$-MIS.

(b) Grid Graph of 5×5. $W_1 = \{1,3,5,7,14,16,18,20,22,24\}$ is an optimal solutions of $\mathscr{C}$-MIS.

Figure 3.18: $\mathscr{C}$-MIS on grid graphs.

returned by the greedy approach, as shown in Figures 3.19, in which we compare these values and the one of the optimal solution returned by Paull and Unger's algorithm.

These experimental results show that the heuristic algorithms find suboptimal solutions of good quality, that do not differ much from each other and from the best solutions. However, we are not able to certify that we have found the maximum cardinality independent set for grid graphs of large dimensions.

n	Random	Greedy	optimal solution
2	2	2	2
3	4	3	4
4	6	6	6
5	10	9	10
6	14	13	-
7	18	17	-
8	23	22	-
9	29	27	-
10	35	32	-
11	43	40	-
12	50	49	-

Figure 3.19: Comparison of maximum cardinalities of independent sets found with Random, Greedy, and the optimal algorithm.

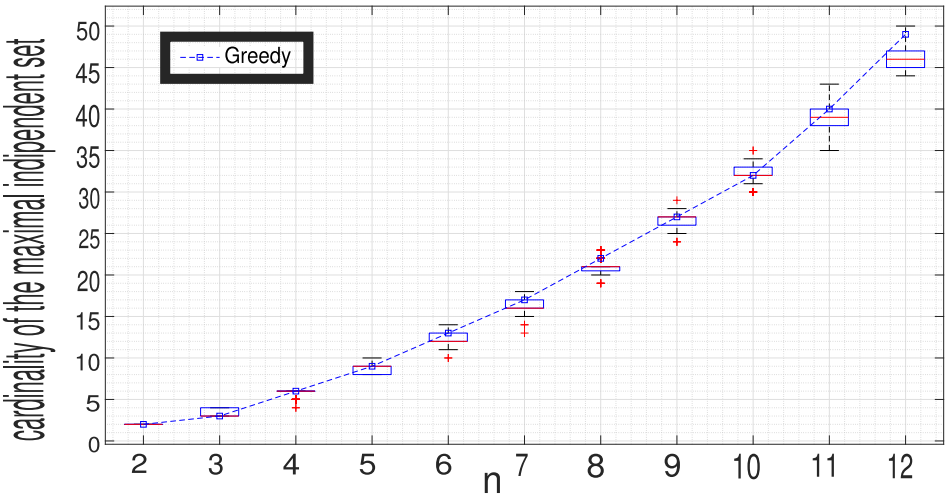


Figure 3.20: Comparison of random and greedy.

Chapter 4

Local Optimization of MAPF Solutions on Directed Graphs

We focus on the Multi-Agent Path Finding (MAPF) problem [1, 45]. Among sub-optimal MAPF solvers, rule-based algorithms are particularly appealing since they are complete. Even in crowded scenarios, they allow finding a feasible solution that brings each agent to its target, preventing deadlock situations [48]. However, generally, rule-based algorithms provide solutions that are much longer than the optimal one. This is a crucial limitation in industrial applications. The main contribution of this Chapter is the introduction of an iterative local search procedure in MAPF. We start from a feasible suboptimal solution, for instance the one provided by a rule-based algorithm. We perform a local search in a neighborhood of this solution, to find a shorter one. Iteratively, we repeat this procedure until the solution cannot be shortened any longer. At the end, we obtain a solution, that is still sub-optimal, but, in general, of much better quality than the initial one. We use dynamic programming for the local search procedure. Under this respect, the fact that our search is *local* is fundamental to reduce the time complexity of the algorithm. Indeed, in principle, it is possible to solve the general MAPF problem by dynamic programming. However, the number of explored states grows exponentially with the number of agents, so that we cannot apply standard dynamic programming to problems involving many agents. As we

will see, the introduction of a locality constraint allows solving the (local) dynamic programming problem in a time that grows only polynomially with respect to the number of agents (see Theorem 4.3.1).

The structure of this Chapter is the following.

In Section 4.1 we define the concept of distance between plans, and we introduce an equivalence class of the plans which considers as equal all the plans that have the same length, the same distance from a starting solution and lead to the same final configuration of the agents. These concepts are necessary to define $\mathscr{N}_r(f_0)$, the neighborhood of radius r of a solution f_0, as the set of all the equivalence classes which have at most distance r from the starting solution.

In Section 4.2 we introduce the iterative Neighborhood Search (Algorithm 1), which performs a local search in the neighborhood $\mathscr{N}_r(f_0)$ of the initial solution f_0, to find a shorter one. Iteratively, this algorithm repeats this procedure until the solution cannot be shortened any longer.

In Section 4.3 we describe a Dynamic Programming algorithm which is used at each iteration of the Neighborhood Search (Algorithm 2) for the optimal solution of the MAPF problem within the neighborhood $\mathscr{N}_r(f_0)$. In Theorem 4.3.1 we prove that the combination of the Neighborhood Search with the Dynamic Programming has polynomial time complexity with respect to the number of nodes of the graph.

In Section 4.4 we performed two sets of experiments on different graphs and with initial solutions generated in two distinct ways. In both experiments, we used the Neighborhood Search with the *Dynamic Programming* to improve the given initial solutions. In particular, we show that on strongly connected graphs with a high number of agents compared to the number of nodes, the proposed local search algorithm manages to improve the solutions provided by diSC (see Section 2.3.5) by up to 80%.

4.1 Problem Definition

4.1.1 MAPF problems

Let $G = (V, E)$ be a directed graph, with vertex set V and edge set E. We assign a unique label to each agent, and set P contains these labels. We remind that a *configuration* is a function $\mathscr{A} : P \to V$ that assigns the occupied vertex to each agent. A configuration is *valid* if it is injective (i.e., each vertex is occupied by at most one agent). Moreover, we remind that set $\chi \subset \{P \to V\}$ represents all valid configurations.

Time is assumed to be discretized. At every time step, each agent occupies one vertex and executes a single action. There are two types of actions: *wait* and *move*. We denote the wait action by ι. An agent that executes this action remains in its current vertex for another time step. We denote a move action by $u \to v$. In this case, the agent moves from its current vertex u to an adjacent vertex v (i.e., $(u, v) \in E$). Therefore, the set of all possible actions for a single agent is $\bar{E} = E \cup \{\iota\}$.

Function $\rho : \chi \times \bar{E} \to \chi$ is a partially defined transition function such that $\mathscr{A}' = \rho(\mathscr{A}, u \to v)$ is the configuration obtained by moving an agent from u to v:

$$\mathscr{A}'(q) := \begin{cases} v, & \text{if } \mathscr{A}(q) = u; \\ \mathscr{A}(q), & \text{otherwise .} \end{cases} \tag{4.1}$$

We remind that notation $\rho(\mathscr{A}, u \to v)!$ means that the function is well-defined. In other words $\rho(\mathscr{A}, u \to v)!$ if and only if $(u, v) \in E$ and $\mathscr{A}' \in \chi$. Moreover, $(\forall \mathscr{A} \in \chi)\ \rho(\mathscr{A}, \iota)!$ and $\rho(\mathscr{A}, \iota) = \mathscr{A}$.

Unlike Chapter 2 and Chapter 3, in this Chapter the movements of the agents can be synchronous. Therefore, at each time step an action is an element of $\mathscr{E} = \bar{E}^{|P|}$, $a = (a_1, \ldots, a_{|P|})$ where a_i is the single move of agent i. We can extend function $\rho : \chi \times \bar{E} \to \chi$ to $\rho : \chi \times \mathscr{E} \to \chi$, by setting $\mathscr{A}' = \rho(\mathscr{A}, a)$ equal to the configuration obtained by moving agent i along edge a_i (or by not moving the agent if $a_i = \iota$). In this case, $(\forall a \in \mathscr{E}, \mathscr{A} \in \chi)\ \rho(\mathscr{A}, a)!$ if and only if the following conditions hold:

1. $\mathscr{A}' \in \chi$: two or more agents cannot occupy the same vertex at the same time step;

2. $\forall i = 1,\dots,|P|$, if $a_i = (u,v)$, then $\nexists j \in \{1,\dots,|P|\}$ such that $a_j = (v,u)$: two agents cannot swap locations in a single time step.

We represent plans as ordered sequences of actions. It is convenient to view the elements of $\mathscr{E}$ as the symbols of a language. We denote by $\mathscr{E}^*$ the Kleene star of $\mathscr{E}$, that is the set of ordered sequences of elements of $\mathscr{E}$ with arbitrary length, together with the empty string ε: $\mathscr{E}^* = \bigcup_{i=1}^{\infty} \mathscr{E}^i \cup \{\varepsilon\}$.

We extend function $\rho : \chi \times \mathscr{E} \to \chi$ to $\rho : \chi \times \mathscr{E}^* \to \chi$. $(\forall s \in \mathscr{E}^*, e \in \mathscr{E}, \mathscr{A} \in \mathscr{C})$ $\rho(\mathscr{A}, se)!$ if and only if $\rho(\mathscr{A}, s)!$ and $\rho(\rho(\mathscr{A}, s), e)!$ and, if $\rho(\mathscr{A}, se)!$, then $\rho(\mathscr{A}, se) = \rho(\rho(\mathscr{A}, s), e)$.

Note that ε is the trivial plan that keeps all agents and holes at their positions.

We denote by $\mathscr{E}^*_{\mathscr{A}} = \{f \in \mathscr{E}^* : \rho(\mathscr{A}, f)!\}$ the set of plans such that $\rho(\mathscr{A}, f)$ is well defined. The problem of detecting a feasible solution is the following:

Problem 4.1.1. *(**Feasibility MAPF problem**). Given a digraph $G = (V,E)$, an agent set P, an initial valid configuration $\mathscr{A}^s$, and a final valid configuration $\mathscr{A}^t$, find a plan f such that $\mathscr{A}^t = \rho(\mathscr{A}^s, f)$.*

Now, for a feasible plan f, we define $|f|$ as the length of plan f, i.e., the number of time steps needed to let all agents reach the final configuration through plan f. Furthermore, given $k \in \mathbb{N}$, we denote by f_k the k-th prefix of f (that is, the prefix of f of length k, made up of the first k actions of f). Note that $|f_k| = k$.

We aim to solve a given MAPF instance while minimizing a global cumulative cost function. We employ the cost function called *Makespan*, equal to the time when the last agent reaches its destination (i.e., the maximum of the individual costs).

Problem 4.1.2. *(**Optimization MAPF problem**). Given $\mathscr{A}^s$ and $\mathscr{A}^t$ initial and final valid configurations on a digraph G, the optimization MAPF problem with Makespan is defined as*

$$\begin{aligned} &\min |f| \\ &s.t.\ \ \mathscr{A}^t = \rho(\mathscr{A}^s, f) \\ &f \in \mathscr{E}^*_{\mathscr{A}^s}. \end{aligned} \tag{4.2}$$

Let f_1 and f_2 be two solutions of the feasibility MAPF problem. We say that f_1 has better quality than f_2 (or, equivalently, f_2 is longer then f_1) if $|f_1| < |f_2|$. Other cost functions have also been used in the literature. *Sum-of-costs*, for example, is the summation, over all agents, of the number of time steps that an agent employs to reach its target without leaving it again. Unfortunately, finding the optimal solution, i.e., the minimal *Makespan* or *sum-of-costs*, has been shown to be NP-hard [44]. Therefore, in this Chapter we propose an approach to detect a good quality sub-optimal solution in polynomial time.

4.1.2 Distances

As said, we propose a solution approach based on the exploration of a *neighborhood* of a reference plan. To define a neighborhood, we introduce distances between vertices, configurations, and plans. Let $G = (V,E)$ be a digraph and P be a set of agents. We define the distance of vertex u from vertex v as the length of the shortest path on G from v to u: $d(u,v) = \ell(\pi_{vu})$, where π_{vu} is the shortest path in G from v to u and $\ell(\pi_{vu})$ is the length of that path, defined as the number of edges of π_{vu}. Note that d is not symmetrical, since π_{uv} and π_{vu} can be different. Next, we define the distance of configuration $\mathscr{A}^1$ from configuration $\mathscr{A}^2$ as the sum of the distances between the vertices that each agent occupies in the two configurations:

$$d : \chi \times \chi \to \mathbb{N} \qquad d(\mathscr{A}^1, \mathscr{A}^2) = \sum_{p \in P} d(\mathscr{A}^1(p), \mathscr{A}^2(p)).$$

Finally, we define the asymmetrical distance between two plans in $\mathscr{E}^*$. To do that, we associate to each plan a function in $\mathscr{L} = \{\psi : \mathbb{N} \to \chi\}$ using the following

$$\Phi_{\mathscr{A}} : \mathscr{E}^*_{\mathscr{A}} \to \mathscr{L}, \qquad f \to \psi_f(k) := \begin{cases} \rho(\mathscr{A}, f_k), & k < |f|, \\ \rho(\mathscr{A}, f), & k \geq |f|, \end{cases}$$

where ψ_f is the function which associates to each $k \in \mathbb{N}$ the configuration at step k, that depends on the k-th prefix of f. We define the distance of plan f from plan g as the distance between the associated functions $\Phi_{\mathscr{A}}(f), \Phi_{\mathscr{A}}(g)$:

$$d : \mathscr{E}^*_{\mathscr{A}} \times \mathscr{E}^*_{\mathscr{A}} \to \mathbb{N} \qquad d(f,g) := \bar{d}(\Phi_{\mathscr{A}}(f), \Phi_{\mathscr{A}}(g)).$$

We can define $\bar{d}$ in various ways, leading to different definitions of the distance between plans f and g:

1. ∞-distance:

$$\bar{d}_\infty(\Phi_{\mathscr{A}}(f),\Phi_{\mathscr{A}}(g)) := \max_{1\leq k\leq \min\{|f|,|g|\}} d(\psi_f(k),\psi_g(k));$$

2. 1-distance:

$$\bar{d}_1(\Phi_{\mathscr{A}}(f),\Phi_{\mathscr{A}}(g)) := \sum_{k=1}^{\min\{|f|,|g|\}} d(\psi_f(k),\psi_g(k));$$

3. max-min distance:

$$\bar{d}^*_\infty(\Phi_{\mathscr{A}}(f),\Phi_{\mathscr{A}}(g)) := \max_{k\in\mathbb{N}} \min_{h\in\mathbb{N}} d(\psi_f(k),\psi_g(h));$$

4. sum-min distance:

$$\bar{d}^*_1(\Phi_{\mathscr{A}}(f),\Phi_{\mathscr{A}}(g)) := \sum_{k=1}^{\min\{|f|,|g|\}} \min_{h\in\mathbb{N}} d(\psi_f(k),\psi_g(h)).$$

Namely, with the ∞-distance, the distance of plans f and g corresponds to the maximum, with respect to time-step k, of the distance between the corresponding configurations at k. With the 1-distance, this distance corresponds to the sum, with respect to time-step k, of the distances between the corresponding configurations at k. With the max-min distance (respectively, the sum-min distance), this distance corresponds to the maximum (respectively, the sum) with respect to k, of the distance of the configuration that plan f reaches at step k with respect to the set of all configurations encountered by plan g. It is easy to see that, for each couple of plans f, g, the distance obtained from the 1-distance is the largest of the four, while the distance obtained from the max-min distance is the smallest. After having defined these distances, we can introduce an interesting variant of the optimization MAPF problem (4.2), namely, the *optimization MAPF problem constrained to a given plan*. This problem is faced when we have a sub-optimal solution f_0 of a MAPF instance, and we want to find another solution of the same problem which is not too far from f_0 and has better quality, i.e.,

shorter length. Given $\mathscr{A}^s$ and $\mathscr{A}^t$, initial and final valid configurations on a digraph G, given $f_0 \in \mathscr{E}^*_{\mathscr{A}^s}$ such that $\mathscr{A}^t = \rho(\mathscr{A}^s, f_0)$ (i.e., f_0 is a feasible solution of the MAPF instance), and given $r \in \mathbb{N}$ and a distance d between plans, the optimization MAPF problem with *Makespan* constrained to f_0 is defined as

$$\begin{aligned} &\min |f| \\ &\text{s.t. } \mathscr{A}^t = \rho(\mathscr{A}^s, f) \\ &f \in \mathscr{E}^*_{\mathscr{A}^s},\ d(f, f_0) \leq r. \end{aligned} \tag{4.3}$$

4.1.3 Domain reduction of Problems (4.2) and (4.3)

In Problems (4.2) and (4.3), variable f belongs to the set of well-defined plans $\mathscr{E}^*_{\mathscr{A}^s}$. In order to reduce the cardinality of the feasible set of these two problems, we leverage some invariance properties. Namely, we define two equivalence relations on the set of plans $\mathscr{E}^*_{\mathscr{A}^s}$ such that the objective function of Problems (4.2) and (4.3) has the same value for all plans on the same equivalence class. Further, a plan is feasible if and only if all plans of the same equivalence class are feasible. In this way, we can convert Problems (4.2) and (4.3) into equivalent problems that have the set of equivalence classes as the optimization domain. Note that the set of equivalence classes corresponds to the states set that we will use in the dynamic programming solution algorithm. We will consider the following two equivalence relations on $\mathscr{E}^*_{\mathscr{A}}$.

Definition 4.1.1. *Let $f_0 \in \mathscr{E}^*_{\mathscr{A}}$ be a reference plan. Given $f, g \in \mathscr{E}^*_{\mathscr{A}}$, then*

1. *$f \sim_1 g$ if and only if*
   - *(a) $|f| = |g|$;*
   - *(b) $\rho(\mathscr{A}, f) = \rho(\mathscr{A}, g)$.*
2. *$f \sim_2 g$ if and only if*

(a) $f \sim_1 g$;

(b) $d(f, f_0) = d(g, f_0)$.

We denote by $\tilde{\mathscr{E}}^i_{\mathscr{A}}$ the set of all equivalence classes of $\sim_i$ on $\mathscr{E}^*_{\mathscr{A}}$. Let $\hat{f} \in \tilde{\mathscr{E}}^i_{\mathscr{A}}$ and $f \in \mathscr{E}^*_{\mathscr{A}}$ be a representative of the equivalence class $\hat{f}$. We define:

- the length of $\hat{f}$, $|\hat{f}| := |f|$;
- a new transition function,

$$\rho^* : \chi \times \tilde{\mathscr{E}}^i_{\mathscr{A}} \to \mathscr{C}, \qquad \rho^*(\mathscr{A}, \hat{f}) := \rho(\mathscr{A}, f).$$

- the distance from $\hat{f}_0$ (the equivalence class to which f_0 belongs), $d(\hat{f}, f_0) := d(f, f_0)$ (only if $i = 2$).

Note that $|\hat{f}|$ is well-defined, since, by definition, all elements of equivalence class $\hat{f}$ have the same length. Similarly, ρ^* and d are well-defined since, for all elements f_1, f_2 of equivalence class $\hat{f}$, $\rho(\mathscr{A}, f_1) = \rho(\mathscr{A}, f_2)$ and, for $\sim_2$, $d(f_1, f_0) = d(f_2, f_0)$.

Let $\alpha_1 : \tilde{\mathscr{E}}^1_{\mathscr{A}} \to \mathbb{N} \times \mathscr{C}$ be such that

$$\alpha_1(\hat{f}) = (|\hat{f}|, \rho^*(\mathscr{A}, \hat{f})). \tag{4.4}$$

This function is well-defined because if $f_1 \sim_1 f_2$ then $|f_1| = |f_2|$ and $\rho(\mathscr{A}, f_1) = \rho(\mathscr{A}, f_2)$. Moreover, α_1 is injective because, if $\hat{f}_1$ and $\hat{f}_2$ are such that $\alpha_1(\hat{f}_1) = \alpha_1(\hat{f}_2)$, then $|\hat{f}_1| = |\hat{f}_2|$ and $\rho^*(\mathscr{A}, \hat{f}_1) = \rho^*(\mathscr{A}, \hat{f}_2)$, and, therefore, $\hat{f}_1 = \hat{f}_2$.

Let $\alpha_2 : \tilde{\mathscr{E}}^2_{\mathscr{A}} \to \mathbb{N} \times \chi \times \mathbb{N}$ be defined as follows:

$$\alpha_2(\hat{f}) = (|\hat{f}|, \rho^*(\mathscr{A}, \hat{f}), d(\hat{f}, f_0)). \tag{4.5}$$

This function is well defined because if $f_1 \sim_2 f_2$, then $|f_1| = |f_2|$, $\rho(\mathscr{A}, f_1) = \rho(\mathscr{A}, f_2)$ and $d(f_1, f_0) = d(f_2, f_0)$. Moreover, α_2 is injective because, if $\hat{f}_1$ and $\hat{f}_2$ are such that $\alpha_1(\hat{f}_1) = \alpha_2(\hat{f}_2)$, then $|\hat{f}_1| = |\hat{f}_2|$, $\rho^*(\mathscr{A}, \hat{f}_1) = \rho^*(\mathscr{A}, \hat{f}_2)$, $d(\hat{f}_1, f_0) = d(\hat{f}_2, f_0)$, and, therefore, $\hat{f}_1 = \hat{f}_2$.

Since $f \sim_i g$, $i = 1,2$, implies that $|f| = |g|$, $\rho(A^s, f) = \rho(A^s, g)$ and $d(f, f_0) \leq r \leftrightarrow d(g, f_0) \leq r$, it turns out that problems (4.2) and (4.3) are invariant under the equivalence relations $\sim_i$. Therefore, given $\hat{f}_0 \in \tilde{\mathscr{E}}^i_{\mathscr{A}^s}$, problem (4.3) (similar for problem (4.2)) can be defined as follows over the set of equivalence classes:

$$\begin{aligned} &\min |\hat{f}| \\ &\text{s.t. } \mathscr{A}^t = \rho^*(\mathscr{A}^s, \hat{f}) \\ &\hat{f} \in \tilde{\mathscr{E}}^i_{\mathscr{A}^s},\ d(\hat{f}, f_0) \leq r. \end{aligned} \tag{4.6}$$

4.1.4 Neighborhoods

Given the distances defined in Section 4.1.2 and the definition of the equivalence classes in Section 4.1.3, we can define the neighborhood of an equivalence class and estimate its cardinality. Such estimate is needed to evaluate the time needed to explore the neighborhoods, an operation that is central in the approach proposed in this Chapter.
Given a radius $r \in \mathbb{N}$, we define the following **neighborhood** of $f_0 \in \mathscr{E}^*_{\mathscr{A}}$:

$$\mathscr{N}_r(f_0) := \{\hat{g} \in \tilde{\mathscr{E}}^i_{\mathscr{A}} :\ |\hat{g}| \leq |f_0|, d(\hat{g}, f_0) \leq r\}.$$

Here we consider the distance based on the max-min distance. However, each of the distances defined in Section 4.1.2 can be used to define the neighborhood. As we will see, different neighborhoods lead to different exploration policies of the MAPF solutions. The following proposition proved in Appendix C provides an upper bound on the cardinality of $\mathscr{N}_r(f_0)$.

Proposition 4.1.1. *The neighborhood of f_0 of radius r has a polynomial cardinality with respect to the number of nodes. In particular, $\exists C = C(r) \in \mathbb{R}$ such that*

$$|\mathscr{N}_r(f_0)| \leq |f_0|^2 \left(1 + C(r+k)^r \phi^r\right),$$

where $k = |P|$ and $\phi = outdeg(G)$.

Note that, since the max-min distance is the smallest among the considered distances, the upper bound provided in the proposition is also valid for the distances based on the other distances previously discussed.

4.2 Iterative Neighborhood Search

In this section we describe the proposed iterative approach to detect a sub-optimal solution of Problem (4.2), or the equivalent counterpart of this problem defined over the set of equivalence classes. The returned solution is locally optimal with respect to the neighborhood of a reference solution. At each iteration, we solve an instance of Problem (4.6). More in detail, the algorithm takes as input a feasible solution f_0, that may be of poor quality. For instance, we can obtain f_0 from a rule-based algorithm, such as diSC (see Section 2.3.5) . We aim at improving f_0, obtaining a shorter solution. To this end, we solve Problem (4.6) with a dynamic programming algorithm. Namely, in neighborhood $\mathscr{N}_r(f_0)$ we search for plans shorter than f_0 through algorithm *DynProg*, that we will describe below. If we cannot obtain a solution shorter than f_0 (that is, f_0 is locally optimal) we stop the algorithm. Otherwise, if we obtain an improved solution f^*, we redefine the reference solution as $f_0 = f^*$ and solve again Problem (4.6). We iterate this procedure until we cannot shorten the current solution any further.

This algorithm can be classified as a **Neighborhood Search** algorithm (see [73]).

To define the neighborhood $N_r(f_0)$, we can use any distance function among those presented in Section 4.1.2. In our numerical experiments, we used the sum-min distance. Algorithm 1 presents the steps of the procedure.

In Algorithm 1, r is the local search radius, $\mathscr{A}^s$ is the initial configuration, and A^t is the final one.

4.3 Dynamic Programming Algorithm

To search for the optimal solution of Problem (4.6), we employ a *Dynamic Programming* (DP) algorithm. In generic DP problems we are given a state space S where

Algorithm 1 Neighborhood Search

Input: $f_0, r, \mathscr{A}^s, \mathscr{A}^t$
Output: f^*
$f^* \leftarrow f_0$
do
  $f_0 \leftarrow f^*$
  $f^* \leftarrow DynProg(f_0, r, \mathscr{A}^s, \mathscr{A}^t)$
while $|f^*| < |f_0|$
return f^*

$A \subset S$ are the target states, an expansion function $g : S \to P(S)$, where P is the power set of S, and an objective function $c : S \to R$. Starting from an initial state $s_0 \in S$, we iteratively expand states with function g to explore the state space and compute $s_t = g^n(s_0) \in A$ with the minimal objective function. In our case, the states represent the equivalence classes of relation $\sim_2$, defined in Section 4.1.3. Namely, we use injection $\alpha_2 : \tilde{\mathscr{E}}^2_{\mathscr{A}} \to \mathbb{N} \times \chi \times \mathbb{N}$ to associate with each equivalence class $\hat{f}$ a triple $(\beta, \gamma, \sigma) = \alpha_2(\hat{f})$, where β is the length of $\hat{f}$, γ is the configuration obtained by applying a plan representative of $\hat{f}$ to the initial state $\mathscr{A}^s$, and σ is the distance of a representative of $\hat{f}$ from reference plan f_0. Namely, the state space is

$$\mathscr{S} := \alpha_2(\tilde{\mathscr{E}}^2_{\mathscr{A}}) \subset \mathbb{N} \times \chi \times \mathbb{N},$$

where α_2 is defined in (4.5). Since α_2 is injective, $\mathscr{S}$ and $\tilde{\mathscr{E}}^2_{\mathscr{A}}$ are in one-to-one correspondence. Each state $s = (\beta, \gamma, \sigma) \in \mathscr{S}$, represents the equivalence class:

$$\alpha_2^{-1}(s) = \{f \in \mathscr{E}_{\mathscr{A}} : \beta = |f|, \gamma = \rho(\mathscr{A}^s, f), \sigma = d(f, f_0)\}.$$

The initial state is $s_0 = \alpha_2(\varepsilon) = (0, \mathscr{A}^s, 0)$. We use a priority queue Q to store the states that have not been visited yet. At the beginning, $Q = \{s_0\}$. We define a partial ordering on $\mathscr{S}$ based on length. Namely, if $s_1 = (\beta_1, \gamma_1, \sigma_1), s_2 = (\beta_2, \gamma_2, \sigma_2)$, $s_1 < s_2$ if $\beta_1 < \beta_2$. We order the elements of Q according to this partial ordering.

A state $s_1 = (\beta_1, \gamma_1, \sigma_1)$ *dominates* $s_2 = (\beta_2, \gamma_2, \sigma_2)$ if

- $\beta_1 \leq \beta_2$,

- $\gamma_1 = \gamma_2$,
- $\sigma_1 \leq \sigma_2$.

In other words, s_1 dominates s_2 if the plans f_1, f_2, corresponding to s_1 and s_2, satisfy the following properties. Plan f_1 is not longer than f_2, f_1 and f_2 lead to the same final configuration, and the distance of f_1 from the reference solution f_0 is not larger than the one of f_2. If s_1 dominates s_2, we can discard s_2. In general, we remove from Q all dominated states. We also define the following *transition function*, which allows to (possibly) add new states to the priority queue:

$$\tilde{\rho} : \mathscr{S} \times \mathscr{E} \to \mathscr{S}$$

$$\tilde{\rho}((\beta,\gamma,\sigma),e) := (\beta+1, \rho(\gamma,e), \sigma + \min_{k\in\mathbb{N}} d(\rho(\gamma,e), \psi_{f_0}(k))).$$

Applying this function on a state $s = (\beta,\gamma,\sigma)$:

- adds 1 to the length of the class $\alpha_2^{-1}(s)$;
- updates the final configuration of the equivalence class, applying the function $\rho(\gamma,e)$ to the final configuration of $\alpha_2^{-1}(s)$ with e being the chosen set of edges;
- updates σ, adding the computed minimum distance between the updated final configuration $\rho(\gamma,e)$ and the reference plan f_0.

We define $\Sigma := \{\tilde{\rho}((\beta,\gamma,\sigma),e) \ : \ e \in \mathscr{E} \text{ and } \rho(\gamma,e) \in \mathscr{B}_r(\gamma)\} \subset \mathscr{S}$, the set of new states which can be generated through the transition function $\tilde{\rho}$ applied to the current state (β,γ,σ) and all possible actions in $\mathscr{E}$ leading to configurations in $\mathscr{B}_r(\gamma)$. Moreover, we denote with $\Gamma := \alpha_2(\mathscr{N}_r(f_0)) \subset \mathscr{S}$, the set of states that can be visited during a neighborhood search.

4.3.1 Algorithm

The Dynamic Programming algorithm is described in Algorithm 2. The priority queue Q maintained inside the algorithm is a set of states, ordered by the length β of their representatives. Function `insert`(Q,x) inserts a state x maintaining the partial order

of Q, in the sense that, after the insertion of x, all the elements of the queue still respect the partial ordering previously defined. Function $\texttt{remove}(Q,x)$ removes x from Q. The head of the queue, that is the state with minimal β, is denoted by $Q[0]$. The algorithm explores the state space starting from the initial state s_0. At each iteration, the state with minimum β is extracted from the queue. If the state extracted is the target state (that is, if $\gamma = \mathscr{A}^t$) the algorithm stops and we return a representative of the optimal solution of Problem (4.6) (for a given equivalence class $\hat{f}$, the function $\texttt{repr}(\hat{f})$ returns a representative of the class). Otherwise, the algorithm employs function $\texttt{expand}(s, f_0, r)$, based on the transition function previously defined, to find new states. If a new state is not dominated, then it is added to the queue Q. Moreover, all states in Q dominated by the newly added state are removed from Q. Note that for more complicated solutions, the algorithm is far more effective.

Theorem 4.3.1. *Algorithm 1 and 2 have polynomial time complexity with respect to the number of nodes of the graph.*

Proof. In Algorithm 2, the time complexity is $O(|Q|^2 \cdot |\Sigma|)$. Sets Q and Σ vary with each iteration, but always remain subsets of Γ. So at each iteration the following upper bound for their cardinality always holds: $|Q| \leq |\Gamma|$, $|\Sigma| \leq |\Gamma|$. Reminding that α_2 is injective and using the result of Proposition 4.1.1, $|\Gamma| = |\mathscr{N}_r(f_0)| \leq |f_0|^2 (1 + C(1+k)^r \phi^r)$, where $k = |P|$ and $\phi = outdeg(G)$. Therefore, the time complexity of Algorithm 2 is $O(|f_0|^6 (1 + C(r+k)^r \phi^r)^3)$. Algorithm 1 recalls at most $|f_0|$ times Algorithm 2, and so it has time complexity $O(|f_0|^7 (1 + C(r+k)^r \phi^r)^3)$.

4.4 Experimental results

We performed two sets of experiments on different graphs and with initial solutions generated in two distinct ways. In both experiments, we used the Neighborhood Search of Algorithm 1, with the *Dynamic Programming* of Algorithm 2, to improve the given initial solutions. The algorithms have been coded with the *C++* programming language, and has been run on a *11th Gen Intel(R) Core(TM) i7-1165G7 @ 2.80GHz* processor with a 16 GB RAM.

Algorithm 2 Dynamic Programming with Dominance

Input: $f_0, r, \mathscr{A}^s, \mathscr{A}^t$
Output: f
$s_0 \leftarrow (0, \mathscr{A}^s, 0)$
`insert`(Q, s_0)
while $Q \neq \emptyset$ **do**:
  $s = (\beta, \gamma, \sigma) \leftarrow Q[0]$
  if $\gamma = \mathscr{A}_t$ **then**
    $f \leftarrow$ `repr`$(\alpha_2^{-1}(s))$
    $Q \leftarrow \emptyset$
  else
    $\Sigma \leftarrow$ `expand`(s, f_0, r)
    for $s_k \in \Sigma$ **do**
      if s_k is not dominated in Q **then**
        `insert`(Q, s_k)
        for $s_i \in Q$ **do**
          if s_k dominates s_i **then**
            `remove`(Q, s_i)
          end if
        end for
      end if
    end for
  end if
end while

return f

4.4.1 Random Graphs with sequentially generated initial solution

In the first set of experiments we generated random directed graphs with a number of nodes $|V|$ ranging from 20 to 100 by 10, and a number of edges $|E|$ equal to $4|V|$. The graphs are generated by creating $|E|$ random ordered pair of nodes and using them to build a directed graph. Only strongly connected graphs are selected. The number of agents $|P|$ ranges from 2 to 10, while $\mathscr{A}^s$ and $\mathscr{A}^t$ are randomly generated. To generate the initial solutions, each agent is brought to its target node one at a time, following the shortest path, in terms of crossed edges, from its source to its target in the graph obtained by removing the nodes currently occupied by all other agents (either their target or their source, depending on whether they have been already moved or not). Note that such procedure is not complete, i.e., it does not guarantee to find a feasible solution when it exists or to establish that no such feasible solution exists. We generated 100 random graphs, for which the described procedure was able to return a feasible solution, for every combination of number of nodes and number of agents. After some tuning, we set the radius r of the neighborhood equal to 5, which turned out to be a good compromise between the quality of the solutions found in the neighborhood and the time needed to explore the neighborhood (note that such time increases exponentially with r). Given the initial solution f_0 and the final one f^* returned by the proposed approach, the percentage decrease of the final solution w.r.t. the initial solution is equal to $100\frac{f_0-f^*}{f_0}\%$. In Figures 4.1 and 4.2 we report the median of the average percentage decrease and of the running time (in seconds), respectively, for every combination of $|P|$ and $|V|$. It is worthwhile to remark that the percentage decrease tends to be lower as the number of agents increases. A tentative explanation is that for cases with a greater number of agents, when the sequential procedure to generate an initial solution is able to return a feasible solution, such solution is already a good one which cannot be largely improved. This phenomenon is not observed in the second set of experiments, where a different procedure to generate an initial feasible solution is employed. Note that the average percentage decrease is lower when the number of agents becomes higher. This can be explained by the nature of the algorithm employed in finding the initial solution. Increasing the number of agents without increasing the number of nodes gives us way less feasible instances and the

solutions found are more difficult to improve.

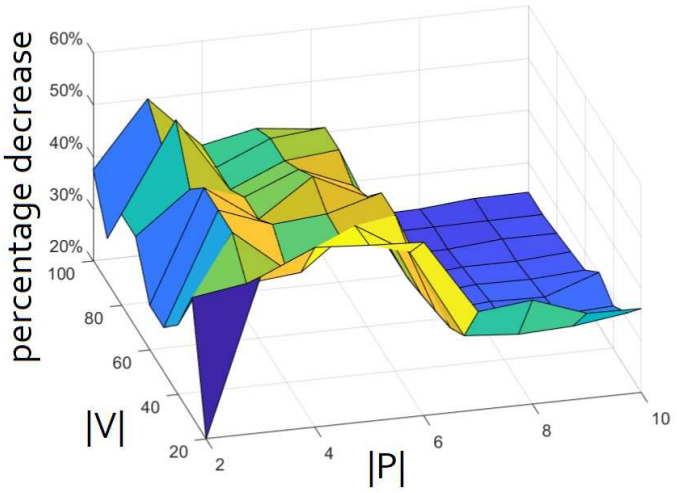


Figure 4.1: Average Percentage Decrease per $|P|$ and $|V|$.

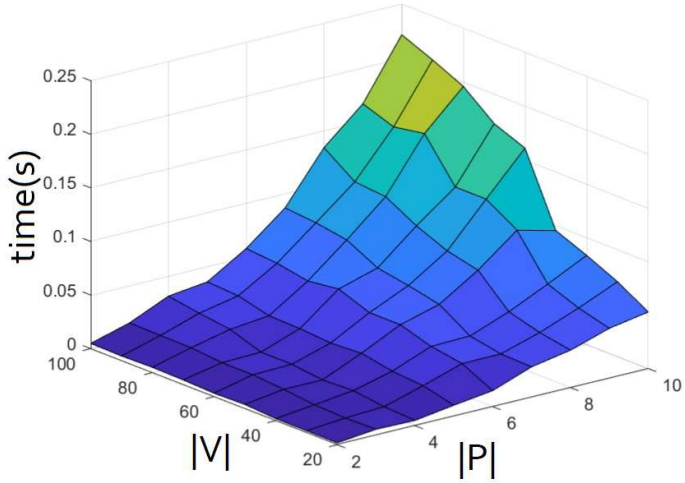


Figure 4.2: Average Running Time per $|P|$ and $|V|$.

4.4.2 Strongly connected with multiple biconnected components graphs and rule-based generated initial solution

In the second set of experiments, we generated strongly connected graphs with multiple biconnected components. The procedure used to generate such graphs can be found in Section 2.3.5. The number of agents $|P|$ ranges from 2 to 6, and the number of nodes vary from 20 to 50 with an increment of 10. The initial and final configuration $\mathscr{A}^s$ and $\mathscr{A}^t$ are randomly generated. In this case the initial solutions (if they exist) are generated through the diSC algorithm (see Section 2.3.5) which is complete, i.e., it always returns a feasible solution in case one exists. Such initial solutions usually have lower quality (i.e., the initial plans are usually longer) with respect to the ones used in the first set of experiments. This might be also the explanation why the percentage decrease in these experiments (see Figure 4.3) appears to be larger w.r.t the first set of experiments, and also tends to increase with the number of agents (differently from the case of random graphs). Again after some tuning, we set the radius r of the neighborhood equal to 3. Figures 4.3 and 4.4 report the median of the average percentage decrease and of the running time (in seconds), respectively, for every combination of $|P|$ and $|V|$. Note that the graphs employed in the second set of experiments appear to be more challenging with respect to the random ones. Indeed, computing times are larger and increase rapidly with the number of agents.

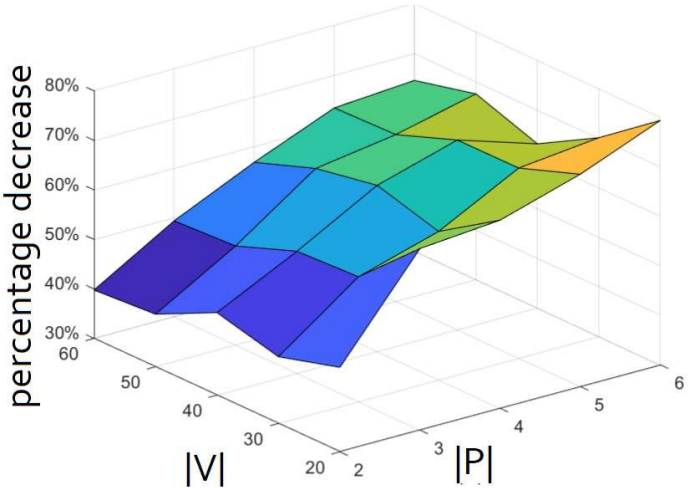


Figure 4.3: Average Percentage Decrease per $|P|$ and $|V|$.

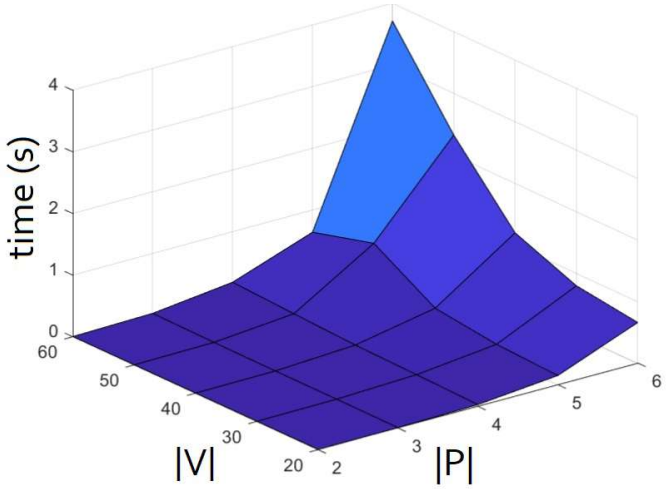


Figure 4.4: Average Running Time per $|P|$ and $|V|$.

Conclusions and Future Research

In this thesis we studied in depth four problems related to the motion planning of a fleet of AGVs:

- the problem of finding the fastest path, considering velocity and acceleration constraints, for a single agent (Chapter 1),
- the problem of finding a feasible path for each agent, preventing deadlock situations (Chapter 2),
- the problem of finding a feasible path for each agent taking into account constraints, for instance due to the size of the vehicles (Chapter 3),
- the problem of optimizing the overall solution, allowing the fleet of agents to complete their tasks in the shortest possible time (Chapter 4).

In particular, what follows is the statement of contribution of this thesis.

In Chapter 1, we addressed a variant of the Shortest Path Problem (SPP). The variant is called BASP (Bounded Acceleration SP) since speed and acceleration constraints are imposed over the arcs. Differently from SPP, where the traveling time of an arc is constant, in BASP the traveling time depends on the initial and final speed along the arc and, thus, due to speed and acceleration constraints, it also depends on the arcs preceding and following it along a path. We proved that BASP is NP-hard, but also that, under the assumption of integer data, it admits a pseudo-polynomial time algorithm. We also proposed an approximation algorithm based on the solution of an

SPP problem over an extended graph. The extended graph is defined by discretizing the admissible speeds at the nodes of the graph. Finally, we performed different computational experiments on two real-life industrial scenarios in order to evaluate the performance of the approximation algorithm, compared with the adaptive A* algorithm for k-BASP. In particular, we noted that the efficiency of the approximation algorithm depends on the discretization step h. Indeed, as h increases, the number of discretized squared speeds decreases, hence, the number of nodes and edges in the extended graph decreases as well, making Dijkstra's algorithm explore a smaller graph and run faster. Therefore, the mean computational times of the latter algorithm are better than that of the adaptive A* algorithm for k-BASP. On the other hand, as h increases, so does the relative error on the travel time. However, we observed that if we choose a good compromise for the discretization step, we obtain a mean computational time that is better than adaptive A* algorithm for k-BASP, maintaining a low mean relative error.

In Chapter 2, we studied the basic algorithm for the path planner of the new *Traffic Manager*. This must always be able to find a possible path for the agents, thus avoiding deadlock situations.

In the first part of the Chapter, we proposed two algorithms with improved length complexity for the motion planning problem and the pebble motion problem on trees. Denoting by n the number of nodes, c the maximum length of corridors and k the number of pebbles, the CATERPILLAR algorithm solves the motion planning problem with $O(nc)$ moves, while the *Leaves procedure* solves the PMT problem in $O(knc+n^2)$ moves. Moreover, we discuss a variant of the PMT problem, the PMT with trans-shipment vertices (ts-PMT), which considers a new type of vertex that cannot host pebbles. This problem is very interesting since MAPF instances on graphs can be reduced to it, and we proved that it can be solved with the *Leaves procedure for* PMT with some minor modifications.

In the second part of the Chapter, we proved that the feasibility of MAPF problems on strongly connected digraphs is decidable in linear time (Theorem 2.3.5). Moreover, we show that a MAPF problem on a strongly connected digraph is feasible if and only if the corresponding PMT problem on the biconnected component tree is feasible

(Corollary 2.3.1). Finally, we presented a complete path planner, called diSC, for solving MAPF problems on strongly connected digraphs. The idea of this algorithm is to convert MAPF to the trans-shipment variant of PMT, and then converting back the obtained solution over the general graph. An upper bound for the solution length can be derived by exploiting the complexity results of this paper.

In Chapter 3, we introduced $\mathscr{C}$-MP and $\mathscr{C}$-MAPF, generalizations of the classical MP and MAPF problems, which take into account additional constraints on the vertices of the graph. We proved that the problem of finding even a feasible solution is NP-hard for both problems. We proposed to tackle these problems by strengthening their constraints in such a way that the strenghtened problems are equivalent to classic MP and MAPF problems over a reduced graph, solvable in polynomial time. However, this method does not allow solving all $\mathscr{C}$-MP and $\mathscr{C}$-MAPF instances, since the strenghtening excludes some (possibly all) feasible solutions of the original problems. Moreover, we dealt with the problem of how to construct the reduced graph in some optimal way, which led to the $\mathscr{C}$-MIS problem. After having established NP-hardness of this problem, we have proposed some heuristics to find a solution of the problem in polynomial time. The search for further, more efficient and/or more effective, heuristics is another possible topic for future research. Simulations over a real-world warehouse show the applicability of the proposed approach in an industrial scenario. In particular, the approach is able to manage the operations of a significant number of agents over a relatively small graph.

In Chapter 4 we proposed an iterative local search procedure for MAPF, in order to shorten a known feasible solution. As already said, diSC algorithm finds a solution that has often a much larger number of steps than the shortest one. With the local search presented in this Chapter, we obtain a solution, that is still sub-optimal, but, in general, of much better quality than the initial one. The proposed algorithm has polynomial time complexity with respect to the number of agents (see Theorem 4.3.1) and has computational times compatible with industrial applications.

For each problem discussed in the thesis, we carried out simulations on warehouse graphs provided by OCME S.r.l. or on randomly generated graphs but with a structure similar to the real OCME ones. The reason for choosing these benchmarks is that all the algorithms presented will be used by the company to improve the efficiency of the Traffic Manager and the coordination of the AGVs. Until now, the new algorithms have been tested in Matlab and partly implemented in C sharp, which is used by the company, but they have not yet been implemented and tested with the Traffic Manager software.

Future Research

For each of the problems addressed in this thesis, we thought about possible future developments:

1. We addressed the problem of finding the fastest route for a single AGV, but we did not address this problem for a fleet of agents. The presence of various AGVs leads to a variant of the Multi-Agent Path Finding (MAPF) problem (see, for instance, [22]), with speed and acceleration constraints. This problem could be the focus of future research. Due to the simultaneous planning of multiple AGVS, this problem is quite different from BASP (and in general much more complex), and is outside the scope of the present work. However, note that some solution approaches for standard MAPF (that does not consider velocity or acceleration bounds), such as conflict-based search, make use of sub-procedures that involve the solution of a number of standard SPP problems. Similarly, one could guess that BASP solutions could be used as basic building block to solve MAPF with speed and acceleration bounds.

2. However, MAPF is only the "one-shot" variant of the actual problem in many applications. Typically, after an agent reaches its goal location, it does not stop and wait there forever. Instead, it is assigned a new goal location and required to keep moving. The resulting problem is referred to as *lifelong* MAPF (Ma et al. 2017), and is characterized by agents constantly being assigned new goal

locations. This problem could be the focus of future research.

3. The method presented in Chapter 3 to solve $\mathscr{C}$-MP and $\mathscr{C}$-MAPF does not allow solving all the instances, since the strenghtening excludes some (possibly all) feasible solutions of the original problems. This suggests that a possible topic for future research is to study other ways to strengthen the original problems in such a way that feasible solutions for a larger class of $\mathscr{C}$-MP and $\mathscr{C}$-MAPF instances can be detected, while preserving polynomiality.

4. Regarding the local optimization of MAPF solutions, we can extend the results presented in Chapter 4 in various respects, that will be the focus of future research:

   - We can define locality constraints different from the ones considered in Section 4.1.2. For instance, we can set a maximum on the number of agents that modify their path with respect to the reference solution. Alternatively, we can set an upper bound on the number of time intervals in which the solution departs from the reference one.
   - We can improve the complexity bound presented in Theorem 4.3.1, currently based on a quite rough bound.

Appendix A

Appendix of Chapter 1

A.1 Proof of Proposition 1.4.1

Each path P from node 0 to node $n+2$ has the following structure

$$0 \to i_1 \to i_2 \to \cdots \to i_r \to n+1 \to n+2,$$

with $i_1 < i_2 < \cdots < i_r$. Let us denote by $N_P = \{i_1, i_2, \ldots, i_r\}$ the set of intermediate nodes in P. The length of path P is $W^2 + \ell(P)$, where $\ell(P) = \sum_{i \in N_P} \beta_i$ is the length of the path up to node $n+1$.

Before proceeding with the proof we give the intuition behind it. We will show that for each path with length $\ell(P) > \frac{W}{2}$ up to node $n+1$, the traveling time from node 0 to node $n+2$ is larger than the traveling time of a path with length $\ell(P) = \frac{W}{2}$ up to node $n+1$ (if any). This will simply follow from the fact that the former path is longer and the speed along the final arc $(n+1, n+2)$ is the same in both cases. Moreover, we will show that also for a path with length $\ell(P) < \frac{W}{2}$ up to node $n+1$, the traveling time from node 0 to node $n+2$ is larger than the traveling time of a path with length $\ell(P) = \frac{W}{2}$ up to node $n+1$ (again, if any). In this case this will be proved by observing that the speed reached along this path at node $n+1$ will be lower than $\sqrt{W}$ (the speed reached by any path with length $\ell(P) \geq \frac{W}{2}$). Since it is not possible to

accelerate along arc $(n+1,n+2)$, the traveling time along this arc will be higher with respect to paths with length $\ell(P) \geq \frac{W}{2}$ up to node $n+1$. In particular, we will show that the increase of the time needed to travel along arc $(n+1,n+2)$ will be larger than the time saved along the sub-path from node 0 to node $n+1$ with respect to a path for which $\ell(P) = \frac{W}{2}$. In conclusion, we will show that the Partition problem has a `yes` answer if and only if the optimal value of the BASP instance is the traveling time of a path with length $\ell(P) = \frac{W}{2}$ up to node $n+1$. In what follows we prove the result in a formal way.

Let us first assume that $\ell(P) < \frac{W}{2}$. In this case, according to the discussion in Section 1.3, the maximum speed which can be reached at node $n+1$ is $v_u = a_{\max} t_P$, where $a_{\max} = 1$ and t_P fulfills $\ell(P) = \frac{1}{2} a_{\max} t^2$, that is, $t_P = \sqrt{2\ell(P)}$. Thus, $v_u = \sqrt{2\ell(P)} < \sqrt{W}$. Along the final arc of the path $(n+1,n+2)$ the maximum acceleration is null, while the maximum deceleration is $-\frac{1}{2W}$. Then, the optimal speed profile along this arc is obtained by keeping the speed $v_u = \sqrt{2\ell(P)}$ for a portion of the arc with length $W^2 - 2\ell(P)W$, while in the last portion, with length $2\ell(P)W$, the speed is decreased with the maximum possible deceleration $-\frac{1}{2W}$. Now we denote by $t^1_{L_1}(\ell(P))$ the time to run along the first portion of the arc, while we denote by $t_{L_2}(\ell(P))$ the time to run along the second part of the arc. We have that

$$t_{L_1}(\ell(P)) = \frac{W^2 - 2\ell(P)W}{\sqrt{2\ell(P)}},$$

while $t_{L_2}(\ell(P))$ fulfills the following condition

$$\sqrt{2\ell(P)}W = \sqrt{2\ell(P)} t_{L_2}(\ell(P)) - \frac{1}{4W} t_{L_2}(\ell(P))^2,$$

that is,

$$t_{L_2}(\ell(P)) = 2W\sqrt{2\ell(P)}.$$

Thus, the overall time to traverse such paths is

$$T_1(\ell(P)) = \sqrt{2\ell(P)} + \frac{W^2 - 2\ell(P)W}{\sqrt{2\ell(P)}} + t_{L_2}(\ell(P)) = \sqrt{2\ell(P)} + \frac{W^2}{\sqrt{2\ell(P)}} + W\sqrt{2\ell(P)}.$$

Now, let us consider paths P such that $\ell(P) \geq \frac{W}{2}$. A lower bound for the time needed to run along the path up to node $n+1$ is given again by the solution of the following simple equation $\ell(P) = \frac{1}{2}a_{\max}t^2$, which is $t_P = \sqrt{2\ell(P)}$. Note that this is a lower bound since with the maximum acceleration we would reach the speed $v_u = a_{\max}t_P = \sqrt{2\ell(P)} \geq \sqrt{W}$, so that we stop accelerating as soon as we reach the maximum speed $\sqrt{W}$. Since $\ell(P) \geq \frac{W}{2}$, we have that the lower bound can be further bounded from below by $\sqrt{W}$. Finally, we observe that such lower bound can be attained if and only if $\ell(P) = \frac{W}{2}$, that is, if and only if the Partition problem admits a solution. Now, over the last arc $(n+1,n+2)$ we can decrease the speed to 0 at node $n+2$ by keeping the maximum deceleration $-\frac{1}{2W}$ over the whole arc. Then, the time needed to traverse this arc, denoted by t_{L_3}, fulfills

$$W^2 = \sqrt{W}t_{L_3} - \frac{1}{4W}(t_{L_3})^2,$$

so that $t_{L_3} = 2W^{3/2}$. Then, a lower bound for the time needed to traverse such paths is $T_2 = \sqrt{W} + 2W^{3/2}$ and, as already pointed out, such lower bound is attained if and only if $\ell(P) = \frac{W}{2}$. Figure A.1 illustrates the optimal speed profiles for three distinct paths P_1, P_2 and P_3, fulfilling $\ell(P_1) < \frac{W}{2}$, $\ell(P_2) = \frac{W}{2}$, and $\ell(P_3) > \frac{W}{2}$, respectively. The three profiles are depicted, respectively with a dashed, a continuous and a dotted line (up to distance $\ell(P_1)$ they are overlapping). We notice that for the shortest path P_1 we are unable to reach the maximum squared speed W, so that along arc $(n+1,n+2)$ we first increase the speed as much as possible with acceleration $a^{\max}_{n+1,n+2}$ (in fact, we keep the speed constant since $a^{\max}_{n+1,n+2} = 0$), and then we decrease the speed as fast as possible with deceleration $a^{\min}_{n+1,n+2}$. For the intermediate path P_2 we reach the maximum squared speed exactly at the end of the path and then, along arc $(n+1,n+2)$ we decrease the speed as fast as possible with deceleration $a^{\min}_{n+1,n+2}$. Finally, for the longest path P_3 we reach the maximum squared speed before the end of the path, and we keep the speed constant in the last part of the path, while along arc $(n+1,n+2)$ we decrease the speed as fast as possible with deceleration $a^{\min}_{n+1,n+2}$. Now, if we are able to prove that $T_2 < T_1(\ell(P))$ when $\ell(P) \leq \frac{W}{2} - 1$, then we are done. We have that

$$T_2 - T_1(\ell(P)) = \sqrt{W} + 2W^{3/2} - W\sqrt{2\ell(P)} - \sqrt{2\ell(P)} - \frac{W^2}{\sqrt{2\ell(P)}}.$$

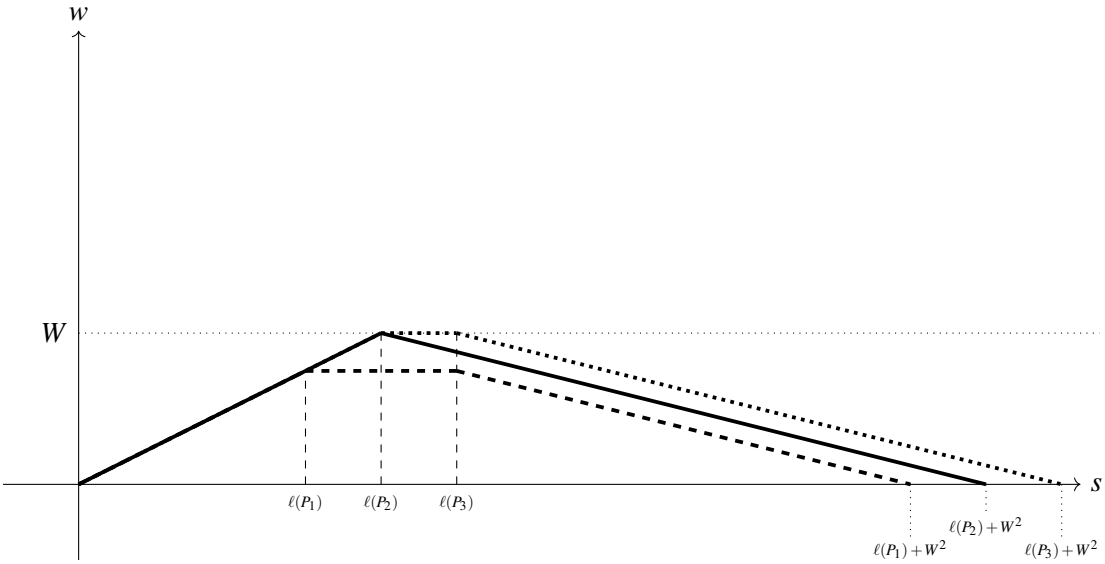


Figure A.1: Maximum (squared) speed profiles along three paths P_1, P_2, P_3 fulfilling $\ell(P_1) < \frac{W}{2}$, $\ell(P_2) = \frac{W}{2}$, and $\ell(P_3) > \frac{W}{2}$.

By the change of variable $x = \sqrt{2\ell(P)}$, this can be rewritten as

$$\frac{-(W+1)x^2 + (\sqrt{W} + 2W^{3/2})x - W^2}{x},$$

which can be easily seen to be negative for all $x \leq \frac{W^{3/2}}{W+1}$. Now, recalling the definition of x, this means that $T_2 - T_1(\ell(P))$ is negative if

$$\ell(P) < \frac{W^3}{2(W+1)^2}.$$

Then, the result follows by observing that

$$\frac{W^3}{2(W+1)^2} > \frac{W}{2} - 1.$$

A.2 Proof of Lemma 1.5.2

For $i \in \{1, \ldots, |P|\}$, set $a_i^+ = 2a^{\max}_{P(i),P(i+1)}\ell_{P(i),P(i+1)}$, $a_i^- = 2a^{\min}_{P(i),P(i+1)}\ell_{P(i),P(i+1)}$. For $i \in \{2, \ldots, |P|\}$, set $w_i^+ = \min\{w^{\max}_{P(i-1),P(i)}, w^{\max}_{P(i),P(i+1)}\}$. It is convenient to rewrite

Problem (1.9) as:

$$
\begin{aligned}
T(P) = \min_{\mathbf{w}\in\mathbb{R}^{|P|+1}} & \sum_{i=1}^{|P|} c_{P(i)P(i+1)}(w_i, w_{i+1}) \\
& 0 \le w_i \le w_i^+, && i \in \{2,\dots,|P|\}, \\
& w_{i+1} \le w_i + a_i^+, && i \in \{1,\dots,|P|\}, \\
& w_i \le w_{i+1} - a_i^-, && i \in \{1,\dots,|P|\}, \\
& w_1 = 0,\, w_{|P|+1} = 0\,.
\end{aligned}
\tag{A.1}
$$

Problem (A.1) belongs to the class of Problem 5 in [74] (nondecreasing right-hand side of the constraints and strictly monotonically decreasing objective function) . As reported there, the solution of Problem (1.9) can be found as follows. Define vectors $\mathbf{F}, \mathbf{B}, \mathbf{W} \in \mathbb{R}^{|P|+1}$ such that:

$$
\begin{aligned}
& F_1 = 0 \\
& F_i = \min\{F_{i-1} + a_{i-1}^+, w_i^+\}, \quad i \in \{2,\dots,|P|+1\}, \\
& B_{|P|+1} = 0 \\
& B_i = \min\{B_{i+1} - a_i^-, w_i^+\}, \quad i \in \{1,\dots,|P|\}, \\
& \mathbf{W} = \min\{\mathbf{F}, \mathbf{B}\}.
\end{aligned}
$$

Note that the components of vector $\mathbf{W}$ are equivalent to the values of function $W(s)$ defined in Section 1.3 evaluated at nodes of the path, while along each arc of the path function $W(s)$ is defined according to (1.2). As a consequence of Theorem 2 of [74] the solution of Problem (A.1) is $\mathbf{w}^*(P) = \mathbf{W}$. On the other hand, Problem (1.12) can be rewritten in the following form:

$$
\begin{aligned}
T^h(P) = \min_{\mathbf{w}\in\mathbb{R}^{|P|+1}} & \sum_{i=1}^{|P|} c_{P(i)P(i+1)}(w_i, w_{i+1}) \\
& 0 \le w_i \le \left\langle w_i^+ \right\rangle, && i \in \{2,\dots,|P|\}, \\
& w_{i+1} \le \left\langle w_i + a_i^+ \right\rangle, && i \in \{1,\dots,|P|\}, \\
& w_i \le \left\langle w_{i+1} - a_i^- \right\rangle, && i \in \{1,\dots,|P|\}, \\
& w_1 = 0,\, w_{|P|+1} = 0\,.
\end{aligned}
\tag{A.2}
$$

Note that, strictly speaking, this is a relaxation of Problem (1.12), since we have not included the last constraint, namely $\langle w_i \rangle = w_i, i \in \{1,\dots,|P|+1\}$. In fact, as shown in [74], at least one constraint is active at each component $w_i^h(P)$, $i \in \{1,\dots,|P|+1\}$, of the optimal solution of (A.2). Since $0 = \langle 0 \rangle$ and, for all $x \in \mathbb{R}$, $\langle \langle x \rangle \rangle = \langle x \rangle$, necessarily $\langle w_i^h(P) \rangle = w_i^h(P)$, $i \in \{1,\dots,|P|+1\}$ (which means that the optimal solution of (A.2) fulfills the last constraint in (1.12)). Also Problem (A.2) belongs to the class of Problem 5 in [74]. Hence, its solution can be computed with the same procedure used for Problem (A.1). Namely, define vectors $\mathbf{F}^h, \mathbf{B}^h, \mathbf{W}^h \in \mathbb{R}^{|P|+1}$ such that:

$$
\begin{aligned}
&F_1^h = 0 \\
&F_i^h = \left\langle \min\{F_{i-1}^h + a_{i-1}^+, w_i^+\} \right\rangle, \quad i \in \{2,\dots,|P|+1\}, \\
&B_{|P|+1}^h = 0 \\
&B_i^h = \left\langle \min\{B_{i+1}^h - a_{i+1}^-, w_i^+\} \right\rangle, \quad i \in \{1,\dots,|P|\}, \\
&\mathbf{W}^h = \min\{\mathbf{F}^h, \mathbf{B}^h\}.
\end{aligned}
$$

Again, by Theorem 2 in [74], $\mathbf{w}^h(P) = \mathbf{W}^h$. Note that, by their definitions, $\mathbf{B}^h \leq \mathbf{B}$ and $\mathbf{F}^h \leq \mathbf{F}$.

Now we are ready to prove Lemma 1.5.2. We first prove that for all $i \in \{1,\dots,|P|+1\}$:

$$F_i - F_i^h \leq h(i-1).$$

Set $\delta_i = F_i - F_i^h$. By the definition of F_i^h we may have that $F_i^h = \left\langle F_{i-1}^h + a_{i-1}^+ \right\rangle$ or $F_i^h = \left\langle w_i^+ \right\rangle$. In the first case, $\delta_i \leq F_{i-1} - F_{i-1}^h + a_{i-1}^+ - \left\langle a_{i-1}^+ \right\rangle \leq \delta_{i-1} + h$. In the second case, $\delta_i \leq w_i^+ - \left\langle w_i^+ \right\rangle \leq h$. Hence, δ satisfies

$$
\begin{aligned}
&\delta_1 = 0 \\
&\delta_{i+1} \leq \delta_i + h, \quad i \in \{1,\dots,|P|\},
\end{aligned}
$$

and it follows that $\delta_i \leq h(i-1)$. Proceeding in the same way as above, setting $\eta_i = B_i - B_i^h$, η satisfies

$$
\begin{aligned}
&\eta_{|P|+1} = 0 \\
&\eta_i \leq \eta_{i+1} + h, \quad i \in \{1,\dots,|P|\},
\end{aligned}
$$

and, consequently:

$$B_i - B_i^h \leq h(|P| + 1 - i).$$

Finally, we observe that

$$w^*(P)_i - w^h(P)_i = \min\{F_i, B_i\} - \min\{F_i^h, B_i^h\} \leq \max\{F_i - F_i^h, B_i - B_i^h\} \leq h|P|,$$

as we wanted to prove.

A.3 Proof of Lemma 1.5.4

Given an arc (i,j) and squared speeds $w,z,\hat{w},\hat{z} \in \mathbb{R}$, with $0 \leq w,z,\hat{w},\hat{z} \leq w_{ij}^{\max}$, $w \leq \hat{w}$ and $z \leq \hat{z}$, for an interval $[a,b] \subseteq [0,\ell_{ij}]$, define

$$e_{[a,b]} = \left| \int_a^b \left(\frac{1}{\sqrt{w_{ij}(s;w,z)}} - \frac{1}{\sqrt{w_{ij}(s;\hat{w},\hat{z})}} \right) ds \right|,$$

where w_{ij} is defined in (1.2). Our goal is to find an upper bound for $\left|c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z})\right| = e_{[0,\ell_{ij}]}$. We consider only the case $\hat{w} \leq \hat{z}$ (the converse case in which $\hat{w} \geq \hat{z}$ is analogous).

We find a bound on $\left|c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z})\right|$ by splitting the integration interval $[0,\ell_{ij}]$ into three parts: the initial part $[0,\bar{s}]$, with $\bar{s} \in [0,\ell_{ij}]$, in which both the speed profiles starting at $\hat{w}$ and w are growing, the second part $[\bar{s},\tilde{s}]$, with $\tilde{s} \in [\bar{s},\ell_{ij}]$ in which the speed profile starting at $\hat{w}$ keeps growing whilst the one starting at w starts decreasing, and the final part $[\tilde{s},\ell_{ij}]$, in which both speed profiles are decreasing (see Figure 1.22). In case the first profile never decreases, we set $\tilde{s} = \ell_{ij}$, and in case the second profile never decreases, we set $\tilde{s} = \bar{s} = \ell_{ij}$. In this way, $\left|c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z})\right| = e_{[0,\bar{s}]} + e_{[\bar{s},\tilde{s}]} + e_{[\tilde{s},\ell_{ij}]}$. We will need the following technical remark.

Remark A.3.1. *Given $c > 0$ and $x \in [0,c]$, it holds that*

$$-\frac{x}{\sqrt{c}} + \sqrt{c} \leq \sqrt{c-x} \leq -\frac{x}{2\sqrt{c}} + \sqrt{c}.$$

Next lemma establishes a bound from above for the error along the first part of the arc.

Lemma A.3.1. *Given arc (i,j) and $0 \leq w, \hat{w}, z, \hat{z}$, with $w \leq \hat{w}$ and $z \leq \hat{z}$ the following bound holds:*

$$\begin{cases} e_{[0,\bar{s}]} = 0 & \text{if } \hat{w} = 0, \\ e_{[0,\bar{s}]} \leq \frac{|w-\hat{w}|}{a_{ij}^{\max}\sqrt{\hat{w}}} & \text{otherwise}. \end{cases}$$

Proof. If $\hat{w} = 0$, then also $w = 0$, hence

$$e_{[0,\bar{s}]} = \int_0^{\bar{s}} \left(\frac{1}{\sqrt{w + 2a_{ij}^{\max} s}} - \frac{1}{\sqrt{\hat{w} + 2a_{ij}^{\max} s}} \right) ds = \int_0^{\bar{s}} \left(\frac{1}{\sqrt{2a_{ij}^{\max} s}} - \frac{1}{\sqrt{2a_{ij}^{\max} s}} \right) ds = 0.$$

Otherwise, if $\hat{w} > 0$, we can bound the error over $[0, \bar{s}]$ as follows

$$\begin{aligned} e_{[0,\bar{s}]} &= \int_0^{\bar{s}} \left(\frac{1}{\sqrt{w + 2a_{ij}^{\max} s}} - \frac{1}{\sqrt{\hat{w} + 2a_{ij}^{\max} s}} \right) ds = \left[\frac{\sqrt{w + 2a_{ij}^{\max} s}}{a_{ij}^{\max}} - \frac{\sqrt{\hat{w} + 2a_{ij}^{\max} s}}{a_{ij}^{\max}} \right]_0^{\bar{s}} = \\ &= \frac{1}{a_{ij}^{\max}} \left(\sqrt{w + 2a_{ij}^{\max} \bar{s}} - \sqrt{w} - \sqrt{\hat{w} + 2a_{ij}^{\max} \bar{s}} - \sqrt{\hat{w}} \right) \leq \\ &\leq \frac{1}{a_{ij}^{\max}} \left(-\frac{|w-\hat{w}|}{2\sqrt{\hat{w} + 2a_{ij}^{\max} \bar{s}}} + \sqrt{\hat{w} + 2a_{ij}^{\max} \bar{s}} + \frac{|w-\hat{w}|}{\sqrt{\hat{w}}} - \sqrt{\hat{w}} - \sqrt{\hat{w} + 2a_{ij}^{\max} \bar{s}} + \sqrt{\hat{w}} \right) = \\ &= \left(\frac{1}{\sqrt{\hat{w}}} - \frac{1}{2\sqrt{\hat{w} + 2a_{ij}^{\max} \bar{s}}} \right) \frac{|w-\hat{w}|}{a_{ij}^{\max}} \leq \frac{|w-\hat{w}|}{a_{ij}^{\max}\sqrt{\hat{w}}}, \end{aligned}$$

where, in the first inequality, we used the fact that, for $|w - \hat{w}| \leq \hat{w} + 2a_{ij}^{\max} \bar{s}$, in view of Remark A.3.1 it holds that:

$$\sqrt{w + 2a_{ij}^{\max} \bar{s}} = \sqrt{\hat{w} - |w-\hat{w}| + 2a_{ij}^{\max} \bar{s}} \leq -\frac{|w-\hat{w}|}{2\sqrt{\hat{w} + 2a_{ij}^{\max} \bar{s}}} + \sqrt{\hat{w} + 2a_{ij}^{\max} \bar{s}},$$

and that, for $|w - \hat{w}| \leq \hat{w}$,

$$\sqrt{w} = \sqrt{\hat{w} - |w-\hat{w}|} \geq -\frac{|w-\hat{w}|}{\sqrt{\hat{w}}} + \sqrt{\hat{w}}.$$

□

The error over the second part $[\bar{s},\tilde{s}]$ of arc (i,j) can be bounded from above as follows.

Lemma A.3.2. *Given arc (i,j) and $0 \le w,\hat{w},z,\hat{z}$, with $w \le \hat{w}$, $z \le \hat{z}$, the following bound holds:*

$$e_{[\bar{s},\tilde{s}]} \le \max\{|w-\hat{w}|,|z-\hat{z}|\}\frac{2}{|a_{ij}^{\min}|\sqrt{\tilde{z}}},$$

where $\tilde{z} = w_{ij}(\tilde{s};\hat{w},\hat{z})$ is the squared speed at $\tilde{s}$ for the profile with boundary speeds $\hat{w},\hat{z}$.

Proof. We can bound from above the error over $[\bar{s},\tilde{s}]$ as follows

$$\begin{aligned}
e_{[\bar{s},\tilde{s}]} &= \int_{\bar{s}}^{\tilde{s}} \left(\frac{1}{\sqrt{\tilde{z}-|z-\hat{z}|+2a_{ij}^{\min}(s-\tilde{s})}} - \frac{1}{\sqrt{\hat{w}+2a_{ij}^{\max}s}} \right) ds \le \int_{\bar{s}}^{\tilde{s}} \frac{ds}{\sqrt{\tilde{z}-|z-\hat{z}|+2a_{ij}^{\min}(s-\tilde{s})}} = \\
&= \left[\frac{\sqrt{\tilde{z}-|z-\hat{z}|+2a_{ij}^{\min}(s-\tilde{s})}}{a_{ij}^{\min}} \right]_{\bar{s}}^{\tilde{s}} = \frac{1}{|a_{ij}^{\min}|} \left(\sqrt{\tilde{z}-|z-\hat{z}|+2a_{ij}^{\min}(\bar{s}-\tilde{s})} - \sqrt{\tilde{z}-|z-\hat{z}|} \right) \le \\
&\le \frac{1}{|a_{ij}^{\min}|} \left(-\frac{|z-\hat{z}|}{2\sqrt{\tilde{z}+2a_{ij}^{\min}(\bar{s}-\tilde{s})}} + \sqrt{\tilde{z}+2a_{ij}^{\min}(\bar{s}-\tilde{s})} + \frac{|z-\hat{z}|}{\sqrt{\tilde{z}}} - \sqrt{\tilde{z}} \right) \le \\
&\le \frac{1}{|a_{ij}^{\min}|} \left(\sqrt{\tilde{z}+2a_{ij}^{\min}(\bar{s}-\tilde{s})} + \frac{|z-\hat{z}|}{\sqrt{\tilde{z}}} - \sqrt{\tilde{z}} \right) \le \\
&\le \frac{1}{|a_{ij}^{\min}|} \left(\sqrt{\tilde{z}+\max\{|w-\hat{w}|,|z-\hat{z}|\}} + \frac{|z-\hat{z}|}{\sqrt{\tilde{z}}} - \sqrt{\tilde{z}} \right) \le \\
&\le \frac{1}{|a_{ij}^{\min}|} \left(\frac{\max\{|w-\hat{w}|,|z-\hat{z}|\}}{2\sqrt{\tilde{z}}} + \sqrt{\tilde{z}} + \frac{|z-\hat{z}|}{\sqrt{\tilde{z}}} - \sqrt{\tilde{z}} \right) \le \max\{|w-\hat{w}|,|z-\hat{z}|\}\frac{2}{|a_{ij}^{\min}|\sqrt{\tilde{z}}},
\end{aligned}$$

where, in the second inequality, we used the fact that, for $|z-\hat{z}| \le \tilde{z}+2a_{ij}^{\min}(\bar{s}-\tilde{s})$, in view of Remark A.3.1, it holds that

$$\sqrt{\tilde{z}-|z-\hat{z}|+2a_{ij}^{\min}(\bar{s}-\tilde{s})} \le -\frac{|z-\hat{z}|}{2\sqrt{\tilde{z}+2a_{ij}^{\min}(\bar{s}-\tilde{s})}} + \sqrt{\tilde{z}+2a_{ij}^{\min}(\bar{s}-\tilde{s})},$$

and that, for $|z-\hat{z}| \le \tilde{z}$,

$$\sqrt{\tilde{z}-|z-\hat{z}|} \ge -\frac{|z-\hat{z}|}{\sqrt{\tilde{z}}} + \sqrt{\tilde{z}},$$

whilst, in the fourth inequality, we used the fact that $2a_{ij}^{\min}(\bar{s}-\tilde{s}) \leq \max\{|w-\hat{w}|, |z-\hat{z}|\}$. Indeed, the following equations hold (see also Figure 1.22)

$$\begin{aligned} \hat{z}+2a_{ij}^{\min}(\tilde{s}-\ell_{ij}) &= \hat{w}+2a_{ij}^{\max}\tilde{s} \\ z+2a_{ij}^{\min}(\bar{s}-\ell_{ij}) &= w+2a_{ij}^{\max}\bar{s}. \end{aligned}$$

Taking the difference between the second and the first equation, we end up with

$$2a_{ij}^{\min}(\bar{s}-\tilde{s}) = |z-\hat{z}|-|w-\hat{w}|+2a_{ij}^{\max}(\bar{s}-\tilde{s}) \leq |z-\hat{z}|-|w-\hat{w}| \leq \max\{|w-\hat{w}|, |z-\hat{z}|\},$$

where the first inequality comes from $a_{ij}^{\max} > 0$ and $\bar{s} \leq \tilde{s}$. □

Finally, we establish a bound for the error along the third part of the arc.

Lemma A.3.3. *Given arc (i,j) and $0 \leq w, \hat{w}, z, \hat{z}$, with $w \leq \hat{w}$, $z \leq \hat{z}$, the following bound holds:*

$$\begin{cases} e_{[\bar{s},\ell_{ij}]} = 0 & \text{if } \hat{z} = 0, \\ e_{[\bar{s},\ell_{ij}]} \leq \frac{|z-\hat{z}|}{|a_{ij}^{\min}|\sqrt{\hat{z}}} & \text{otherwise}. \end{cases}$$

Proof. The proof of Lemma A.3.3 is analogous to that of Lemma A.3.1. □

Now, we can prove Lemma 1.5.4, that is, we can provide an estimate on the absolute error over the entire arc (i,j).

Proof. By Lemmas A.3.1, A.3.2 and A.3.3, we have the following cases. If $\hat{w} = \hat{z} = 0$, then

$$c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z}) = 0$$

(note that in this case $\bar{s} = \tilde{s}$, so that $e_{[\bar{s},\tilde{s}]} = 0$). If $\hat{w} = 0$, then:

$$c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z}) = e_{[\bar{s},\tilde{s}]} + e_{[\bar{s},\ell_{ij}]} \leq |z-\hat{z}|\frac{2}{|a_{ij}^{\min}|\sqrt{\tilde{z}}} + \frac{|z-\hat{z}|}{|a_{ij}^{\min}|\sqrt{\hat{z}}} \leq |z-\hat{z}|\frac{3}{|a_{ij}^{\min}|\sqrt{\hat{z}}},$$

where the last inequality follows from $\tilde{z} \geq \hat{z}$. Similarly, for $\hat{z} = 0$:

$$\begin{gathered} c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z}) = e_{[0,\bar{s}]} + e_{[\bar{s},\tilde{s}]} \leq \\ \leq \frac{|w-\hat{w}|}{a_{ij}^{\max}\sqrt{\hat{w}}} + |w-\hat{w}|\frac{2}{|a_{ij}^{\min}|\sqrt{\tilde{z}}} \leq |w-\hat{w}|\frac{3}{\min\{a_{ij}^{\max}, -a_{ij}^{\min}\}\sqrt{\hat{w}}} \end{gathered}$$

where the last inequality follows from $\tilde{z} \geq \hat{w}$. Finally, if $\hat{w}, \hat{z} > 0$:

$$c_{ij}(w,z) - c_{ij}(\hat{w},\hat{z}) = e_{[0,\bar{s}]} + e_{[\bar{s},\bar{s}]} + e_{[\bar{s},\ell_{ij}]} \leq$$
$$\leq \frac{|w-\hat{w}|}{a_{ij}^{\max}\sqrt{\hat{w}}} + \max\{|w-\hat{w}|,|z-\hat{z}|\}\frac{2}{|a_{ij}^{\min}|\sqrt{\tilde{z}}} + \frac{|z-\hat{z}|}{|a_{ij}^{\min}|\sqrt{\hat{z}}} \leq$$
$$\leq \frac{|w-\hat{w}| + 2\max\{|w-\hat{w}|,|z-\hat{z}|\} + |z-\hat{z}|}{\min\{a_{ij}^{\max}, -a_{ij}^{\min}\}\sqrt{\min\{\hat{w},\hat{z}\}}} = \frac{4\max\{|w-\hat{w}|,|z-\hat{z}|\}}{\min\{a_{ij}^{\max}, -a_{ij}^{\min}\}\sqrt{\min\{\hat{w},\hat{z}\}}}.$$

Now the results immediately follows from the fact that, by the definitions of $a_{\min}$ and $\bar{w}$, it holds that $a_{ij}^{\max}, |a_{ij}^{\min}| \geq a_{\min}$, $\hat{w} \geq \bar{w}$ if $\hat{w} > 0$, and $\hat{z} \geq \bar{w}$ if $\hat{z} > 0$. □

A.4 Proof of Proposition 1.6.2

Proposition A.4.1. *Let $\mu, \alpha : [0,+\infty) \to \mathbb{R}^+$, for $i \in \{1,2\}$, let F_i be the function defined in* (1.5) *where F_i replaces F and $w_{0,i}$ replaces $\mu(0)$, with $0 \leq w_{0,i} \leq \mu(0)$; and let $\bar{\lambda}$ be such that $\mu(\bar{\lambda}) = \int_0^{\bar{\lambda}} \alpha(\lambda)d\lambda$. Then $(\forall \lambda \geq \bar{\lambda})\ F_1(\lambda) = F_2(\lambda)$.*

Proof. W.l.o.g., assume that $w_{0,1} \geq w_{0,2}$. This implies that $(\forall \lambda \geq 0)\ F_1(\lambda) \geq F_2(\lambda)$. Indeed, assume by contradiction that there exists $\bar{\lambda}$ such that $F_1(\bar{\lambda}) < F_2(\bar{\lambda})$, then, by continuity of F_1 and F_2, this implies that there exists $\hat{\lambda} \leq \bar{\lambda}$ such that $F_1(\hat{\lambda}) = F_2(\hat{\lambda})$, thus $(\forall \lambda \geq \hat{\lambda})\ F_1(\lambda) = F_2(\lambda)$, since, for $\lambda \geq \hat{\lambda}$, $F_1(\lambda)$ and $F_2(\lambda)$ solve the same differential equation with the same initial condition at $\lambda = \hat{\lambda}$, contradicting the assumption. Further, note that $(\exists \tilde{\lambda} \in (0,\bar{\lambda}])\ F_2(\tilde{\lambda}) = \mu(\tilde{\lambda})$. Indeed, if by contradiction $(\forall \lambda \in (0,\bar{\lambda}])\ F_2(\lambda) < \mu(\lambda)$, then $(\forall \lambda \in (0,\bar{\lambda}])\ F_2'(\lambda) = \alpha(\lambda)$, so that $F_2(\bar{\lambda}) - F_2(0) = \int_0^{\bar{\lambda}} \alpha(\lambda)\ d\lambda = \mu(\bar{\lambda})$, which contradicts the assumption. Hence, $(\exists \hat{\lambda} \in \mathbb{R}^+)\ F_2(\hat{\lambda}) = F_1(\hat{\lambda}) = \mu(\hat{\lambda})$ and, consequently, $(\forall \lambda \geq \hat{\lambda})\ F_1(\lambda) = F_2(\lambda)$, which implies the thesis, being $\bar{\lambda} \geq \hat{\lambda}$. □

For $p \in \mathscr{P}$, $\lambda \in [0,\ell(p)]$, we set $\mathscr{W}_p(\lambda) = w^*(p)$, the square of the optimal speed profile for traversing path p, evaluated at arc-length λ, with respect to p.

Proposition A.4.2. 1) *Let $p_1, p_2, q \in \mathscr{P}$, be such that $p_1q, p_2q \in \mathscr{P}$, then $(\forall \lambda \geq \ell^+(q))\ \mathscr{W}_{p_1q}(\ell(p_1)+\lambda) = \mathscr{W}_{p_2q}(\ell(p_2)+\lambda)$.*

2) *Let* $p, q_2, q_1 \in \mathscr{P}$*, be such that* $pq_1, pq_2 \in \mathscr{P}$*, then* $(\forall \lambda \leq \ell^-(p))\ \mathscr{W}_{pq_1}(\lambda) = \mathscr{W}_{pq_2}(\lambda)$.

Proof. We only prove 1), the proof of 2) is analogous. Note that, for $\lambda \geq 0$, $\mathscr{W}_{p_1q}(\lambda + \ell(p_1)) = \min\{F_1(\lambda), B(\lambda)\}$, $\mathscr{W}_{p_2q}(\lambda + \ell(p_2)) = \min\{F_2(\lambda), B(\lambda)\}$, where F_1, F_2 are the be the function defined in (1.5) with initial conditions $w_{0,1} = \mathscr{W}_{p_1}(\ell(p_1))$ and $w_{0,2} = \mathscr{W}_{p_2}(\ell(p_2))$, respectively, and B is be the function defined in (1.6). By Proposition A.4.1, for $\lambda \geq \ell^+(q)$, $F_1(\lambda) = F_2(\lambda)$. Consequently, $(\forall \lambda \geq \ell^+(q))\ \mathscr{W}_{p_1q}(\ell(p_1) + \lambda) = \mathscr{W}_{p_2q}(\ell(p_2) + \lambda)$. □

Proof of Proposition 1.6.2.
We have that $T(p_1t\sigma) - T(p_1t) = \int_0^{\ell(p_1t\sigma)} (\mathscr{W}_{p_1t\sigma}(\lambda))^{-\frac{1}{2}} d\lambda - \int_0^{\ell(p_1t)} (\mathscr{W}_{p_1t}(\lambda))^{-\frac{1}{2}} d\lambda = \int_{\ell(p_1)+\ell^-(t)}^{\ell(p_1t\sigma)} (\mathscr{W}_{p_1t\sigma}(\lambda))^{-\frac{1}{2}} d\lambda - \int_{\ell(p_1)+\ell^-(t))}^{\ell(p_1t)} (\mathscr{W}_{p_1t}(\lambda))^{-\frac{1}{2}} d\lambda$, where we used that, by ii) of Proposition A.4.2, $(\forall \lambda \leq \ell(p_1) + \ell^-(t)) \mathscr{W}_{p_1t\sigma}(\lambda) = \mathscr{W}_{p_1t}(\lambda)$. Similarly, $T(p_2t\sigma) - T(p_2t) = \int_{\ell(p_2)+\ell^-(t)}^{\ell(p_2t\sigma)} (\mathscr{W}_{p_2t\sigma}(\lambda))^{-frac12} d\lambda - \int_{\ell(p_2)+\ell^-(t)}^{\ell(p_2t)} (\mathscr{W}_{p_2t}(\lambda))^{-frac12} d\lambda$. Moreover, by i) of Proposition A.4.2, we have that $(\forall \lambda \geq \ell^+(t\sigma))\ \mathscr{W}_{p_1t\sigma}(\ell(p_1) + \lambda) d\lambda = \mathscr{W}_{p_2t\sigma}(\ell(p_2) + \lambda) d\lambda$ and $(\forall \lambda \geq \ell^+(t))\ \mathscr{W}_{p_1t}(\ell(p_1) + \lambda) d\lambda = \mathscr{W}_{p_2t}(\ell(p_2) + \lambda) d\lambda$ which imply that $T(p_1t\sigma) - T(p_1t) = T(p_2t\sigma) - T(p_2t)$, since $\ell^+(t) \leq \ell^-(t)$ and, as noticed in Section 1.6, $\ell^+(t\sigma) \leq \ell^+(t)$. □

Appendix B

Appendix of Chapter 2

B.1 Basic plans

In this section we explain in more detail the basic plans.

k-CYCLE ROTATION. Let $C = (V_C, E_C)$ be a cycle, with a set of pebbles P, and a set of holes H, with $|H| \geq 1$. Let $\mathscr{A}$ be an initial configuration, $v \in \mathscr{A}(H)$, and $w \in V_C$ be such that $(v,w) \in E_C$ (i.e., w is the successor of v on C).

Let $\pi = u_1 = w, \ldots, u_n = v$ be the path on C from w to v. We define plan 1-CYCLE ROTATION as

$$r_1^C = (u_{n-1} \to u_n, \ldots, u_1 \to u_2). \tag{B.1}$$

In other words, for each j from $n-1$ to 1, if there is a pebble on u_j, we move it on u_{j+1}. The new configuration $\bar{\mathscr{A}}$ is defined as follows:

$$\bar{\mathscr{A}}(q) := \begin{cases} u_{j+1}, & \text{if } \mathscr{A}(q) = u_j \quad j = 1, \ldots, n-1; \\ w, & \text{if } \mathscr{A}(q) = v; \end{cases} \tag{B.2}$$

which means that all pebbles and holes on C change positions.

For $k \in \mathbb{N}$, a k-CYCLE ROTATION over C is obtained by performing k 1-CYCLE ROTATIONS over C. We denote the plan corresponding to a k-Cycle Rotation over C by r_k^C. Let $l_C = |V_C|$ be the length of cycle C. Plan $r_{l_c}^C$ is a *complete* rotation of C and brings all pebbles and holes back to their initial positions. In other words $r_{l_c}^C \sim \varepsilon$,

where ε is the empty plan. Since $\varepsilon \sim r^C_{l_C} = r^C_k r^C_{l_C-k}$, it follows that complementary rotation $r^C_{l_C-k}$ is an inverse plan of r^C_k.

If $C = [C_1, \ldots, C_m]$ is an ordered sequence of cycles, that are subgraphs of the same graph, and $k = (k_1, \ldots, k_m) \in \mathbb{N}^n$, R^C_k denotes the plan obtained by concatenating a k_1-Cycle Rotation over C_1, a k_2-Cycle Rotation over C_2, and analogous rotations over the remaining cycles of C, namely:

$$R^C_k = r^{C_1}_{k_1} \ldots r^{C_n}_{k_n}.$$

Set $s = (s_n, s_{n-1}, \ldots, s_1) = (l_{C_n} - k_n, \ldots, l_{C_1} - k_1)$ and $\hat{C} = [C_n, C_{n-1}, \ldots, C_1]$. Then

$$R^C_k R^{\hat{C}}_s = r^{C_1}_{k_1} \ldots r^{C_n}_{k_n} r^{C_n}_{s_n} \ldots r^{C_1}_{s_1} \sim \varepsilon,$$

since $r^{C_n}_{k_n} r^{C_n}_{s_n} \sim \varepsilon$, and analogous reductions holds for the remaining terms. This implies that $R^{\hat{C}}_s$ is an inverse plan of R^C_k. We denote $R^{\hat{C}}_s$ by $(R^C_k)^+$. In other words, an inverse plan of R^C_k consists in a sequence of complementary rotations, in inverse order.

BRING HOLE FROM v TO w. Let $\mathscr{A}$ be an initial configuration, such that $v \in \mathscr{A}(H)$ (i.e., v is an unoccupied vertex). There are two cases:

1. v and w belong to the same cycle C. In this case, let $\pi = u_1 = w, \ldots, u_n = v$ be the path on C from w to v. We define the plan BRING HOLE FROM v TO w as

$$h_{v,w} = (u_{n-1} \to u_n, \ldots, u_1 \to u_2). \tag{B.3}$$

   In other words, for each j from $n-1$ to 1, if there is a pebble on u_j, we move it on u_{j+1}. The new configuration $\bar{\mathscr{A}}$ is defined as follows:

$$\bar{\mathscr{A}}(q) := \begin{cases} u_{j+1}, & \text{if } \mathscr{A}(q) = u_j \quad j = 1, \ldots, n-1; \\ w, & \text{if } \mathscr{A}(q) = v; \\ \mathscr{A}(q), & \text{otherwise,} \end{cases} \tag{B.4}$$

   which means that only pebbles and holes along path π change positions. Moreover, since the graph is strongly connected, by Proposition 2.3.1 there exists a reverse plan $h^{-1}_{v,w}$, which returns pebbles and holes to their initial positions. In particular in this case the reverse plan consists in moving the pebbles along the cycle until the initial configuration is obtained again. We call BRING BACK HOLE FROM w TO v the plan $h^{-1}_{v,w}$.

2. Otherwise, by Observation 2.3.1, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \ldots, C_{i_n}]$ such that $v \in C_{i_1}$, $w \in C_{i_n}$, and for all $j = 1, \ldots, n$, $C_{i_j} \cap C_{i_{j+1}}$ contains at least one node a_{i_j}. Starting from C_{i_1}, we perform the following operations: we perform a $d(v, a_{i_1})$-CYCLE ROTATION over cycle C_{i_1}; for all $j = 2, \ldots, n-1$ we perform a $d(a_{i_j}, a_{i_{j+1}})$-CYCLE ROTATION over cycle C_{i_j}; we perform a $d(a_{i_{n-1}}, y)$-CYCLE ROTATION over cycle C_{i_n}. Setting $k = (d(w, a_{i_1}), d(a_{i_2}, a_{i_3}), \ldots, d(a_{i_{n-2}}, a_{i_{n-1}}))$, this sequence of rotations corresponds to R_k^C. Now, the hole is in the same cycle as w, so we continue as in step 1. So, BRING HOLE FROM v TO w is defined as

$$h_{v,w} = R_k^C \, h_{a_{i_{n-1}}, w}.$$

In this case, the reverse plan BRING BACK HOLE FROM w TO v is defined by

$$h_{v,w}^{-1} = h_{a_{i_{n-1}}, w}^{-1} (R_k^C)^+.$$

where $s = (s_n, s_{n-1}, \ldots, s_1) = (l_{C_n} - k_n, \ldots, l_{C_1} - k_1)$ and $\hat{C} = [C_n, C_{n-1}, \ldots, C_1]$.

BRING HOLE FROM v TO A SUCCESSOR OF w. Let $\mathscr{A}$ be an initial configuration, such that $v \in \mathscr{A}(H)$. Let $\pi = u_1 = w, \ldots, u_n = v$ be a shortest path from w to v, where u_2 is the successor of w along π. Then, BRING HOLE FROM v TO A SUCCESSOR OF w ($h_{v,s(w)}$) is defined as BRING HOLE FROM v TO u_2.

BRING BACK HOLE FROM A SUCCESSOR OF w TO v. Let $h_{v,s(w)}$ be a plan BRING HOLE FROM v TO w. We call BRING BACK HOLE FROM w TO v its reverse plan $h_{v,s(w)}^{-1}$.

B.2 Movements through biconnected components

In this section we show proofs of the lemmas for movements through biconnected components. These proofs provide instructions for the algorithm.

Proof of Entry Lemma 2.3.2. Fig. 2.18 illustrates this proof. Let $y \in V$ be such that $(v, y) \in E$. Let h_1 be the *destination* hole in w ($\mathscr{A}(h_1) = w$), and let h_2 be a *transport* hole in $V \setminus \{w\}$ (note that h_2 exists since we are assuming that $|H| \geq 2$).

If G is a partially-bidirectional cycle, we set $n = 1$ and $C_1 = C_G$, where C_G is the directed cycle contained in G. We perform a $d(w,y)$-CYCLE ROTATION over cycle C_G, where $d(w,y)$ is the distance between nodes w and y. In this way, hole h_1 moves to y. Next, pebble p moves on y with $v \to y$. Finally, we perform the complementary $L_{C_G} - d(y,w)$-CYCLE ROTATION over C_G, in order to move p to w. Namely, the plan is $f_{vw} = r^{C_G}_{d(w,y)}(v \to y)(r^{C_G}_{d(w,y)})^+$, and the final configuration is $\bar{\mathscr{A}}[v,w]$. Indeed, apart from p and h_1, which are exchanged, all pebbles and holes are moved l_{C_G} times, which means that they complete a full revolution, returning to their initial positions. If G has a *r-oed* $L = [L_0, \dots, L_r]$, let u be a successor of w such that $\mathscr{A}(h_2) = u$. If it does not exist, we perform BRING HOLE FROM $\mathscr{A}(h_2)$ TO A SUCCESSOR OF w and set u as the successor of w, which corresponds to the new position of h_2. At the end, we will bring back h_2 to its initial position with BRING BACK HOLE FROM A SUCCESSOR OF w TO $\mathscr{A}(h_2)$. By Observation 2.3.1, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \dots, C_{i_n}]$ such that $v \in C_{i_1}$ and $u \in C_{i_n}$ and for all $j = 1, \dots, n$, $C_{i_j} \cap C_{i_{j+1}}$ has at least two nodes a_{i_j} and b_{i_j} with $(a_{i_j}, b_{i_j}) \in E$. Starting from C_{i_1}, we perform the following operations: we perform a $d(w, a_{i_1})$-CYCLE ROTATION over cycle C_{i_1}; for all $j = 2, \dots, n-1$ we perform a $d(a_{i_j}, a_{i_{j+1}})$-CYCLE ROTATION over cycle C_{i_j}; we perform a $d(a_{i_{n-1}}, y)$-CYCLE ROTATION over cycle C_{i_n}. Setting $k = (d(w, a_{i_1}), d(a_{i_2}, a_{i_3}), \dots, d(a_{i_{n-1}}, y))$, this sequence of rotations corresponds to R^C_k. Then, we move p to y and perform the inverse sequence $(R^C_k)^+$. At the end of this plan, p is on w and all other pebbles are in their initial positions. Hence, the overall plan that allows us to prove the thesis is $f_{vw} = h_{h_2 s(w)} R^C_k (v \to y)(R^C_k)^+ h^{-1}_{hs(w)}$.

Proof of Go out Lemma 2.3.3. Let h_1 be the *destination* hole in w ($\mathscr{A}(h_1) = w$), and let h_2 be a *transport* hole in $V \setminus \{w\}$. Without loss of generality we can suppose that exists u a successor of v such that $\mathscr{A}(h_2) = u$. Otherwise, it would be enough to perform at the beginning BRING HOLE FROM $\mathscr{A}(h_2)$ TO A SUCCESSOR OF v and at the end we would bring back h_2 to its initial position with BRING BACK HOLE FROM A SUCCESSOR OF v TO $\mathscr{A}(h_2)$. By Observation 2.3.1, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \dots, C_{i_m}]$ such that $v, u \in C_{i_1}$ and $y \in C_{i_n}$ and for all $j = 1, \dots, n$, $C_{i_j} \cap C_{i_{j+1}}$ has at least two nodes a_{i_j} and b_{i_j} with $(a_{i_j}, b_{i_j}) \in E$. Starting from C_{i_1},

we perform the following operations: we perform a $d(u,b_{i_1})$-CYCLE ROTATION over cycle C_{i_1}; for all $j = 2,\dots,n-1$ we perform a $d(b_{i_j},b_{i_{j+1}})$-CYCLE ROTATION over cycle C_{i_j}; we perform a $d(b_{i_{n-1}},y)$-CYCLE ROTATION over cycle C_{i_n}. Setting $k = (d(u,b_{i_1}),d(b_{i_2},b_{i_3}),\dots,d(b_{i_{n-1}},y))$, this sequence of rotations corresponds to R_k^C. After these rotations, p is on $x \in V$, a predecessor of y. Then, we move p to y and then w and perform the inverse sequence $(R_k^C)^+$. At the end of this plan, p is on w and all other pebbles are in their initial positions. Hence, the overall plan that allows us to prove the thesis is $f_{vw} = R_k^C(x \to y)(y \to w)(R_k^C)^+ h_{hs(w)}^{-1}$.

Proof of Attached-Edge Lemma 2.3.4. Fig. 2.19 illustrates this proof. If the strongly biconnected component of D has a *r-oed*, the proof follows from Lemma 2.3.5. Otherwise, the strongly biconnected component is a partially-bidirectional cycle $G = (V,E)$. We consider the cycle C_G contained in G, which has lenght l_{C_G}. Let h_1 be the *destination* hole on w and h_2 a *transport* hole (which exists, since $|H| \geq 2$). Without loss of generality, suppose that $\mathscr{A}(h_2) = v$. Indeed, if this were not the case, we can BRING HOLE FROM $\mathscr{A}(h_2)$ TO v and finally bring back it to its initial position. In this case, the new initial configuration is $\bar{A} = \rho(\mathscr{A}, h_{\mathscr{A}(h_2)v})$ and we have to consider $\bar{u} = \mathscr{A}(p)$ and $\bar{w} = \bar{\mathscr{A}}(h_1)$. Let $y \in V$ be the cycle node that shares and arc with v, and consider the distances from u and w to y, $d_1 := d(u,y)$ and $d_2 := d(w,y)$. Performing a d_1-CYCLE ROTATION over C_G, we move p from w to y. Then, we move p on v with $y \to v$. Now, let

$$k = \begin{cases} d_2 - d_1 & if\ d_2 \geq d_1, \\ l + d_2 - d_1 & if\ d_2 < d_1. \end{cases}$$

We perform a k-CYCLE ROTATION over C_G, so that we move the hole h_1 from w to y. Next, we move p from v to y with $v \to y$. Finally, we perform a $d(y,w)$-CYCLE ROTATION over C_G to move p on w. To conclude, $f_{uw} = r_{d(u,y)}^{C_G}(y \to v)\, r_k^{C_G}(v \to y)\, r_{d(y,w)}^{C_G}$.

Proof of Stay in Lemma 2.3.5. Fig. 2.20 illustrates this proof. Let h_1 be the *destination* hole in w ($\mathscr{A}(h_1) = w$), and let h_2 be a *transport* hole in $V \setminus \{w\}$. Without loss of generality we can suppose that exists u a successor of v such that $\mathscr{A}(h_2) = u$ (see Figure 2.20a). Otherwise, it would be enough to perform at the beginning BRING

HOLE FROM $\mathscr{A}(h_2)$ TO A SUCCESSOR OF v and at the end we would bring back h_2 to its initial position with BRING BACK HOLE FROM A SUCCESSOR OF v TO $\mathscr{A}(h_2)$.

By Observation 2.3.1, there exists a sequence of cycles $C = [C_{i_1}, C_{i_2}, \ldots, C_{i_m}]$ such that $v \in C_{i_1}$ and $w \in C_{i_n}$ and for all $j = 1, \ldots, n$, $C_{i_j} \cap C_{i_{j+1}}$ has at least two nodes a_{i_j} and b_{i_j} with $(a_{i_j}, b_{i_j}) \in E$. In particular, choose $a_{i_{m-1}}$ such that his predecessor node x in $C_{i_{m-1}}$ does not belong to C_{i_m}.

Now, starting from C_{i_1}, we perform the following operations: we perform a $d(v, a_{i_1})$-CYCLE ROTATION over cycle C_{i_1}; for all $j = 2, \ldots, m-3$ we perform a $d(a_{i_j}, a_{i_{j+1}})$-CYCLE ROTATION over cycle C_{i_j}; we perform a $d(a_{i_{m-2}}, x)$-CYCLE ROTATION over cycle $C_{i_{m-1}}$ (Fig. 2.20b) and then a $d(w, a_{i_{m-1}})$-CYCLE ROTATION over cycle C_{i_m}. Setting $k = (d(w, a_{i_1}), d(a_{i_2}, a_{i_3}), \ldots, d(a_{i_{m-2}}, x), d(w, a_{i_{m-1}}))$, this sequence of rotations corresponds to R_k^C. Then, we move p to $a_{i_{m-1}}$ (Fig. 2.20c) and perform the inverse sequence $(R_k^C)^+$. At the end of this plan, p is on w and all other pebbles are in their initial positions (Fig. 2.20e). Hence, the overall plan that allows us to prove the thesis is $f_{vw} = R_k^C(x \to a_{i_{n-1}})(R_k^C)^+$.

Proof of Two Biconnected Components Lemma 2.3.6. Let $B_1 = (V_1, E_1)$ and $B_2 = (V_2, E_2)$ be the two biconnected components and v be the articulation point, i.e., $V_1 \cup V_2 = V$, $E_1 \cup E_2 = E$, $V_1 \cap V_2 = \{v\}$, and $E_1 \cap E_2 = \emptyset$. Let h_1 be the *destination* hole on b, and let h_2 be a *transport* hole. We discuss different cases.

1. $a \in V_1 \setminus \{v\}$ and $b \in V_2 \setminus \{v\}$.

   Without loss of generality, we assume that $\mathscr{A}(h_2) = v$. Indeed, if this is not the case, we can BRING HOLE FROM $\mathscr{A}(h_2)$ TO v, and the new initial configuration is $\bar{\mathscr{A}} = \rho(\mathscr{A}, h_{\mathscr{A}(h_2)v})$. Note that $h_{\mathscr{A}(h_2)v}$ could change the position either of h_1 or of p, i.e., either $\bar{a} = \bar{\mathscr{A}}(p) \neq a$ or $\bar{b} = \bar{\mathscr{A}}(h_1) \neq b$. By the procedure described below, we will reach $\bar{\mathscr{A}}[\bar{a}, \bar{b}]$, and at that point we will need to perform BRING BACK HOLE FROM v TO $\mathscr{A}(h_2)$ in order to obtain, by Observation 2.3.3, $\mathscr{A}[a, b] = \rho(\bar{\mathscr{A}}[\bar{a}, \bar{b}], h^{-1}_{v, \mathscr{A}(h_2)})$.

   Assuming $\mathscr{A}(h_2) = v$, let $y \in V_1$ be such that $(y, v) \in E_1$, i.e., y is a predecessor

of v. Now, by *Entry Lemma* 2.3.2, there exists a plan g such that, setting $\mathscr{A}^1 = \rho(\mathscr{A}, g)$, $\mathscr{A}^1(p) = b$ and, for all $q \in H \cup P \setminus \{p\}$ such that $\mathscr{A}^1(q) \in V_2$, $\mathscr{A}^1(q) = \mathscr{A}(q)$ holds. Plan g is defined as follows

$$g = h_{v,s(b)} R_k^C t_{a,v} (R_k^C)^+ h_{v,s(b)}^{-1},$$

where C is a sequence of cycles, k a vector, and $t_{a,v}$ is a plan which moves pebble p to the unoccupied vertex v. In particular, if B_1:

- has a *r-oed*: $t_{a,v} = a \Rightarrow v$, (see Definition 2.1.2) which by Theorem 2.1.1 exists since in B_1 there is at least one hole (h_1);
- is a partially-bidirectional cycle: $t_{a,v} = r_{d(a,v)}^{C_{B_1}}$.

After performing g, both holes h_1 and h_2 are in B_1 (in particular, $\mathscr{A}^1(h_2) = v$). If B_1 has a *r-oed*, by Lemma 2.3.5 there exists a plan f so that $\mathscr{A}^2 = \rho(\mathscr{A}^1, f)$ is such that $\mathscr{A}^2(h_1) = a$ and for all $q \in P \cup H \setminus \{h_1\}$, $\mathscr{A}^2(q) \in V_1$, $\mathscr{A}^2(q) = \mathscr{A}(q)$. So, finally $\mathscr{A}^2 = \mathscr{A}[a,b]$. If B_1 is a partially-bidirectional cycle, performing $r_{d(v,a)}^{C_{B_1}}$ is sufficient to bring p on a and the other pebbles of B_1 on their initial positions.

2. Suppose that $a, b \in V_1$.

   - B_1 has a *r-oed*. We can assume without loss of generality that $\mathscr{A}(h_2) \in V_1$ (if not, it would be enough to BRING HOLE FROM $\mathscr{A}(h_2)$ TO v and finally BRING BACK HOLE FROM v TO $\mathscr{A}(h_2)$). Since in B_1 there are two holes, by Lemma 2.3.5 we can move p to b without changing the final position of the other pebbles.
   - B_1 is a cycle. We can assume without loss of generality that $\mathscr{A}(h_2) \in V_2 \setminus \{v\}$. Then, by point 1) of this proof, we can first move p from a to $\mathscr{A}(h_2)$ and then from $\mathscr{A}(h_2)$ to b, without changing the final position of the other pebbles.

B.3 *CONVERT-PATH* algorithm

Finally, we show the proof of the Theorem 3.4.1, which provides the sequence of instructions for the *CONVERT-PATH* algorithm.

Proof of Theorem 3.4.1. Let f_{ab} be a plan on D. For each single move $m = u \to v$ in this plan, recalling Observation 2.3.2 and noting that u, v either belong to the same biconnected component or to the same corridor, there are two cases. In the first case there exists a non-trivial strongly biconnected component $B = (V_B, E_B)$ such that $u, v \in V_B$. Then, we define m', a corresponding plan on T, as follows: let S be the star in T corresponding to B, and let s be the trans-shipment vertex of S; then, $m' = (u \to s)(s \to v)$. In the second case, i.e., u, v belong to the same corridor, then $m' = m$. f'_{ab} is defined as the composition of all the moves m' just described.

Conversely, let f'_{ab} be a plan on T. Then, by Observation 2.3.4

1. If a and b are not in the same star on T, then they are not in the same strongly biconnected component on D. By Observation 2.3.4 f_{ab} will be composed by movements: from/to a corridor to/from a strongly biconnected component (f_{ab} exists by *Attached-Edge Lemma* 2.3.4); from a strongly biconnected component to another one, connected by an articulation point (f_{ab} exists by *Two Biconnected Components Lemma* 2.3.6); from a node to another one of the same corridor ($f'_{ab} = f_{ab}$).

2. If a and b belong to the same "star" on T, then a and b belong to the same strongly biconnected component $B_1 = (W_1, F_1)$ on D. There are two possibilities:

   (a) B_1 has at least two holes. In this case, if B_1 has a *r-oed*, by Lemma 2.3.5 $f_{a,b}$ exists. If B_1 is a partially-bidirectional cycle there are two cases:

   - there is another biconnected component $B_2 = (W_2, F_2)$ such that B_1 and B_2 are joined by an articulation point. In this case existence of $f_{a,b}$ follows from Lemma 2.3.6;
   - there is a node $v \in V \setminus W_1$ such that a node $w \in W_1$ with $(v,w), (w,v) \in E$ exists. Therefore, $G = (W_1 \cup \{v\}, F_1 \cup \{(v,w), (w,v)\})$ is a cycle

with an attached edge, and existence of $f_{a,b}$ follows from Lemma 2.3.4;

(b) B_1 has only one hole. In this case, let h_1 be the hole on b and h_2 a hole such that $\mathscr{A}(h_2) = w \notin W_1$ (which exists since $|H| \geq 2$) and $u \in W_1$ a node different from a and b (which exists since non-trivial biconnected components have at least three nodes). Let $h_{w,u}$ be the plan BRING HOLE FROM w TO u which moves the hole, and $\bar{\mathscr{A}} = (\mathscr{A}, h_{w,u})$. Then:

a') if $\bar{\mathscr{A}}(p), \bar{\mathscr{A}}(h_1) \in W_1$, then we do perform BRING HOLE FROM w TO u, we replace a and b with $\bar{\mathscr{A}}(p)$ and $\bar{\mathscr{A}}(h_1)$, and by point a) we find the plan $f_{\bar{\mathscr{A}}(p)\bar{\mathscr{A}}(h_1)}$; finally we perform $h_{w,u}^{-1}$ and h_2 returns to its initial position.

b') if $\bar{\mathscr{A}}(h_1) \in W_1$ but $\bar{\mathscr{A}}(p) \notin W_1$, we do perform BRING HOLE FROM w TO u and we fall into the case where start and final position of the pebble are not in the same biconnected component. Therefore, $f_{ab} = h_{w,u} f_{\bar{\mathscr{A}}(p)\bar{\mathscr{A}}(h_1)} h_{w,u}^{-1}$, where $f_{\bar{\mathscr{A}}(p)\bar{\mathscr{A}}(h_1)}$ exists by point 1).

c') if $\bar{\mathscr{A}}(p) \in W_1$ but $\bar{\mathscr{A}}(h_1) \notin W_1$, we do not perform BRING HOLE FROM w TO u. First, we move h_1 away from b by BRING HOLE FROM b TO u. Then, we perform BRING HOLE FROM w TO b. Then, we can proceed as in a') or b') with u replaced by b. Finally, we will need to perform BRING BACK HOLE FROM u TO b. In formulas, given $\tilde{\mathscr{A}} = \rho(\mathscr{A}, h_{b,u} h_{w,b})$ the final plan is $f_{ab} = h_{b,u} h_{w,b} f_{\tilde{\mathscr{A}}(p)\tilde{\mathscr{A}}(h_1)} h_{w,b}^{-1} h_{b,u}^{-1}$.

Appendix C

Appendix of Chapter 4

In this Appendix, we prove the result of Proposition 4.1.1, which provides an upper bound on the cardinality of $\mathscr{N}_r(f_0)$.

First, we define a ball $\mathscr{B}_r(v)$ centered at $v \in V$ of radius $r \in \mathbb{N}$:

$$\mathscr{B}_r(v) := \{u \in V : \ d(u,v) \leq r\}.$$

We denote by $\bar{\mathscr{B}}_r(v)$ the border of the ball, obtained by replacing $\leq$ with $=$ in the definition. Let $\phi = outdeg(G)$ be the maximum out-degree of digraph $G = (V,E)$. The following proposition provides an upper bound on the cardinality of $\bar{\mathscr{B}}_r(v)$.

Proposition C.0.1. *It holds that*

$$|\bar{\mathscr{B}}_r(v)| \leq \phi^r. \tag{C.1}$$

Proof. Let n_h be the number of nodes at distance h from v. Note that $n_1 \leq \phi$, and $\forall h \geq 2 \ n_h \leq n_{h-1}(\phi - 1)$. By induction, $n_h \leq \phi(\phi-1)^{h-1} \ \forall h \geq 1$. Therefore, an upper-bound for the number of nodes on the border of the ball is

$$|\bar{\mathscr{B}}_r(v)| \leq \phi(\phi-1)^{r-1} \leq \phi^r.$$

Next, we define a ball centered at $\mathscr{A} \in \mathscr{C}$ of radius r:

$$\mathscr{B}_r(\mathscr{A}) := \{\mathscr{A}^* \in \mathscr{C} : \ d(\mathscr{A}^*, \mathscr{A}) \leq r\},$$

denoting by $\bar{\mathscr{B}}_r(\mathscr{A})$ its border. An upper bound for the cardinality of the ball is given by the following proposition.

Proposition C.0.2. *It holds that*

$$|\mathscr{B}_r(\mathscr{A})| \leq 1 + \frac{(r+k-1)!}{(r-1)!(k-1)!}\phi^r. \tag{C.2}$$

Proof. First of all, we find an upper bound for the number of configurations at distance h from $\mathscr{A}$, i.e., an upper-bound for the cardinality of the border of the ball centered in $\mathscr{A}$ of radius h:

$$\bar{\mathscr{B}}_h(\mathscr{A}) = \left\{ \mathscr{A}^* \in \mathscr{C} : \sum_{i=1}^{|P|} d(\mathscr{A}(p_i), \mathscr{A}^*(p_i)) = h \right\}.$$

Let $(h_1, \cdots, h_{|P|})$ be a $|P|$-decomposition of h (i.e., $\sum_{i=1}^{|P|} h_i = h$). An upper bound for the number of configurations for which $d(\mathscr{A}(p_i), \mathscr{A}^*(p_i)) = h_i$ is

$$\prod_{i=1}^{|P|} |\bar{\mathscr{B}}_{h_i}(\mathscr{A}(p_i))| \leq \prod_{i=1}^{|P|} \phi^{h_i} = \phi^h.$$

By standard combinatorial arguments, the number of $|P|$-decompositions of h is

$$\frac{(h+(|P|-1))!}{h!(|P|-1)!}.$$

Therefore, the cardinality of the border of the ball of radius h can be bounded from above by:

$$|\bar{\mathscr{B}}_h(\mathscr{A})| = \frac{(h+(|P|-1))!}{h!(|P|-1)!} \prod_{i=1}^{|P|} |\bar{\mathscr{B}}_{h_i}(\mathscr{A}(p_i))| \leq$$

$$\leq \frac{(h+(|P|-1))!}{h!(|P|-1)!}\phi^h,$$

and the total number of configurations in $\mathscr{B}_r(\mathscr{A})$ can be overestimated as follows:

$$|\mathscr{B}_r(\mathscr{A})| = 1 + \sum_{h=1}^{r} |\bar{\mathscr{B}}_h(\mathscr{A})| \leq$$

$$\leq 1+\sum_{h=1}^{r}\frac{(h+|P|-1)!}{h!(|P|-1)!}\phi^h \leq 1+r\frac{(r+|P|-1)!}{r!(|P|-1)!}\phi^r.$$

Finally, given a radius $r\in\mathbb{N}$, we define the following **neighborhood** of $f_0\in\mathscr{E}^*_{\mathscr{A}}$:

$$\mathscr{N}_r(f_0):=\{\hat{g}\in\tilde{\mathscr{E}}^i_{\mathscr{A}}:\ |\hat{g}|\leq|f_0|, d(\hat{g},f_0)\leq r\}.$$

Here we consider the distance based on the max-min distance. The following proposition provides an upper bound on the cardinality of $\mathscr{N}_r(f_0)$. Note that, since the max-min distance is the smallest among the considered distances, the upper bound provided in the proposition is also valid for the distances based on the other distances previously discussed.

Proposition C.0.3. *It holds that:*

$$|\mathscr{N}_r(f_0)|\leq|f_0|^2\left(1+\frac{(r+k-1)!}{(r-1)!(k-1)!}\phi^r\right), \tag{C.3}$$

where $k=|P|$ *and* $\phi=outdeg(G)$.

Proof. Let $\hat{f}\in\tilde{\mathscr{E}}^i_{\mathscr{A}}$, $f_0\in\mathscr{E}^*_{\mathscr{A}}$ with $\hat{f}\in\mathscr{B}_r(f_0)$. Let f be a representative of $\hat{f}$ and let $I=\{1,\cdots,|\hat{f}|\}$ and $J=\{1,\cdots,|f_0|\}$. Then:

$$\alpha_1(\mathscr{N}_r(f_0))\subset\left(I\times\bigcup_{j\in J}\mathscr{B}_r(\psi_{f_0}(j))\right).$$

Indeed, for each representative f of $\hat{f}$, it holds that $|f|\leq|f_0|$ and, moreover, $d(\hat{f},f_0)\leq r$ implies that

$$\max_{1\leq i\leq|f|}\min_{1\leq j\leq|f_0|}d(\psi_f(i),\psi_{f_0}(j))\leq r,$$

and, in particular, for $i=|f|$ there exists j such that

$$d(\rho(\mathscr{A},f),\psi_{f_0}(j))\leq r,$$

which means that $\rho(\mathscr{A},f)\in\mathscr{B}_r(\psi_{f_0}(j))$. Therefore, recalling that α is injective,

$$|\mathscr{N}_r(f_0)|\leq\left|I\times\bigcup_{j\in J}\mathscr{B}_r(\psi_{f_0}(j))\right|.$$

Then, also in view of (C.2), we have that:

$$|\mathscr{N}_r(f_0)| \leq |f_0| \left(\sum_{j=1}^{|f_0|} |\mathscr{B}_r(\psi_{f_0}(j))| \right) \leq$$

$$\leq |f_0|^2 \left(1 + \frac{(r+k-1)!}{(r-1)!(k-1)!} \phi^r \right).$$

C.1 Proof of Proposition 4.1.1.

Note that

$$\frac{(r+k-1)!}{(r-1)!(k-1)!} \leq \frac{(r+k)^r}{(r-1)!}.$$

Defining $C := \frac{1}{(r-1)!}$ and reminding the result of Proposition C.0.3 we have that

$$|\mathscr{N}_r(f_0)| \leq |f_0|^2 (1 + C(r+k)^r \phi^r),$$

where $k = |P|$ and $\phi = outdeg(G)$, which is the thesis of Proposition 4.1.1.

Bibliography

[1] Roni Stern, Nathan Sturtevant, Ariel Felner, Sven Koenig, Hang Ma, Thayne Walker, Jiaoyang Li, Dor Atzmon, Liron Cohen, TK Kumar, et al. Multi-agent pathfinding: Definitions, variants, and benchmarks. In *Proceedings of the International Symposium on Combinatorial Search*, volume 10, pages 151–158, 2019.

[2] Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Efficient solution algorithms for the bounded acceleration shortest path problem. In *2021 60th IEEE Conference on Decision and Control (CDC)*, pages 5729–5734. IEEE, 2021.

[3] Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Shortest path with acceleration constraints: complexity and approximation algorithms. *Computational Optimization and Applications*, 83(2):555–592, 2022.

[4] Stefano Ardizzoni, Luca Consolini, Mattia Laurini, and Marco Locatelli. Solution algorithms for the bounded acceleration shortest path problem. *IEEE Transactions on Automatic Control*, 68(3):1910–1917, 2022.

[5] Vincenzo Auletta, Angelo Monti, Mimmo Parente, and Pino Persiano. A linear-time algorithm for the feasibility of pebble motion on trees. *Algorithmica*, 23(3):223–245, 1999.

[6] Adi Botea and Pavel Surynek. Multi-agent path finding on strongly biconnected digraphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 29, 2015.

[7] Boris De Wilde, Adriaan W Ter Mors, and Cees Witteveen. Push and rotate: a complete multi-agent pathfinding algorithm. *Journal of Artificial Intelligence Research*, 51:443–492, 2014.

[8] Daniel Martin Kornhauser, Gary Miller, and Paul Spirakis. *Coordinating Pebble Motion on Graphs, the Diameter of Permutation Groups, and Applications*. PhD thesis, M. I. T., Dept. of Electrical Engineering and Computer Science, 1984.

[9] Stefano Ardizzoni, Luca Consolini, Marco Locatelli, Irene Saccani, and Bernhard Nebel. An algorithm with improved complexity for pebble motion/multi-agent path finding on trees. *Journal of Artificial Intelligence*.

[10] Stefano Ardizzoni, Luca Consolini, Marco Locatelli, and Irene Saccani. Multi-agent path finding on strongly connected digraphs. In *2022 IEEE 61st Conference on Decision and Control (CDC)*, pages 7194–7199. IEEE, 2022.

[11] Stefano Ardizzoni, Luca Consolini, Marco Locatelli, and Irene Saccani. Constrained multi-agent path finding on directed graphs. *Automatica*.

[12] Stefano Ardizzoni, Luca Consolini, Marco Locatelli, and Irene Saccani. Local optimization of mapf solutions on directed graphs. In *2023 IEEE 62st Conference on Decision and Control (CDC)*, 2023.

[13] A. Abbadi and V. Prenosil. Safe path planning using cell decomposition approximation. In M. Hrubý, editor, *International Conference Distance learning simulation and communication*, pages 8–14, May 2015.

[14] R. Seif and M. A. Oskoei. Mobile robot path planning by RRT* in dynamic environments. *International Journal of Intelligent Systems and Applications*, 7(5):24–30, 2015.

[15] R. V. Cowlagi and P. Tsiotras. Shortest distance problems in graphs using history-dependent transition costs with application to kinodynamic path planning. In *2009 American Control Conference*, pages 414–419, 2009.

[16] B. C. Dean. Shortest paths in fifo time-dependent networks: Theory and algorithms,. Technical report, Massachusetts Institute of Technology, 2004.

[17] E. He, N. Boland, G. Nehmauser, and M. Savelsbergh. Time-dependent shortest path problems with penalties and limits on waiting. *INFORMS Journal on Computing*, 33(3):997–1014, 2021.

[18] I. Ioachim, S. Gélinas, F. Soumis, , and J. Desrosiers. A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, 31:193–204, 1998.

[19] N. J. Nilsson. *Problem-Solving Methods in Artificial Intelligence*. McGraw-Hill, New York, 1971.

[20] S. Koenig, M. Likhachev, and D. Furcy. Lifelong planning A*. *Artificial Intelligence*, 155(1–2):93–146, 2004.

[21] D. Ferguson and A. Stentz. Field D*: an interpolation-based path planner and replanner. *Springer Tracts in Advanced Robotics*, 28:239–253, 2007.

[22] D. H. Kim, N. Trong Hai, and W. Yeol Joe. A guide to selecting path planning algorithm for automated guided vehicle (AGV). In *AETA 2017 - Recent Advances in Electrical Engineering and Related Sciences: Theory and Application*, pages 587–596, Cham, 2018. Springer.

[23] K. Kant and S. W. Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *The International Journal of Robotics Research*, 5(3):72–89, 1986.

[24] M. Raineri, S. Perri, and C. Guarino Lo Bianco. Online velocity planner for laser guided vehicles subject to safety constraints. In *2017 IEEE/RSJ International*

Conference on Intelligent Robots and Systems (IROS), pages 6178–6184, Sep. 2017. doi:10.1109/IROS.2017.8206519.

[25] D. Lam, C. Manzie, M. C. Good, and R.R. Bitmead. Receding horizon time-optimal control for a class of differentially flat systems. *Systems & Control Letters*, 83:61–66, 2015.

[26] T. Faulwasser, V. Hagenmeyer, and R. Findeisen. Optimal exact path-following for constrained differentially flat systems. In *Proceedings of the 18th World Congress of The International Federation of Automatic Control*, pages 9875–9880, 2011.

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

[28] Jack Edmonds and Delbert Ray Fulkerson. Bottleneck extrema. *Journal of Combinatorial Theory*, 8(3):299–306, 1970.

[29] Maurice Pollack. The maximum capacity through a network. *Operations Research*, 8(5):733–736, 1960.

[30] Robert Garfinkel, Elena Fernández, and Timothy J Lowe. The k-centrum shortest path problem. *Top*, 14:279–292, 2006.

[31] Daniel Villeneuve and Guy Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165(1):97–107, 2005.

[32] Bi Yu Chen, Xiao-Wei Chen, Hui-Ping Chen, and William HK Lam. Efficient algorithm for finding k shortest paths based on re-optimization technique. *Transportation Research Part E: Logistics and Transportation Review*, 133:101819, 2020.

[33] Michael R Garey and David S Johnson. Computers and intractability. *A Guide to the*, 1979.

[34] Gabriel Y Handler and Israel Zang. A dual algorithm for the constrained shortest path problem. *Networks*, 10(4):293–309, 1980.

[35] Paola Festa. Constrained shortest path problems: state-of-the-art and recent advances. In *2015 17th International Conference on Transparent Optical Networks (ICTON)*, pages 1–17. IEEE, 2015.

[36] Jérémy Omer and Michael Poss. Time-dependent shortest paths with discounted waits. *Networks*, 74(3):287–301, 2019.

[37] Moshe Dror. Note on the complexity of the shortest path models for column generation in vrptw. *Operations Research*, 42(5):977–978, 1994.

[38] Martin Desrochers and François Soumis. A reoptimization algorithm for the shortest path problem with time windows. *European Journal of Operational Research*, 35(2):242–254, 1988.

[39] Stuart E Dreyfus. An appraisal of some shortest-path algorithms. *Operations research*, 17(3):395–412, 1969.

[40] Ariel Orda and Raphael Rom. Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length. *Journal of the ACM (JACM)*, 37(3):607–625, 1990.

[41] L. Consolini, M. Laurini, M. Locatelli, and A. Minari. A solution of the minimum-time speed planning problem based on lattice theory. *Journal of the Franklin Institute*, 357:7617–7637, 2020.

[42] D. Gelperin. On the optimality of A*. *Artificial Intelligence*, 8(1):69–76, 1977.

[43] S. Edelkamp and S. Schroedl. *Heuristic Search: Theory and Applications*. Elsevier, 225 Wyman Street, Waltham, MA 02451, USA, 2011.

[44] Jingjin Yu and Steven M LaValle. Structure and intractability of optimal multi-robot path planning on graphs. In *Twenty-Seventh AAAI Conference on Artificial Intelligence*, 2013.

[45] Bernhard Nebel. On the computational complexity of multi-agent pathfinding on directed graphs. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pages 212–216, 2020.

[46] Roberto Asín Achá, Rodrigo López, Sebastian Hagedorn, and Jorge A Baier. Multi-agent path finding: A new boolean encoding. *Journal of Artificial Intelligence Research*, 75:323–350, 2022.

[47] Ebtehal Turki Saho Alotaibi and Hisham Al-Rawi. Push and spin: A complete multi-robot path planning algorithm. In *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pages 1–8. IEEE, 2016.

[48] Adi Botea, Davide Bonusi, and Pavel Surynek. Solving multi-agent path finding on strongly biconnected digraphs. *Journal of Artificial Intelligence Research*, 62:273–314, 2018.

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

[50] David Silver. Cooperative pathfinding. In *Proceedings of the Aaai Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 1, pages 117–122, 2005.

[51] Gabriele Röger and Malte Helmert. Non-optimal multi-agent pathfinding is solved (since 1984). In *Proceedings of the International Symposium on Combinatorial Search*, volume 3, pages 173–174, 2012.

[52] Mokhtar M Khorshid, Robert C Holte, and Nathan R Sturtevant. A polynomial-time algorithm for non-optimal multi-agent pathfinding. In *Fourth Annual Symposium on Combinatorial Search*, 2011.

[53] Gilad Goraly and Refael Hassin. Multi-color pebble motion on graphs. *Algorithmica*, 58(3):610–636, 2010.

[54] Athanasios Krontiris, Ryan Luna, and Kostas Bekris. From feasibility tests to path planners for multi-agent pathfinding. In *International Symposium on Combinatorial Search*, volume 4, 2013.

[55] Christos H Papadimitriou, Prabhakar Raghavan, Madhu Sudan, and Hisao Tamaki. Motion planning on a graph. In *Proceedings 35th Annual Symposium on Foundations of Computer Science*, pages 511–520. IEEE, 1994.

[56] Zhilin Wu and Stéphane Grumbach. Feasibility of motion planning on acyclic and strongly connected directed graphs. *Discrete Applied Mathematics*, 158(9):1017–1028, 2010.

[57] Zhilin Wu and Stéphane Grumbach. Feasibility of motion planning on directed graphs. In *Theory and Applications of Models of Computation: 6th Annual Conference, TAMC 2009, Changsha, China, May 18-22, 2009. Proceedings 6*, pages 430–439. Springer, 2009.

[58] Vincenzo Auletta and Pino Persiano. Optimal pebble motion on a tree. *Information and Computation*, 165(1):42–68, 2001.

[59] Gruia Călinescu, Adrian Dumitrescu, and János Pach. Reconfigurations in graphs and grids. *SIAM Journal on Discrete Mathematics*, 22(1):124–138, 2008.

[60] Adrian Dumitrescu. Mover problems. *Thirty Essays on Geometric Graph Theory*, pages 185–211, 2013.

[61] Hang Ma and Sven Koenig. Optimal target assignment and path finding for teams of agents. In Catholijn M. Jonker, Stacy Marsella, John Thangarajah, and Karl Tuyls, editors, *Proceedings of the 2016 International Conference on Autonomous Agents & Multiagent Systems, Singapore, May 9-13, 2016*, pages 1144–1152. ACM, 2016. URL: `http://dl.acm.org/citation.cfm?id=2937092`.

[62] Mehmet Hakan Karaata. A stabilizing algorithm for finding biconnected components. *Journal of Parallel and Distributed Computing*, 62(5):982–999, 2002.

[63] Douglas Brent West et al. *Introduction to graph theory*, volume 2. Prentice hall Upper Saddle River, 2001.

[64] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[65] Aric A. Hagberg, Daniel A. Schult, and Pieter J. Swart. Exploring network structure, dynamics, and function using networkx. In Gaël Varoquaux, Travis Vaught, and Jarrod Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 11 – 15, Pasadena, CA USA, 2008.

[66] Jiaoyang Li, Pavel Surynek, Ariel Felner, Hang Ma, TK Satish Kumar, and Sven Koenig. Multi-agent path finding for large agents. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 7627–7634, 2019.

[67] Dor Atzmon, Roni Stern, Ariel Felner, Glenn Wagner, Roman Barták, and Neng-Fa Zhou. Robust multi-agent path finding. In *Proceedings of the International Symposium on Combinatorial Search*, volume 9, pages 2–9, 2018.

[68] Petr Kolman and Ondřej Pangrác. On the complexity of paths avoiding forbidden pairs. *Discrete Applied Mathematics*, 157(13):2871–2876, 2009.

[69] Eugene L. Lawler, Jan Karel Lenstra, and AHG Rinnooy Kan. Generating all maximal independent sets: NP-hardness and polynomial-time algorithms. *SIAM Journal on Computing*, 9(3):558–565, 1980.

[70] Christos G. Cassandras and Stéphane Lafortune. *Introduction to Discrete Event Systems*. Springer, third edition edition, 2021.

[71] W. Murray Wonham and Kai Cai. *Supervisory Control of Discrete-Event Systems*. Communications and Control Engineering. Springer International Publishing : Imprint: Springer, 1st ed. 2019 edition, 2019.

[72] Peyman Gohari and Walter Murray Wonham. On the complexity of supervisory control design in the RW framework. *IEEE Transactions on Systems, Man and Cybernetics, Part B (Cybernetics)*, 30(5):643–652, 2000.

[73] David Pisinger and Stefan Ropke. Large neighborhood search. *Handbook of metaheuristics*, pages 99–127, 2019.

[74] L. Consolini, M. Locatelli, A. Minari, and A. Piazzi. An optimal complexity algorithm for minimum-time velocity planning. *Systems & Control Letters*, 103:50–57, 2017.

Ringraziamenti

Ringrazio i professori Marco Locatelli e Luca Consolini per avermi accompagnato con serenità e positività nella scoperta del mondo della ricerca, facendomi scoprire la bellezza di questo mestiere. Li ringrazio anche per avermi dato fiducia e per aver creduto in me, per avermi lasciato il giusto spazio, consigliandomi e guidandomi quando necessario.

I thank Professor Bernhard Nebel for his full availability to work on a project with me, making me grow thanks to his considerable experience. I also thank him for giving me useful and valuable advice for my future.

Ringrazio l'azienda OCME s.r.l. e in particolare Marcello Martinucci e Federico Iasoni per aver seguito insieme a me il progetto MAF, contribuendo a mantenere il collegamento tra azienda e univerità, nonostante le difficoltà incontrare lungo il percorso.

Ringrazio la mia famiglia per avermi sempre sostenuto nelle mie scelte e per aver creduto in me. In particolare, ringrazio Eleonora per esserci stata sempre, in ogni momento, sia di gioia che di difficoltà.

Ringrazio i colleghi Mattia, Irene e Rafael per il lavoro svolto insieme e anche Davide, Marina, Andrea, Giammarco e Shabnam per il supporto e per i momenti di svago.