Graphs of Convex Sets with Applications to Optimal Control and Motion Planning

by

Tobia Marcucci

B.S., University of Pisa (2013)
S.M., University of Pisa (2015)

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2024

Authored by:	Tobia Marcucci Department of Electrical Engineering and Computer Science May 14, 2024
Certified by:	Russ Tedrake Professor of Electrical Engineering and Computer Science Thesis Supervisor
Certified by:	Pablo A. Parrilo Professor of Electrical Engineering and Computer Science Thesis Supervisor
Accepted by:	Leslie A. Kolodziejski Professor of Electrical Engineering and Computer Science Chair, Department Committee on Graduate Students

Graphs of Convex Sets with Applications to Optimal Control and Motion Planning

by

Tobia Marcucci

Submitted to the Department of Electrical Engineering and Computer Science
on May 14, 2024, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis introduces a new class of problems at the interface of combinatorial and convex optimization. We consider graphs where each vertex is paired with a convex program, and each edge couples two programs through additional convex costs and constraints. We call such a graph a Graph of Convex Sets (GCS). Over a GCS we can formulate any optimization problem that we can formulate over an ordinary weighted graph, with scalar costs on the vertices and edges. In fact, for any fixed choice of the variables in the convex programs, a GCS reduces to a weighted graph where we can seek, e.g., a path, a matching, a tour, or a spanning tree of minimum cost. The challenge in a GCS problem lies in solving the discrete and the continuous components of the problem jointly.

By combining the modelling power of graphs and convex optimization, GCSs are a flexible framework to formulate and solve many real-world problems. The graph and the combinatorial goal (e.g., finding a path or a tour) model the high-level discrete skeleton of a problem. The convex costs and constraints fill in the low-level continuous details. The primary contribution of this thesis is an efficient and unified method for solving any GCS problem. Starting from an integer-linear-programming formulation of an optimization problem over a weighted graph, this method formulates the corresponding GCS problem as an efficient Mixed-Integer Convex Program (MICP). This MICP can then be solved to global optimality using common branch-and-bound solvers, or approximately by rounding the solution of its convex relaxation. Importantly, both the formulation of the MICP and its solution are fully automatic, and a user of our framework does not need any expertise in mixed-integer optimization.

We first describe the GCS framework and the formulation of our MICP in general terms, without presupposing the specific combinatorial problem to be solved over the GCS. We illustrate our techniques through multiple examples spanning logistics, transportation, scheduling, navigation, and computational geometry. Then we focus on the Shortest-Path Problem (SPP) in GCS. This problem is particularly interesting since it generalizes a wide variety of multi-stage decision-making problems and, using our techniques, it can be solved very effectively. We consider two main applications of the SPP in GCS: optimal control of dynamical systems and collision-free motion

planning. In these two areas, our techniques either generalize or significantly improve upon algorithms and optimization methods that have been developed for decades and are widely used in academia and industry.

Lastly, the techniques introduced in this thesis are implemented in the software packages `Drake` and `gcspy`. The former is a large and mature software for robotics. It is open-source and widely used by the community. The second is a very simple and lightweight Python package which is also open source. In this thesis, we will illustrate the usage of `gcspy` through multiple basic examples.

Thesis Supervisor: Russ Tedrake
Title: Professor of Electrical Engineering and Computer Science

Thesis Supervisor: Pablo A. Parrilo
Title: Professor of Electrical Engineering and Computer Science

Acknowledgments

The PhD at MIT has been the most exciting intellectual experience of my life. There are many people that supported and helped me during this long journey, and I would like to thank them all.

I begin with my advisors, Russ Tedrake and Pablo Parrilo. Russ hosted me seven years ago as a visiting student and gave me the extraordinary opportunity of joining the Robot Locomotion Group (RLG). His way of doing research and thinking about science has shaped me as a researcher and, more importantly, as a person. He has been a brilliant advisor, capable of both directing me towards fundamental research questions and helping me with the lowest-level technical issues. In a time where academic research is every day more frenetic, Russ has taught me the importance of slowing down, understanding things carefully, and never being satisfied with superficial explanations. I could not have asked for a more valuable lesson to learn. Working with Pablo has been a unique learning experience, and I am deeply thankful to have had this opportunity. I have been fortunate to meet multiple people that have incredible wisdom and knowledge of their field, but none quite matches Pablo. Pablo has been crucial for me to identify a research topic where theory and practice meet, and mutually amplify each other.

I am deeply thankful to Stephen Boyd for all the time he dedicated to me, and for hosting me at Stanford University during the last years of the PhD. Collaborating with him has been a true privilege, as well as a great pleasure. I like to believe that through the numerous meetings and conversations, I have absorbed even a tiny fraction of his sharpness and clarity of thought.

I would like to thank all the members of the RLG. Jack Umenberger has been the best collaborator I could ever imagine having, and also one of my closest friends. I wish that in the future there will be opportunities of working together again. I am very thankful to Robin Deits, who has been a fantastic mentor for my first few years at MIT. I would like to thank Twan Koolen for the many research insights and chats, as well as for all the lunches, dinners, and weekends together. Some of the material in the last part of this thesis is fruit of the work of Mark Petersen and David von Wrangel, to whom I am very grateful. A special thanks goes also to all the other members of the RLG that I have interacted and collaborated with: Pang, Terry, Abhishek, Yunzhu, Hongkai, Greg, Maggie, Max, Kaiqing, Lucas, Pete, Weiqiao, Shen, Sadra, Frank, Alex, Peter, Savva, Bernhard, Lujie, and Tommy. I would also like to thank Alexandre Megretski for the many meetings during the first half of my PhD.

I would not have started working on the topics of this thesis if it was not for the inspiration that I got from my professors at the University of Pisa. In particular,

Antonio Bicchi and Marco Gabiccini. I would like to thank them for setting me on this fantastic journey, and for their patience during my first years of research. From the days at the University of Pisa, a special thanks goes also to Cosimo Della Santina, Manolo Garabini, and Alessio Artoni.

I have been fortunate to have Manfred Morari and Sertac Karaman as members of my thesis committee. I am deeply thankful to them for the support over the last few years, and for the many insightful suggestions on the material presented in this thesis.

I would like to thank my family, that, despite the physical distance, has always been very close to me. Lastly, I am grateful for having met Eleni, who has been the most wonderful companion I could have ever imagined for this journey.

Contents

Chapter 1

Introduction

1.1 Motivation and goals

Making optimal decisions quickly is a central theme in engineering. Hugely scalable methods for discrete decision making are widely used in everyday life. For example, through a graph search, Google Maps can find the fastest route from San Francisco to New York in a fraction of a second. Similarly, continuous (specifically, convex) optimization is the backbone of some of the most advanced engineering systems in the world, such as the FALCON 9 rocket by SpaceX [18]. However, the efficient blending of discrete and continuous decision making remains an open challenge. Examples of this challenge can be found in almost any engineering field, e.g., power electronics, energy systems, chemical processes, finance, and logistics. Below are three examples that come from world-leading companies in robotics:

- The robot Atlas by Boston Dynamics is the most agile humanoid in the world and has literally revolutionized people's perception of robotics. To navigate through the parkour course shown in the left of Figure 1-1, Atlas must determine both the sequence of parkour obstacles to overcome and the motion of its center of mass and limbs. The first involves a discrete optimization problem, where the robot matches each footstep with a flat region on the ground. The second is a continuous optimization problem that requires computing a physically feasible way to run through the selected footsteps. Importantly, these two problems are tightly coupled and need to be solved jointly for the robot to move efficiently. Currently, Atlas relies on an engineer that hand-codes the solution of the discrete problem [24]. Therefore, the robot is not able to autonomously plan its motion through a novel parkour environment, or even adapt its motion along a pre-established path if, e.g., a ramp is missing. The development of planning

Figure 1-1: Three examples of discrete-continuous decision making in robotics. *Left:* the humanoid robot Atlas by Boston Dynamics doing parkour. *Center:* the robot arm Robin by Amazon Robotics sorting packages. *Right:* a Skydio drone flying through a forest.

and control algorithms that can reason about the discrete and the continuous problems simultaneously is a major bottleneck towards the full autonomy of this robot.

- The robot arm Robin in the center of Figure 1-1 has sorted more than one billion packages in the Amazon warehouses [2]. Its sorting problem involves a complex mix of continuous trajectory optimization and discrete bin sorting and packing. The amount of packages that this robot handles in a unit of time is almost proportional to the income of the company. Therefore, there is a huge value in deploying advanced optimization methods that improve even by a small percentage the productivity of this robot. A one-percent increase of Robin's operating speed would result in ten million extra packages processed every year.

- A drone flying through a forest has to plan its continuous position and orientation, and simultaneously answer discrete questions like "Do I avoid the tree by going left or right?". Skydio drones (right of Figure 1-1) are some of the most advanced autonomous drones in the world. They are exceptionally proficient in short-term reactive collision avoidance [172], but can lack long-term reasoning when avoiding obstacles in intricate environments.

Why are we unable to deal with these discrete-continuous problems efficiently? The first hurdle is the number of moving pieces that these problems have. Just writing on a computer the mathematical problem of optimizing the motion of a humanoid robot requires days of work for an expert engineer. The second hurdle is the computational hardness (typically NP-hardness) of these problems: even when the optimization problem is formulated in full detail, solving it can be extremely challenging and time consuming.

The objective of this thesis is the development of a modeling and computational framework that meets the following criteria:

1. *A framework that captures the essence of our decision-making problems.* Specifically, the interplay between the high-level discrete skeleton and the low-level continuous details that appear in problems like the ones shown in Figure 1-1. The framework should do so at the modelling level, by allowing a modular assembly of a decision-making problem. It should also do so at the mathematical level, by providing simple abstractions that allow us to make crisp statements about our problems and algorithms.

2. *A framework that is amenable to efficient optimization.* Optimization concepts and tools such as tight convex relaxations, strong mixed-integer formulations, and approximation algorithms give us the vocabulary and the means to deal with and overcome the hardness of our problems. Our decision-making framework must take full advantage of the most powerful techniques that come from combinatorial and convex optimization.

3. *A framework that is high-level for the user.* The optimization techniques just mentioned allow us to cope with hard problems efficiently, but it is unrealistic to expect that the average user of our framework will be familiar with them. Our framework must then shield the user from the complexity of the optimization tools that are used behind the scenes to solve the decision-making problems.

1.2 Graphs of convex sets

In this thesis we propose a framework that we call **Graph of Convex Sets** (GCS); see Figure 1-2 for an illustration. A GCS is a graph (either directed or undirected) where every vertex is paired with a convex program. Each convex program is defined by a set of variables, a set of convex constraints, and a convex cost function that we seek to minimize. Every edge in a GCS is associated with additional convex costs and constraints that couple the convex programs at the endpoints of the edge.

Any optimization problem that we can formulate over an ordinary weighted graph is naturally generalized to GCSs. In fact, if we fix the value of the variables in each convex program, a GCS reduces to an ordinary weighted graph where vertex and edge costs are scalars, obtained by evaluating the cost functions in the GCS. Then, in this weighted graph, we can search for, e.g., a path, a tour, or a spanning tree of minimum cost. The challenge in a GCS problem is the coupling between the continuous and the discrete components of the problem: the continuous variables of each convex

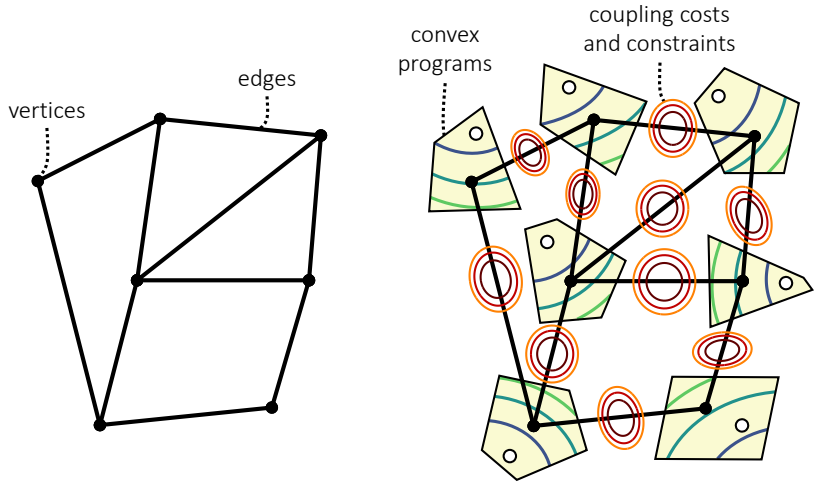

Figure 1-2: *Left:* ordinary graph with vertices and edges. *Right:* graph of convex sets, where vertices are paired with convex programs and edges with convex costs and constraints.

program need to be chosen so that the optimal value of the resulting discrete problem is minimum. This leads to an optimization problem that is mixed, continuous and discrete.

As examples, Figure 1-3 illustrates the **Shortest-Path Problem** (SPP), the **Traveling-Salesperson Problem** (TSP), and the **Minimum-Spanning-Tree Problem** (MSTP) in GCS. Let us consider the SPP first. In its canonical formulation, the SPP asks for a path of minimum cost that connects two prescribed vertices of a weighted graph. The cost of a path is defined as the sum of the costs of the vertices and edges along the path. In the SPP in GCS the vertices in the selected path decide which convex programs are activated and deactivated. Similarly, the edges in the path decide which coupling costs and constraints are enforced. The cost of the path is then the optimal value of a larger convex program, that includes all the activated convex programs, coupling costs, and coupling constraints. A naive algorithm for the SPP in GCS would then solve a convex program for each path in our graph, and choose the path that yields the minimum optimal value. Of course this algorithm would be very inefficient, since the number of paths in a graph can grow very quickly with the size of the graph. Our goal is to devise an efficient way of optimizing the discrete path through the graph and the continuous variables in the activated convex programs jointly. The TSP in GCS and the MSTP in GCS are formulated through analogous extensions of the ordinary TSP and MSTP.

The main contribution of this thesis is an efficient and unified method for solving any GCS problem. This method starts from the formulation of a graph optimization problem (e.g., the SPP, the TSP, or the MSTP) as an **Integer Linear Program**

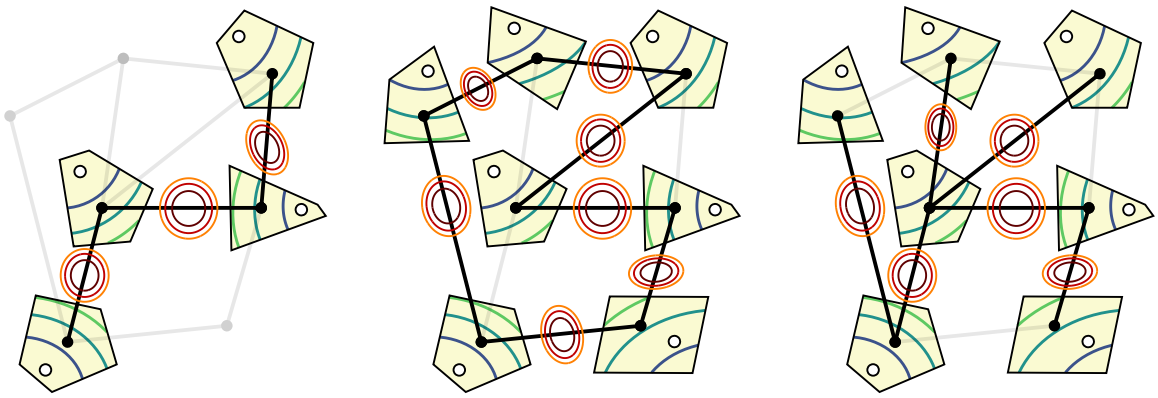

Figure 1-3: Illustrative examples of GCS problems. *Left:* shortest-path problem in GCS. *Center:* travelling-salesperson problem in GCS. *Right:* minimum-spanning-tree problem in GCS.

(ILP).[1] These ILPs are classical and very well studied. They typically have one binary variable per vertex and edge in the graph. The role of these variables is to take unit value if the corresponding vertex or edge is part of the optimal solution (e.g., the optimal path, tour, or spanning tree), and to take value of zero otherwise. Our method takes this ILP that models the discrete graph problem and automatically translates it into an efficient **Mixed-Integer Convex Program** (MICP) that models the corresponding GCS problem.[2] This translation is based on perspective transformations (or, as we will call them in this thesis, homogenization transformations). This is a popular tool in mixed-integer optimization [34, 177, 68, 87, 89] that allows us to activate and deactivate the convex costs and constraints in the GCS using the binary variables from the ILP.

Using a **Branch-and-Bound** (BB) algorithm, MICPs can always be solved to global optimality or declared infeasible if a solution does not exist. However, in general, the runtime of these algorithms can be very large. Our MICP is efficient in two main directions:

- It computationally lightweight, i.e., it has a small number of variables and constraints.

- It has tight convex relaxation, i.e., the convex program obtained by allowing the discrete variables to take continuous values approximates tightly the initial nonconvex problem.

[1] An ILP is an optimization problem with affine cost, affine constraints, and integer decision variables.

[2] An MICP is an optimization problem with convex cost and convex constraints. One part of the variables is real valued, and the other part is integer valued.

Because of these two features of our MICPs, BB algorithms converge quickly and reliably. Alternatively, our MICPs can be solved even faster, but approximately, by directly rounding the solution of their convex relaxation.

How does the GCS framework perform in terms of the criteria listed in the previous section?

1. *GCS captures the essence of our decision-making problems.* It encapsulates the interaction and the feedback between the discrete and the continuous components of our problems. The graph models the high-level discrete decision making, e.g., the parkour obstacle sequence, the bin sorting, and the collision avoidance for the problems in Figure 1-1. The convex programs describe the low-level continuous elements, such as the robot dynamics and energy consumption. A GCS problem can be defined in a modular fashion, one convex program at the time. GCS problems are also easy to work with at a mathematical level. For example, their computational complexity can be easily established via simple reduction arguments.

2. *GCS is amenable to efficient optimization.* Our framework takes full advantage of fundamental results from combinatorial and graph optimization, and naturally translates them into our mixed-integer setting. The MICPs that our method generates automatically are highly efficient. In multiple areas, they generalize or significantly improve upon the mixed-integer formulations that have been proposed in the literature to solve narrow special cases of our problems.

3. *GCS is high-level for the user.* Formulating a GCS problem requires only basic familiarity with graphs and convex optimization. Then, the reformulation into an MICP and the interplay between the discrete and the continuous components of the problem is handled completely automatically.

The GCS framework has numerous applications, spanning logistics, transportation, scheduling, navigation, computational geometry, and data analysis. In this thesis we will briefly mention some of these, but we will mostly focus on optimal control of dynamical systems and collision-free motion planning of robotic systems. In these two areas, our optimization techniques outperform algorithms that have been developed for decades. We also highlight that the techniques introduced in this thesis are implemented in the software packages `Drake` [183] and `gcspy`. The former is a large and mature software for robotics that is open-source and widely used in academia and industry. The second is a very simple and lightweight Python package, the use of which will be illustrated in this thesis through basic numerical examples.

1.3 Related works

1.3.1 Discrete-continuous optimization and decision-making

We overview existing frameworks for mixed discrete-continuous optimization and decision making, using the three bullet points at the end of Section 1.1 as bases for the comparison.

Mixed-integer optimization

A **Mixed-Integer Program** (MIP) is an optimization problem that involves both discrete (integer) and continuous (real) variables. MIPs have been deeply studied for decades [140, 66], and the techniques developed to solve these problems will form the computational backbone of this thesis. The expressive power of mixed-integer optimization is very large; in fact, most of the problems that we are interested in this thesis can, and will eventually be, formulated as MIPs. On the other hand, as a modelling language, mixed-integer optimization is too low-level for most of the problems that we are interested in. Formulating, e.g., the problems in Figure 1-1 directly as MIPs would be extremely tedious; from the definition of the problem variables to the encoding of all the necessary constraints that couple the continuous and discrete choices. In addition, formulating an MIP is very error prone: there are typically many ways in which a given problem can be formulated as an MIP, and the efficiency of the formulation is highly sensitive to small modeling choices. Understanding these interactions requires experience and a deep knowledge of the downstream algorithms that are used to solve the MIP. Although our framework will eventually call an MIP solver, our goal is to provide the users with a higher-level interface that shields them from any concern regarding the efficiency of the underlying MIP, which is constructed automatically.

Satisfiablity

Boolean satisfiability problems have also been interfaced to continuous (convex) optimization, for example in [171]. These frameworks can effectively solve problems that involve complex logical implications between continuous variables and constraints. However, similarly to mixed-integer optimization, these languages can be too low-level for most users, and they require a significant modeling effort and experience.

Hybrid dynamical systems

Hybrid systems are dynamical systems that exhibit both discrete and continuous behaviors [26, 12, 122, 187, 81]. Many problems that involve a mix of discrete and continuous decision making can be seen as optimal-control problems for hybrid systems. On the other hand, general hybrid-system models can sometimes be too detailed and unstructured for efficient optimization, and their control methods predominantly leverage analytical techniques over numerical ones [117, 165].

Special classes of discrete-time hybrid systems that are amenable to numerical (mixed-integer) optimization include: Mixed Logical-Dynamic (MLD) systems, PieceWise-Affine (PWA) systems, and Switched-Affine (SA) systems. (See [12] and [23, Chapter 16] for more complete lists.) MLD systems are constrained linear systems whose states and control inputs have mixed real and integer values. They have great modelling power, but, from a computational point of view, they have similar drawbacks to mixed-integer optimization: they are a low-level description of the system dynamics, and their computation times can be sensitive to small modelling choices. PWA and SA systems are described in terms of families of affine dynamics that are activated depending on the current state and control action. Although, under mild assumptions, their representation power is the same as MLD [92], these are more high-level. Mainly because they do not require the user to explicitly encode the discrete relations between the continuous states and control inputs. This is taken care of in the transcription as an MIP, which can be done automatically with efficient methods [137, 129].

Computation times are the main limitation to a widespread application of mixed-integer optimization for the control of the hybrid systems above [139, 174, 130]. In Chapter 10 we will see that optimal-control problems for PWA and SA systems are naturally formulated as SPPs in GCS. The techniques introduced in this thesis recover the results in [137, 129] as special cases, and also lead to novel mixed-integer formulations that, for certain problems, are orders of magnitude faster to solve.

1.3.2 Graph problems

The GCS framework shares multiple similarities with problems that have been studied before in the fields of graph theory and combinatorial optimization. Here we briefly describe these works, highlighting their similarities and differences from the GCS framework.

Network-based convex optimization

Identical structures to the our GCSs have been proposed to model convex optimization problems in the software packages `SnapVX` [88] and `Plasmo.jl` [97]. These works deal with purely continuous optimization problems, where the sparsity pattern of the objective function and constraints is modelled by a graph. This graph structure facilitates the modelling of a problem and is leveraged by the underlying optimization algorithm. Our work adds an extra layer of complexity, since we pose combinatorial problems over these graphs. First we must select a discrete subgraph among a certain family (e.g., paths, tours, or spanning trees), and then we have to optimize the continuous variables paired with the vertices in this subgraph. In other words, our problem simplifies to the one in [88, 97] when the only subgraph that we can select is the whole graph.

Graph problems with neighborhoods

Graph problems where the vertices are allowed to move within corresponding sets are often called problems with neighborhoods. The TSP, the MSTP, and the **Facility-Location Problem** (FLP) are three of the combinatorial problems that have been studied most extensively in their variants with neighborhoods [4, 193, 27]. Exact algorithms for these problems generally rely on expensive mixed-integer nonconvex optimization [76, 19], and are mostly limited to neighborhoods in two or three dimensions. GCS problems are a large superclass of these problems with neighborhoods, and the techniques we introduce in this thesis can easily scale to high dimensions and graphs with hundreds or thousands of vertices and edges.

The SPP in GCS will be a major focus of this thesis, and its special case with neighborhoods has not been studied as extensively as the problems above. The SPP with neighborhoods has been analyzed in [53] under stringent assumptions that ensure polynomial-time solvability: the sets are disjoint rectilinear polygons in the plane, and the edge lengths penalize the $\mathcal{L}_1$ distance between the vertices. The applications we target with this thesis, however, do not satisfy any of these hypotheses. A well-studied special case of the SPP with neighborhoods is the touring-polygon problem which, in its unconstrained version, requires finding the shortest path between two points that visits a set of polygons in a given order [54]. In case of convex polygons, this problem is easily solved using convex optimization, whereas, for nonconvex polygons, it is NP-hard [54, Theorem 6]. The SPP in GCS differs from this problem in that the programs paired with the GCS vertices are convex, and the order in which we visit them is not predefined. Similar in spirit are also some classical problems in

computational geometry: the safari [142], the zoo-keeper [191], and the watchman-route [36] problems. (See also [114, Part IV] and the references therein.)

An important application of graph problems with neighborhoods is robot sensory coverage [38, 73, 20, 29]. Exact formulations of these problems are also based on mixed-integer nonconvex optimization, and are limited to relatively small coverage tasks. Therefore, in practice, these problems are solved approximately [29]. The techniques we propose in this thesis have the potential of significantly extending the reach of exact methods also in this area.

Graph problems with clusters

Generalized Steiner problems [55], or generalized network-design problems [61, 156], can be thought of as the discrete counterpart of the problems with neighborhoods. The vertex set is partitioned into clusters and the problem constraints are expressed in terms of these clusters, rather than the original vertices. For example, in the TSP with clusters [141, 64] we seek a shortest tour that visits each cluster of vertices at least or exactly once. Analogous generalizations have been studied for many other problems (e.g., the MSTP [138, 56, 62], the vehicle routing problem [79], and graph coloring [115, 48]). A clustered version of the SPP has been presented in [116]: each vertex in the graph is assigned a nonnegative weight, and the total vertex weight incurred by the shortest path within each cluster must not exceed a given value. In the same paper, a pseudo-polynomial algorithm based on Dijkstra's algorithm [52] is proposed for the solution of this problem.

The problems that we analyze in this thesis can be approximated as graph problems with clusters in a natural way. In low-dimensional spaces, this approximation can be computationally efficient and sufficiently accurate for many practical applications. However, this strategy is infeasible in high dimensions, where covering a volume of space with a cluster requires an exponential number of points.

Euclidean shortest paths

Another problem that is close in spirit to the SPP in GCS is the Euclidean SPP [114]. In this problem we seek a continuous path that connects two points and does not collide with a collection of polygonal obstacles. In two dimensions, the shortest path is a polygonal line whose corners are vertices of the obstacles. By constructing a visibility graph, the problem is then reduced to a discrete graph search and solvable in polynomial time [121, 113]. In three dimensions or more this strategy breaks; in fact, the problem becomes NP-hard [32, Theorem 2.3.2]. An approximation algorithm

for the three-dimensional case has been proposed [145]. Practical algorithms for the multidimensional case based on a grid-discretization of the space have been considered in [185, 107]. More recently, exact-geometry algorithms for problems of this nature have been discussed in [49], and a moment-based technique for computing Euclidean shortest paths in case of semialgebraic obstacles has been proposed in [59].

A main difference between the Euclidean SPP and the SPP in GCS is that in the first problem we seek a continuous path, while in the second we optimize a finite set of points. However, we will see in Chapter 11 that, given a decomposition of the obstacle-free space into convex sets, the Euclidean SPP can be reduced to the SPP in GCS exactly. It is important also to highlight that our techniques are not limited to Euclidean distances but only assume convexity of the problem cost functions. As an example, this allows us to define the distance between two points as the energy consumed by a dynamical system to move between them (see the examples in Chapter 10).

1.3.3 Robot motion planning

Collision-free robot motion planning will be one of the main applications considered in this thesis. The efficient design of trajectories that avoid obstacles is a crucial challenge in robot autonomy. From quadrotors to robot arms, from autonomous cars to legged robots, almost every modern robotic system relies on a collision-avoidance planner to move in its environment. We briefly discuss the techniques currently used to solve these problems, and how the proposed framework improves on them.

Numerical optimization offers mature tools for motion planning in high-dimensional spaces under kinematic and dynamic constraints (see, e.g., [17, 5, 180, 157, 168, 147, 195] among the many works in this area). However, by transcribing the planning problem as a nonconvex optimization, and by relying on local solvers, these techniques can often fail in finding a feasible trajectory if there are obstacles in the environment. Strategies to overcome local optima in trajectory optimization exist, and include sampling trajectories from probability distributions [198, 100, 106], constructing trajectory libraries offline [176, 182, 14, 123, 126], and breaking down the problem into a sequence of subproblems of increasing difficulty [125, 35, 28]. Although these strategies can be effective for many problems, roboticists typically prefer sampling-based planners when facing cluttered environments [105, 112]. Sampling-based algorithms are probabilistically complete, meaning that, if a feasible obstacle-free trajectory exists, they will eventually find one, regardless of the number of obstacles. However, even using asymptotically optimal sampling-based planners [102, 103, 74, 98], the trajectories that we design can be considerably suboptimal in high dimensions, where

dense sampling is infeasible. In addition, even though many of these algorithms support kinodynamic constraints [101, 190, 82], continuous differential constraints are difficult to impose on discrete samples, making these kinodynamic variants less successful in practice.

The promise of the planners based on mixed-integer optimization [166, 161, 136, 47] is to take the best of the two worlds above: the completeness of sampling-based algorithms, and the ease with which trajectory optimization handles the robot kinematics and dynamics, with the added bonus of global optimality. However, the spread of these techniques has been severely limited by their runtimes, which even for small problems can be on the order of minutes.

Convex optimization [25] is commonly viewed as a tool unsuitable for designing trajectories around obstacles. On the one hand, this belief is well grounded given the computational hardness of exact collision-free motion planning [160, 31]. On the other hand, these negative results do not exclude the possibility of approximate motion planners based on convex optimization that perform very well on the problems we typically encounter in practice. The motion-planning method proposed in this thesis is natively a mixed-integer algorithm. However, our MIPs are dramatically more efficient than the ones proposed before: the convex relaxation of our programs is very tight, and a cheap rounding of their solution is typically sufficient to design globally-optimal trajectories. This reduces the MIP to a cheap convex program, and automatically provides optimality bounds for the planned trajectories.

Task and motion planning

The problem of blending high-level discrete logic and low-level continuous motion planning has been deeply studied by the artificial-intelligence community, under the name of **Task And Motion Planning** (TAMP). The techniques developed in this area are highly scalable, and have been used to successfully solve a variety of robotics problems [75]. Although optimization methods are widely used within TAMP algorithms [184], the TAMP formalism does not yield a simple mathematical abstraction of our problems. In addition, existing solution methods for TAMP do not consider the discrete-continuous problem as a whole, but are complex software stacks of high-level logic planners and low-level motion planners [67]. The techniques presented in this thesis can potentially lead to simpler and more effective optimization methods for solving TAMP problems.

1.4 Thesis structure

This thesis is composed of three parts. In the first part, we review the necessary background in convex optimization (Chapter 2), mixed-integer optimization (Chapter 3), and graphs (Chapter 4). In the second part, we describe the GCS framework and the formulation of our MICP (Chapter 5). Chapter 6 illustrates multiple real-world applications spanning logistics, transportation, scheduling, navigation, and computational geometry. Chapter 7 describes `gcspy`, an open-source Python implementation of the methods introduced in this thesis. In Chapter 8, we analyze in more abstract terms the technique used to formulate our MICP, and we connect our approach to existing relaxation techniques for nonconvex optimization. The third part of this thesis is focused on the SPP in GCS (Chapter 9), and its applications to optimal control of dynamical systems (Chapter 10) and collision-free motion planning (Chapter 11). We will see that in optimal control our techniques generalize existing approaches to formulate control problems as MIPs, and also provide novel formulations that, for some problems, can be orders of magnitude faster to solve. In motion planning our methods outperform algorithms that have been developed for decades and are widely used in academia and industry.

Part I

Background

Chapter 2

Convex analysis and optimization

In this chapter we collect some basic concepts from convex analysis and convex optimization. We will only cover the material that is necessary for the upcoming chapters. Most of the results on convex analysis that we report can also be found in [162]. Other precious resources on this topic are [94, 15]. For the background on convex optimization we point the reader to [25, 13].

2.1 Sets

A set $\mathcal{K} \subseteq \mathbb{R}^n$ is said to be a **cone** if for all $\boldsymbol{x} \in \mathcal{K}$ and $y > 0$ we have

$$y\boldsymbol{x} \in \mathcal{K}.$$

In other words, we say that $\mathcal{K}$ is a cone if $y\mathcal{K} := \{y\boldsymbol{x} : \boldsymbol{x} \in \mathcal{K}\} \subseteq \mathcal{K}$ for all $y > 0$. Note that if $y > 0$ and $\mathcal{K}$ is a cone, then $1/y > 0$ and $(1/y)\mathcal{K} \subseteq \mathcal{K}$. We can then multiply both sides of the last inclusion by y to obtain $\mathcal{K} \subseteq y\mathcal{K}$. We conclude that a cone $\mathcal{K}$ verifies the equality

$$y\mathcal{K} = \mathcal{K}$$

for all $y > 0$.

A set $\mathcal{C} \subseteq \mathbb{R}^n$ is **convex** if the line segment connecting any two points in $\mathcal{C}$ is also contained in $\mathcal{C}$. In formulas, $\mathcal{C}$ is convex if for all $\boldsymbol{x}_0, \boldsymbol{x}_1 \in \mathcal{C}$ and $y \in [0, 1]$ we have

$$(1 - y)\boldsymbol{x}_0 + y\boldsymbol{x}_1 \in \mathcal{C}.$$

Using the set notation, the set $\mathcal{C}$ is convex if it satisfies

$$(1 - y)\mathcal{C} + y\mathcal{C} = \mathcal{C}$$

for all $y \in [0,1]$, where the plus sign denotes the Minkowski addition of two sets. Note that the intersection of a (potentially infinite and even uncountable) collection of convex sets is convex. Also any affine transformation of a convex set (e.g., the projection onto a lower dimensional space) is convex.

The following examples of convex cones will be particularly important for us:

- The **origin** $\mathbb{O}^n := \{\mathbf{0} \in \mathbb{R}^n\}$.
- The **nonnegative orthant** $\mathbb{R}^n_{\geq 0} := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{x} \geq \mathbf{0}\}$.
- The **second-order cone** (or **Lorentz cone**) $\mathbb{L}_n := \{(\boldsymbol{x}, y) \in \mathbb{R}^n : \|\boldsymbol{x}\|_2 \leq y\}$.
- The **semidefinite cone** $\mathbb{S}_n := \{\boldsymbol{X} \in \mathbb{R}^{n \times n} : \boldsymbol{X} = \boldsymbol{X}^\top, \boldsymbol{a}^\top \boldsymbol{X} \boldsymbol{a} \geq 0 \text{ for all } \boldsymbol{a} \in \mathbb{R}^n\}$ is a convex cone in the space of n-by-n matrices. It is also isomorphic to a convex cone in $\mathbb{R}^{n(n+1)/2}$, where the isomorphism stacks in a vector the entries in the upper (or lower) triangle of the symmetric matrix $\boldsymbol{X} \in \mathbb{S}_n$.

The **convex hull** of a set $\mathcal{S} \subseteq \mathbb{R}^n$, denoted as $\text{conv}(\mathcal{S})$, is the smallest convex set that contains $\mathcal{S}$. Equivalently, it is the intersection of all the convex sets that contain $\mathcal{S}$. We call a point $\sum_{i=1}^m y_i \boldsymbol{x}_i$ with $y_1, \ldots, y_m \geq 0$ and $\sum_{i=1}^m y_i = 1$ a **convex combination** of the points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_m \in \mathbb{R}^n$. The convex hull of a set $\mathcal{S}$ can be verified to be the set of points that can be expressed as convex combinations of points in $\mathcal{S}$. A point $\boldsymbol{x}$ of a convex set $\mathcal{C}$ is said to be an **extreme point** if it cannot be expressed as the convex combination of points $\boldsymbol{x}_0, \ldots, \boldsymbol{x}_m \in \mathcal{C}$ different from $\boldsymbol{x}$. We denote with $\text{ext}(\mathcal{C})$ the set of extreme points of a convex set $\mathcal{C}$. It is easily verified that a compact convex set is equal to the convex hull of its extreme points.

A **polyhedron** is a convex set of the form $\mathcal{P} := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \geq \mathbf{0}\}$ for some matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$ and vector $\boldsymbol{b} \in \mathbb{R}^m$. Geometrically, the polyhedron $\mathcal{P}$ is the intersection of m closed halfspaces. A polyhedron has a finite number of extreme points. A bounded polyhedron is called a **polytope**.

In practice, most of the sets that we encounter in convex optimization are described in **conic form**:

$$\mathcal{C} := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathcal{K}\} \tag{2.1}$$

for some matrix $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, vector $\boldsymbol{b} \in \mathbb{R}^m$, and closed convex cone $\mathcal{K} \subseteq \mathbb{R}^m$. When the cone $\mathcal{K}$ is the origin $\mathbb{O}^m$, we obtain an affine space. When $\mathcal{K}$ is the nonnegative orthant $\mathbb{R}^m_{\geq 0}$, we have a polyhedron. Convex quadratic sets can be represented in conic form by letting $\mathcal{K}$ be the Cartesian product of second-order cones $\mathbb{L}_{m_1}, \ldots, \mathbb{L}_{m_k}$ such that $m_1 + \ldots + m_k = m$. A spectrahedron is a set of the form (2.1) where $\mathcal{K}$ is (isomorphic to) a semidefinite cone.

2.1.1 Homogenization

Every set $\mathcal{S}$ in n dimensions is naturally paired with a cone in $n+1$ dimensions through an operation called **homogenization**:[1]

$$\tilde{\mathcal{S}} := \{(\boldsymbol{x}, y) \in \mathbb{R}^{n+1} : y > 0,\ \boldsymbol{x} \in y\mathcal{S}\}.$$

It is easy to show that the set $\tilde{\mathcal{S}}$ is a cone and is convex if and only if the original set $\mathcal{S}$ is convex. Note also that $\boldsymbol{x} \in \mathcal{S}$ if and only if $(\boldsymbol{x}, 1) \in \tilde{\mathcal{S}}$.

The homogenization $\tilde{\mathcal{S}}$ does not contain the origin, and the cone $\tilde{\mathcal{S}}$ is not closed unless $\mathcal{S}$ is empty. When doing numerical computations, we will always work with the closure $\mathrm{cl}(\tilde{\mathcal{S}})$ of the homogenization. Also, most of the times we will deal with compact sets, for which the closure of $\tilde{\mathcal{S}}$ is simply obtained by including the origin:

$$\mathrm{cl}(\tilde{\mathcal{S}}) = \{(\boldsymbol{x}, y) \in \mathbb{R}^{n+1} : y \geq 0,\ \boldsymbol{x} \in y\mathcal{S}\} = \tilde{\mathcal{S}} \cup \{\mathbf{0}\}. \tag{2.2}$$

Among its many uses, the operation of homogenization allows us to parameterize in a convex fashion the convex hull of the union of a collection of convex sets. The following result was first introduced for polyhedral sets in [9], and extended to convex sets in [34].

Lemma 2.1. *Let $\mathcal{C}_1, \ldots, \mathcal{C}_m$ be compact convex sets. We have*

$$\mathrm{conv}\left(\bigcup_{i=1}^{m} \mathcal{C}_i\right) = \left\{\boldsymbol{x} : (\boldsymbol{x}, 1) = \sum_{i=1}^{m} (\boldsymbol{x}_i, y_i),\ (\boldsymbol{x}_i, y_i) \in \mathrm{cl}(\tilde{\mathcal{C}}_i) \text{ for } i = 1, \ldots, m\right\}. \tag{2.3}$$

Proof. Let us call $\mathcal{C}$ the set on the right-hand side. This set is convex, since it is the orthogonal projection of a convex set in the space of the variables $\boldsymbol{x}$, $\boldsymbol{x}_i$, and y_i. The set $\mathcal{C}$ contains each set $\mathcal{C}_i$, which is recovered by fixing $y_i = 1$ in the definition of $\mathcal{C}$. Therefore $\mathcal{C}$ contains the convex hull of the union of the sets $\mathcal{C}_i$. For the reverse inclusion, we note that any point $\boldsymbol{x} \in \mathcal{C}$ can be expressed as $\boldsymbol{x} = \sum_{i \in \mathcal{I}} y_i \boldsymbol{x}'_i$, where $\mathcal{I}$ is the set of i such that $y_i > 0$ and $\boldsymbol{x}'_i := \boldsymbol{x}_i / y_i$. Since $\boldsymbol{x}_i \in y_i \mathcal{C}_i$ we have that $\boldsymbol{x}'_i \in \mathcal{C}_i$. Therefore $\boldsymbol{x}$ is a convex combination of points in the sets $\mathcal{C}_i$, and $\mathcal{C}$ is contained in the convex hull of the union of these sets. □

The homogenization of a convex set $\mathcal{C}$ described in conic form is computed very easily. In fact, for $y > 0$, we observe that

$$y\mathcal{C} = \{y\boldsymbol{x} : \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathcal{K}\} = \{y\boldsymbol{x} : \boldsymbol{A}(y\boldsymbol{x}) + \boldsymbol{b}y \in y\mathcal{K}\} = \{\boldsymbol{x}' : \boldsymbol{A}\boldsymbol{x}' + \boldsymbol{b}y \in \mathcal{K}\},$$

[1] The term homogenization is used, e.g., in [197, Definition 1.13]. Sometimes this is also called **perspective** [94, Section IV.2.2] or **conic hull** [13, Section 3.3].

and this gives us

$$\tilde{\mathcal{C}} = \{(\boldsymbol{x}, y) : y > 0,\ \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}y \in \mathcal{K}\}. \tag{2.4}$$

In addition, it is also easily verified that

$$\mathrm{cl}(\tilde{\mathcal{C}}) = \{(\boldsymbol{x}, y) : y \geq 0,\ \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}y \in \mathcal{K}\}. \tag{2.5}$$

The expressions (2.4) and (2.5) have great practical relevance. Roughly speaking, they tell us that if we can efficiently do computations with a set $\mathcal{C}$ described in conic form, then the same is true for (the closure of) its homogenization $\tilde{\mathcal{C}}$.

2.1.2 Dual and polar cones

Let $\mathcal{K} \subseteq \mathbb{R}^n$ be a cone. The set

$$\mathcal{K}^* := \{\boldsymbol{a} : \boldsymbol{x}^\top \boldsymbol{a} \geq 0 \text{ for all } \boldsymbol{x} \in \mathcal{K}\}$$

is called the **dual cone** of $\mathcal{K}$. Since $\mathcal{K}^*$ is the intersection of (infinitely many) closed half-spaces, it is closed and convex, even if the original cone $\mathcal{K}$ is neither closed nor convex. Note that the dual cone is unchanged if we take the closure or the convex hull of $\mathcal{K}$. It can also be seen that by taking the dual of a cone twice we obtain the closure of the convex hull of the original cone:

$$\mathcal{K}^{**} = \mathrm{cl}(\mathrm{conv}(\mathcal{K})). \tag{2.6}$$

Therefore, if the cone $\mathcal{K}$ is closed and convex, then we have $\mathcal{K}^{**} = \mathcal{K}$.

We say that a linear inequality $\boldsymbol{a}^\top \boldsymbol{x} + b \geq 0$ is **valid** for a set $\mathcal{S} \subseteq \mathbb{R}^n$ if it is satisfied by all the points $\boldsymbol{x} \in \mathcal{S}$. Valid inequalities give us a second cone in $n+1$ dimensions that is naturally associated to a set in n dimensions. This cone is called the **polar** of $\mathcal{S}$ and contains the coefficients of all the valid inequalities:[2]

$$\mathcal{S}^\circ := \{(\boldsymbol{a}, b) : \boldsymbol{a}^\top \boldsymbol{x} + b \geq 0 \text{ for all } \boldsymbol{x} \in \mathcal{S}\}.$$

Similarly to the dual cone, the polar is always closed and convex, even when $\mathcal{S}$ is not.

The cones constructed by the operations of homogenization and polar are connected by the following lemma.

[2]The name polar is a little unusual here: the polar is typically a set in n dimensions.

Lemma 2.2. *For a set $\mathcal{S} \subseteq \mathbb{R}^n$, the polar $\mathcal{S}^\circ$ and the closure of the convex hull of the homogenization* $\mathrm{cl}(\mathrm{conv}(\tilde{\mathcal{S}}))$ *are dual cones to each other.*

This lemma is immediately implied by multiple fundamental results in convex analysis. In Section 2.4.1 below we give an elementary proof that does require any prior knowledge.

As a corollary of Lemma 2.2, we have that a closed convex set $\mathcal{C}$ is such that

$$\mathrm{cl}(\tilde{\mathcal{C}}) = (\mathcal{C}^\circ)^* = \{(\boldsymbol{x}, y) : \boldsymbol{a}^\top \boldsymbol{x} + by \geq 0 \text{ for all } (\boldsymbol{a}, b) \in \mathcal{C}^\circ\}. \tag{2.7}$$

Also, by slicing the sets above with the hyperplane $\{(\boldsymbol{x}, y) : y = 1\}$, we obtain the well-known fact that a closed convex set is the intersection of all the closed halfspaces that contain it:

$$\mathcal{C} = \{\boldsymbol{x} : \boldsymbol{a}^\top \boldsymbol{x} + b \geq 0 \text{ for all } (\boldsymbol{a}, b) \in \mathcal{C}^\circ\}. \tag{2.8}$$

2.2 From sets to functions

Let $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ be a function. The **epigraph** of f is the set of points that lie above the graph of f:

$$\mathrm{epi}(f) := \{(\boldsymbol{x}, y) \in \mathbb{R}^{n+1} : f(\boldsymbol{x}) \leq y\}.$$

Note that the epigraph does not "cover" the points $\boldsymbol{x}$ where f takes infinite value. Through this mapping, any function in n variables is uniquely associated with a set in $n + 1$ dimensions.

Conversely, to retrieve a function in n variables from a set $\mathcal{S} \in \mathbb{R}^{n+1}$, we define the operation

$$\mathrm{ipo}(\mathcal{S})(\boldsymbol{x}) := \inf\{y : (\boldsymbol{x}, y) \in \mathcal{S}\}, \tag{2.9}$$

where the infimum of an empty set is defined to be infinite. Pictorially, the graph of the function $\mathrm{ipo}(\mathcal{S})$ is the "lower boundary" of the set $\mathcal{S}$.[3] If the set $\mathcal{S}$ is convex, then the function $\mathrm{ipo}(\mathcal{S})$ is also convex.

The equality

$$f = \mathrm{ipo}(\mathrm{epi}(f))$$

[3]The operation in (2.9) is widely used in convex analysis, but to the best of our knowledge it does not have a commonly accepted name. Here the word ipo (Greek prefix for "below") is meant to contrast the word epi (Greek prefix for "above").

holds for any function f. Therefore the function ipo is the left inverse of the function epi. Conversely, if we start from a set $\mathcal{S}$, we can have that $\mathcal{S} \neq \text{epi}(\text{ipo}(\mathcal{S}))$, since the function $\text{ipo}(\mathcal{S})$ loses all the information about the "upper boundary" of $\mathcal{S}$.

The properties of a function f are inherited by the properties of its epigraph. A function f is said to be **closed** if its epigraph is a closed set. While the closure of a function is defined as

$$\text{cl}(f) := \text{ipo}(\text{cl}(\text{epi}(f))).$$

In words, we construct the epigraph of f, we take its closure, and then we map this closed set back to a closed function.

We say that a function $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is **positively homogeneous** if its epigraph is a cone. This implicit definition is easily seen to be equivalent to the following explicit definition: a function f is positively homogeneous if for all $y > 0$ we have

$$yf(\boldsymbol{x}) = f(y\boldsymbol{x}).$$

Note that the closure of any positively homogeneous function has value equal to zero at the origin.

The function f is called **convex** if its epigraph is convex. The equivalent explicit definition is that f is convex if, for all $\boldsymbol{x}_0$, $\boldsymbol{x}_1$ and $y \in [0, 1]$, we have

$$f((1-y)\boldsymbol{x}_0 + y\boldsymbol{x}_1) \leq (1-y)f(\boldsymbol{x}_0) + yf(\boldsymbol{x}_1).$$

In words, every line segment connecting two points on the graph of f lies entirely above the graph of f. This formula is easily extended to convex combinations that involve more than two points. Given scalars $y_i \geq 0$ such that $\sum_{i=1}^m y_i = 1$, we have

$$f\left(\sum_{i=1}^m y_i \boldsymbol{x}_i\right) \leq \sum_{i=1}^m y_i f(\boldsymbol{x}_i). \tag{2.10}$$

Note that a convex function that takes only finite values $f : \mathbb{R}^n \to \mathbb{R}$ is necessarily closed.

The **homogenization** of a function f is constructed similarly to the closure:

$$\tilde{f} := \text{ipo}\left(\widetilde{\text{epi}(f)}\right).$$

We lift the function to its epigraph, we homogenize, and we take the infimum to retrieve a function.[4] Note that $\tilde{f}$ is a function in $n+1$ variables, since the homoge-

[4]To be more precise, this construction necessitates reordering the variables, as the outer infimum is with respect to the variable introduced by the epigraph, not the homogenization.

nization increases the dimension of the space by one. The function $\tilde{f}$ is convex if and only if f is convex. In addition, $\tilde{f}$ is positively homogeneous, since its epigraph is a cone. An equivalent explicit definition of the homogenization of a function is

$$\tilde{f}(\boldsymbol{x}, y) := \begin{cases} yf(\boldsymbol{x}/y) & \text{if } y > 0 \\ \infty & \text{otherwise} \end{cases}.$$

Since this operation is not very common, for completeness, we report in Section 2.4.2 below a proof of the equivalence of the two definitions of homogenization of a function.

Similarly to the homogenization of a set, in practice, we always work with the closure of $\tilde{f}$. If the original function f is closed, by taking the closure we change the value of the homogenization $\tilde{f}$ only for $y = 0$. In particular, we ensure that

$$\mathrm{cl}(\tilde{f})(\mathbf{0}, 0) = 0. \tag{2.11}$$

The values $\mathrm{cl}(\tilde{f})(\boldsymbol{x}, 0)$ for $\boldsymbol{x} \neq 0$ are a little more complicated to describe, but also irrelevant to the results in this thesis.

A set $\mathcal{C}$ in conic form (2.1) is said to be a **conic representation** of a function f if $\mathrm{epi}(f) = \mathcal{C}$. We will also say that a function f is described in **conic form** if we have a conic representation of it. In convex optimization, we often work with conic representations of functions instead than with the explicit descriptions of them. Important for this thesis is the observation that if we are given a conic representation of a function f then a conic representation of $\tilde{f}$ is easily computed using the formula (2.4). While (2.5) can be used to describe $\mathrm{cl}(\tilde{f})$ in conic form.

2.3 Convex optimization

A **Convex Program** (CP) is an optimization problem of the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}) && \text{(2.12a)} \\ \text{subject to} \quad & \boldsymbol{x} \in \mathcal{C}, && \text{(2.12b)} \end{aligned}$$

where the objective function $f : \mathbb{R}^n \to \mathbb{R}$ and the constraint set $\mathcal{C} \subset \mathbb{R}^n$ are convex. Sometimes we will refer to the set $\mathcal{C}$ as the **feasible set** of the CP, and to its elements as **feasible solutions**. The CP is said to be **feasible** if a feasible solution exists, i.e., if $\mathcal{C} \neq \emptyset$. A feasible solution $\boldsymbol{x}_{\mathrm{opt}}$ is **optimal** if $f(\boldsymbol{x}_{\mathrm{opt}}) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x} \in \mathcal{C}$. If an optimal solution exists, we call $f_{\mathrm{opt}} := f(\boldsymbol{x}_{\mathrm{opt}})$ the **optimal value** of the CP.

A fundamental property of CPs is that any locally optimal solution (i.e., any

point $\boldsymbol{x}_{\text{opt}}$ such that $f(\boldsymbol{x}_{\text{opt}}) \leq f(\boldsymbol{x})$ for all $\boldsymbol{x}$ in a neighborhood of $\boldsymbol{x}_{\text{opt}}$) is also an optimal solution according to the (global) definition just given [25, Section 4.2.2]. A commonly satisfied assumption for the existence of an optimal solution is that the feasible set $\mathcal{C}$ is nonempty and compact.

CPs are a fundamental class of optimization problems, with applications in essentially every engineering discipline. The great majority of the CPs that we encounter in practice can be solved efficiently and reliably using interior-point methods (or other specialized algorithms such as the simplex method). However, strictly speaking, it is not always true that a CP can be solved efficiently: for example, the set of nonnegative polynomials is a convex cone (in the space of the polynomial coefficients), but checking if a polynomial is nonnegative is NP-hard. In this thesis we will be a little imprecise, and use the term "convex optimization problem" almost as a synonym of "optimization problem that is efficiently solvable".

2.3.1 Conic optimization

A **Conic Program** (KP) is a CP with linear objective function and constraint set in conic form:

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^\top \boldsymbol{x} && \text{(2.13a)} \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \in \mathcal{K}, && \text{(2.13b)} \end{aligned}$$

where $\boldsymbol{c} \in \mathbb{R}^n$, $\boldsymbol{A} \in \mathbb{R}^{m \times n}$, $\boldsymbol{b} \in \mathbb{R}^m$, and $\mathcal{K} \subseteq \mathbb{R}^m$ is a closed convex cone. Special classes of KPs are:

- **Linear Program** (LP) when $\mathcal{K}$ is the nonnegative orthant.
- **Second-Order Cone Program** (SOCP) when $\mathcal{K}$ is the Cartesian product of second-order cones.
- **SemiDefinite Program** (SDP) when $\mathcal{K}$ is (isomorphic to) the semidefinite cone.

These classes of problems have increasing modelling power: every LP is an SOCP, and every SOCP is an SDP. SDPs are efficiently solvable, and cover the vast majority of the practical uses of convex optimization.

Another important class of CPs are **Quadratic Programs** (QP), these are CPs in the form (2.12) with quadratic objective function f and polyhedral constraint set $\mathcal{C}$. Every QP can be formulated as an SOCP. However, in many cases, active-set

algorithms specialized to this class of problems can be more effective than using an SOCP solver.

2.3.2 Conic duality

The KP (2.13) is associated to the following dual program

$$\begin{aligned} \text{maximize} \quad & -\boldsymbol{b}^\top \boldsymbol{\lambda} && (2.14)\\ \text{subject to} \quad & \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{c}, && (2.15)\\ & \boldsymbol{\lambda} \in \mathcal{K}^*, && (2.16) \end{aligned}$$

where the variable is $\boldsymbol{\lambda} \in \mathbb{R}^m$. Note that the dual program is also a KP.

The optimal value of the primal problem (2.13) is greater than or equal to the optimal value of the dual problem (2.14). In fact,

$$\boldsymbol{c}^\top \boldsymbol{x} - (-\boldsymbol{b}^\top \boldsymbol{\lambda}) = \boldsymbol{x}^\top \boldsymbol{c} + \boldsymbol{b}^\top \boldsymbol{\lambda} = \boldsymbol{x}^\top \boldsymbol{A}^\top \boldsymbol{\lambda} + \boldsymbol{b}^\top \boldsymbol{\lambda} = (\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b})^\top \boldsymbol{\lambda} \geq 0,$$

where the last inequality uses the duality of $\mathcal{K}$ and $\mathcal{K}^*$. This property is called **weak duality**.

Strong duality holds when the primal problem (2.13) and the dual problem (2.14) have exactly the same optimal value. For this to be true we need to make further assumptions on the problem data. For example, a sufficient condition for strong duality to hold are the so-called Slater conditions. These require both the primal and the dual problem to be strictly feasible. For the primal this means that there exists $\boldsymbol{x}$ such that $\boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}$ is in the interior of $\mathcal{K}$. For the dual we require the existence of $\boldsymbol{\lambda}$ in the interior of $\mathcal{K}^*$ such that $\boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{c}$.

The following result is well known, and will be useful in a few occasions. We give a proof of it based on duality in Section 2.4.3 below.

Lemma 2.3. *Let $\mathcal{P} := \{\boldsymbol{x} \in \mathbb{R}^n : \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b} \geq \boldsymbol{0}\}$ be a polyhedron. For any valid inequality $(\boldsymbol{c}, d) \in \mathcal{P}^\circ$ there exists a vector $\boldsymbol{\lambda} \in \mathbb{R}^m_{\geq 0}$ such that $\boldsymbol{c} = \boldsymbol{A}^\top \boldsymbol{\lambda}$ and $d \geq \boldsymbol{b}^\top \boldsymbol{\lambda}$.*

2.3.3 Disciplined convex programming

In order to solve a CP it is not sufficient to write it symbolically on a computer and send it to a solver. In fact, solvers for convex optimization can typically handle only a limited family of conic constraints, and it is our job to transform generic convex functions and sets in these canonical forms. **Disciplined Convex Programming** (DCP) [85] is an effective way to automatize these transformations. It is a modular

and algorithmic framework for assembling convex optimization problems. The idea is that the user must follow a restricted set of rules and methods to assemble a CP; then the DCP software automatically certifies the convexity of the problem and communicates with the solver. Popular software packages for DCP are `cvx` [84], `cvxpy` [50], and `Convex.jl` [186].

More precisely, the DCP framework has two main components [85, Section 5]:

- *A library of atoms.* This is a collection of functions and sets whose properties are known and explicitly declared.
- *A convexity ruleset.* This governs how the atoms, the variables, the parameters, and the numeric values in the optimization problem can be combined to produce convex results.

A disciplined CP is an optimization problem built in accordance with the convexity ruleset using elements from the atom library.

In this thesis, we will make large use of the homogenization operations defined above. Given a generic convex set $\mathcal{C}$ or convex function f, computing the homogenization ($\tilde{\mathcal{C}}$ or $\tilde{f}$) can be very tedious and error prone. Using DCP this process is fully automatic: any DCP-compliant set or function is automatically described in conic form (2.1), then computing the homogenization can be easily done using (2.4).

2.4 Supporting proofs

We collect in this final section the proofs that are not necessary for the main body of the chapter.

2.4.1 Duality of homogenization and polar

In Section 2.1.2 we have stated that for a set $\mathcal{S} \subseteq \mathbb{R}^n$ the cones $\mathcal{S}^\circ$ and $\mathrm{cl}(\mathrm{conv}(\tilde{\mathcal{S}}))$ are dual to each other. We give here a basic proof of this statement.

Since the cones $\mathcal{S}^\circ$ and $\mathrm{cl}(\mathrm{conv}(\tilde{\mathcal{S}}))$ are closed and convex, it suffices to show that the first is the dual of the second. In turn, this is equivalent to show that

$$\mathcal{S}^\circ = \tilde{\mathcal{S}}^*,$$

where the closure and the convex hull are omitted since they do not affect the dual cone. We show mutual inclusion of the two sets above.

Let $(\boldsymbol{a}, b) \in \mathcal{S}^\circ$. By definition, we have $\boldsymbol{a}^\top \boldsymbol{x} + b \geq 0$ for all $\boldsymbol{x} \in \mathcal{S}$. We multiply this inequality by $y > 0$ and have that $\boldsymbol{a}^\top (y\boldsymbol{x}) + by \geq 0$ for all $y > 0$ and $\boldsymbol{x} \in \mathcal{S}$. We

define $\boldsymbol{x}' := y\boldsymbol{x}$ and conclude that $\boldsymbol{a}^\top \boldsymbol{x}' + by \geq 0$ for all $y > 0$ and $\boldsymbol{x}' \in y\mathcal{S}$, i.e., for all $(\boldsymbol{x}', y) \in \tilde{\mathcal{S}}$. This shows that $(\boldsymbol{a}, b) \in \tilde{\mathcal{S}}^*$ and, therefore, $\mathcal{S}^\circ \subseteq \tilde{\mathcal{S}}^*$.

Assume that $(\boldsymbol{a}, b) \in \tilde{\mathcal{S}}^*$. We have $\boldsymbol{a}^\top \boldsymbol{x} + by \geq 0$ for all $(\boldsymbol{x}, y) \in \tilde{\mathcal{S}}$, i.e., for all $y > 0$ and $\boldsymbol{x} \in y\mathcal{S}$. Defining $\boldsymbol{x}' := \boldsymbol{x}/y$, we have $\boldsymbol{a}^\top \boldsymbol{x}' + b \geq 0$ for all $\boldsymbol{x}' \in \mathcal{S}$. This shows that $(\boldsymbol{a}, b) \in \mathcal{S}^\circ$ and $\tilde{\mathcal{S}}^* \subseteq \mathcal{S}^\circ$.

2.4.2 Homogenization of a function

In Section 2.2 we have given two definitions of the homogenization of a function f. The first is $\tilde{f} = \text{ipo}\left(\widetilde{\text{epi}(f)}\right)$. The second is $\tilde{f}(\boldsymbol{x}, y) = yf(\boldsymbol{x}/y)$ if $y > 0$ and $\tilde{f}(\boldsymbol{x}, y) = \infty$ otherwise. Let us show that these definitions are in fact equivalent.

For any $y > 0$, we have

$$\begin{aligned} y\text{epi}(f) &= \{y(\boldsymbol{x}, y') : f(\boldsymbol{x}) \leq y'\} \\ &= \{(y\boldsymbol{x}, yy') : yf(\boldsymbol{x}) \leq yy'\} \\ &= \{(\boldsymbol{x}', y'') : yf(\boldsymbol{x}'/y) \leq y''\}. \end{aligned}$$

Using this equality, we have

$$\widetilde{\text{epi}(f)} = \{(\boldsymbol{x}, y', y) : y > 0,\ (y', \boldsymbol{x}) \in y\text{epi}(f)\} = \{(\boldsymbol{x}, y', y) : y > 0,\ yf(\boldsymbol{x}/y) \leq y'\}.$$

We can then conclude that the two definitions are equivalent:

$$\text{ipo}\left(\widetilde{\text{epi}(f)}\right)(\boldsymbol{x}, y) = \inf\{y' : y > 0,\ yf(\boldsymbol{x}/y) \leq y'\} = \begin{cases} yf(\boldsymbol{x}/y) & \text{if } y > 0 \\ \infty & \text{otherwise} \end{cases}.$$

2.4.3 Implied valid inequalities

We give a short proof of Lemma 2.3 based on LP duality (although this fact could be verified using more basic tools). We want to check whether the following LP is feasible for any valid inequality $(\boldsymbol{c}, d) \in \mathcal{P}^\circ$:

$$\begin{aligned} \text{maximize} \quad & 0 \\ \text{subject to} \quad & \boldsymbol{A}^\top \boldsymbol{\lambda} = \boldsymbol{c}, \\ & \boldsymbol{b}^\top \boldsymbol{\lambda} \leq d. \end{aligned}$$

The dual of this problem is the LP

$$\begin{aligned} \text{minimize} \quad & \boldsymbol{c}^\top \boldsymbol{x} + dy \\ \text{subject to} \quad & \boldsymbol{A}\boldsymbol{x} + \boldsymbol{b}y \geq 0, \\ & y \geq 0. \end{aligned}$$

The dual LP is always feasible, since $\boldsymbol{x} = \boldsymbol{0}$ and $y = 0$ is a feasible solution. Thus, by strong duality, the primal LP is feasible if and only if the dual LP has finite optimum. The constraints in the dual LP are equivalently rewritten equivalent as $(\boldsymbol{x}, y) \in \mathrm{cl}(\tilde{\mathcal{P}})$. By Lemma 2.2, the sets $\mathcal{P}^\circ$ and $\tilde{\mathcal{P}}^*$ are equal. Therefore, our assumption $(\boldsymbol{c}, d) \in \mathcal{P}^\circ$ is equivalent to $\boldsymbol{c}^\top \boldsymbol{x} + dy \geq 0$ for all $(\boldsymbol{x}, y) \in \mathrm{cl}(\tilde{\mathcal{P}})$. We conclude that the optimal value of the dual LP cannot be negative, and the dual LP has finite optimum $\boldsymbol{x} = \boldsymbol{0}$ and $y = 0$.

Chapter 3

Mixed-integer optimization

Mixed-integer optimization is the computational backbone of the techniques introduced in this thesis. Generally speaking, a **Mixed-Integer Program** (MIP) can be very hard to solve and the runtimes of a mixed-integer solver can grow very quickly (exponentially) with the problem size. However,this is the worst-case scenario, and a lot can be done to construct highly effective MIPs that can be solved quickly for most of (if not all) the instances that we encounter in practice. The goal of this chapter is twofold. First, we introduce the essential background on mixed-integer optimization: what an MIP is, how MIPs are classified, and what algorithms can be used to solve these problems. Secondly, we introduce the notions and the tools necessary to distinguish between efficient and inefficient MIPs.

Two great sources for all the details about mixed-integer optimization are the classical book [140] and the more recent book [42].

3.1 Mixed-integer programs

An MIP is an optimization problem with continuous variables $\boldsymbol{x} \in \mathbb{R}^n$ and discrete (integer) variables $\boldsymbol{y} \in \mathbb{Z}^m$. Given an objective function $f : \mathbb{R}^{n+m} \to \mathbb{R}$ and a constraint set $\mathcal{S} \subseteq \mathbb{R}^{n+m}$, we can state a generic MIP as

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}, \boldsymbol{y}) && \text{(3.1a)} \\ \text{subject to} \quad & (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}, && \text{(3.1b)} \\ & \boldsymbol{y} \in \mathbb{Z}^m. && \text{(3.1c)} \end{aligned}$$

(We explicitly state the constraint $\boldsymbol{y} \in \mathbb{Z}^m$ since, if not specified otherwise, variables of optimization problems will always be assumed to be real valued.) We call the set

$$\mathcal{T} := \mathcal{S} \cap \mathbb{R}^n \times \mathbb{Z}^m$$

feasible set of the MIP (3.1) and its elements $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T}$ **feasible solutions**. The MIP is **feasible** if a feasible solution exists and infeasible otherwise. A feasible solution $(\boldsymbol{x}_{\text{opt}}, \boldsymbol{y}_{\text{opt}})$ is **optimal** if $f(\boldsymbol{x}_{\text{opt}}, \boldsymbol{y}_{\text{opt}}) \leq f(\boldsymbol{x}, \boldsymbol{y})$ for all $(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{T}$. If an optimal solution exists, we denote with $f_{\text{opt}} := f(\boldsymbol{x}_{\text{opt}}, \boldsymbol{y}_{\text{opt}})$ the MIP **optimal value**.

If we discard constrain (3.1c) from our MIP, we obtain the following optimization problem:

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}, \boldsymbol{y}) && \text{(3.2a)} \\ \text{subject to} \quad & (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}. && \text{(3.2b)} \end{aligned}$$

This problem is called the **relaxation** of the MIP. We denote an optimal solution of the relaxation as $(\boldsymbol{x}_{\text{relax}}, \boldsymbol{y}_{\text{relax}})$, and we let $f_{\text{relax}} := f(\boldsymbol{x}_{\text{relax}}, \boldsymbol{y}_{\text{relax}})$ be its optimal value. Observe that

$$f_{\text{relax}} \leq f_{\text{opt}},$$

since by discarding a constraint from an optimization (minimization) problem the optimal value can only decrease. As discussed in Section 3.2 below, the tightness of this inequality plays a central role in the efficiency of an MIP. Loosely speaking, an MIP is efficiently solvable if its relaxation can be solved quickly and the gap $f_{\text{opt}} - f_{\text{relax}} \geq 0$ is small.

MIPs are classified according to the properties of the function f and the set $\mathcal{S}$, which can significantly affect our ability of solving an MIP efficiently. Below we define the classes of MIPs that are most relevant for this thesis.

3.1.1 Mixed-Boolean programs

A **Mixed-Boolean Program** (MBP) is an MIP where the discrete variables can only take binary value: $\boldsymbol{y} \in \{0, 1\}^m$. Equivalently, a MBP is a problem of the form of (3.1) where

$$\mathcal{S} \subset \mathbb{R}^n \times [0, 1]^m.$$

All the MIPs that we will encounter in this thesis are, in fact, MBPs. However, the term MBP is unusual and, although technically our problems will be MBPs, we will still call them as MIPs.

3.1.2 Mixed-integer convex programs

If the objective function f and the constraint set $\mathcal{S}$ are convex, we call problem (3.1) a **Mixed-Integer Convex Program** (MICP). MICPs are a fundamental class of MIPs since their relaxations are convex optimization problems, which (in most of the cases) can be solved very quickly. To emphasize the convexity assumption, we call the relaxation of an MICP **convex relaxation**. Using the **Branch-and-Bound** (BB) algorithm described in Section 3.3.2 below, most MICPs can be reliably solved to global optimality, although the algorithm might take a long time.

In contrast to the class of MICPs, we will occasionally call **Mixed-Integer Non-Convex Program** (MINCP) an MIP where the objective function f and/or the constraint set $\mathcal{S}$ are nonconvex. Most MINCPs are intractable; the main exceptions are very small MINCPs where the feasible set $\mathcal{T}$ can be finely discretized and searched exhaustively.

3.1.3 Mixed-integer conic programs

If the objective function f is linear and the constraint set $\mathcal{S}$ is a closed convex set in conic form, then problem (3.1) is called **Mixed-Integer Conic Program** (MIKP). The subclasses of this family of problems are classified as in Section 2.3. We have a

- **Mixed-Integer Linear Program** (MILP) when the set $\mathcal{S}$ is a polyhedron,
- **Mixed-Integer Second-Order Cone Program** (MISOCP) when $\mathcal{S}$ is convex quadratic,
- **Mixed-Integer SemiDefinite Program** (MISDP) when $\mathcal{S}$ is a spectrahedron.

The relaxations of these problems are named in the natural way. For example, we call **linear relaxation** the relaxation of an MILP.

We could say that MILPs play a more important role in mixed-integer optimization than LPs play in convex optimization. The first reason for this is geometric. Polyhedra are the simplest shape that can enclose finite sets of points. Therefore, as we will also see in the next chapter, they are a natural candidate to model the discrete side of an MIP. The second reason is algorithmic. As discussed in Section 3.3.2 below, the simplex algorithm for linear optimization can be integrated in the BB procedure more efficiently than the interior-point algorithm used for more general conic programs.

A **Mixed-Integer Quadratic Program** (MIQP) is an MIP of the form (3.1) with f quadratic and $\mathcal{S}$ polyhedral. These problems are representable as MISOCPs

but, similar to MILPs, they can be solved more efficiently using specialized BB methods that leverage the polyhedral constraint set. Having said this, nowadays also MISOCPs and MISDPs can be solved quite effectively even with freely available solvers (see, e.g., `Pajarito` [39]).

3.1.4 Integer programs

An **Integer Program** (IP) is an MIP with only integer variables, i.e., $n = 0$. IPs can be classified the same ways as MIPs. For example, an **Integer Linear Program** (ILP) is an IP with linear objective and polyhedral constraint set. We will see in the next chapter that all the graph problems of interest for this thesis are naturally formulated as ILPs.

3.2 What makes a good MIP?

An MIP can be solved using a variety of methods. Some of them are general purpose algorithms, like the BB method in Section 3.3.2. Others are tailored to a specific subclass of MIPs, and leverage particular properties of the objective function f and the constraint set $\mathcal{S}$. However, what almost all the MIP solution algorithms have in common, is that they rely on the relaxation (3.2) to obtain important information about the original MIP. For the efficient solution of an MIP is then fundamental that the relaxation has two characteristics:

- it can be solved solved quickly,
- it approximates the original MIP tightly.

The first requirement is typically met if the relaxation is a conic program such as an LP, an SOCP, or even an SDP. In addition, this program must have a small number of variables and constraints.

One way to quantify the second requirement is to look at the **relaxation gap**. For problems with positive optimal value f_{opt}, we define the relaxation gap as

$$\frac{f_{\text{opt}} - f_{\text{relax}}}{f_{\text{opt}}}.$$

The relaxation gap is nonnegative and, assuming that the relaxation has nonnegative optimal value f_{relax}, it is at most one. A small relaxation gap means that the relaxation is a good approximation of the MIP (a perfect approximation if the relaxation gap is zero). A relaxation gap close to one means that solving the relaxation conveys almost

no information about the optimal solution of the MIP. Of course, the relaxation gap is not a direct way to estimate the efficiency of an MIP, since in order to compute it we must have solved the MIP already. What the relaxation gap gives us is a useful metric to compare two alternative MIPs that model the same underlying problem (two, so called, MIP formulations of the same problem).

Consider the MIP (3.1), and assume that its objective function f is linear. (This can be assumed without loss of generality: if f is nonlinear, we can introduce a slack variable s, add the constraint $s \geq f(\boldsymbol{x}, \boldsymbol{y})$, and minimize s instead.) A second MIP with the same objective function f and different constraint set $\mathcal{S}' \subset \mathbb{R}^{n+m}$ models the same problem as the MIP (3.1) if

$$\mathcal{S}' \cap \mathbb{R}^n \times \mathbb{Z}^m = \mathcal{T}.$$

We then observe that the relaxation gap of the second MIP is lower than or equal to the one of the first if

$$\mathcal{S}' \subseteq \mathcal{S}.$$

In this case, we say that the second MIP formulation is **stronger** than the first. We also note that if the MIP (3.1) satisfies

$$\mathcal{S} = \text{conv}(\mathcal{T}) \tag{3.3}$$

then minimizing a linear function over $\mathcal{T}$ or $\mathcal{S}$ is equivalent (i.e., gives the same optimal value). Therefore, if (3.3) holds, then the relaxation gap is guaranteed to be zero, and this formulation is the strongest possible (a so called **perfect formulation**). Note, however, that a perfect formulation is not necessarily the most efficient, since the set $\mathcal{S}$ might be defined by many constraints and the relaxation might be computationally very expensive.

Stronger formulations can also be constructed by using auxiliary variables, i.e., the second MIP in the comparison above can have a higher-dimensional constraint set $\mathcal{S}'$. In this case, we say that the second formulation is stronger than the first if the projection of $\mathcal{S}'$ onto the space of the original variables $\boldsymbol{x}$ and $\boldsymbol{y}$ is contained in $\mathcal{S}$. In fact, sometimes the convex hull of the feasible set $\mathcal{T}$ can have very complex shape, but it can be efficiently described as the projection of a simple higher-dimensional convex set. A perfect formulation that uses auxiliary variables is called an **extended formulation** [41]. (This term also implicitly implies that the formulation has a number of constraints and variables that is polynomial in the size of the original problem.)

3.3 Solution methods

We conclude this chapter by describing in more details how an MIP can be solved. We discuss two solution methods that will be used many times in this thesis: rounding and BB. The former is a simple heuristic approach. The latter is a more sophisticated algorithm, which is guaranteed to always identify an optimal solution.

3.3.1 Rounding

Rounding entails the following simple steps:

1. We compute $(\boldsymbol{x}_{\text{relax}}, \boldsymbol{y}_{\text{relax}})$ by solving the MIP relaxation.

2. If an entry of the vector $\boldsymbol{y}_{\text{relax}} \in \mathbb{R}^m$ is fractional, we approximate it with an integer value (i.e., we "round" it). This gives us a vector $\boldsymbol{y}_{\text{round}} \in \mathbb{Z}^m$.

3. We return $(\boldsymbol{x}_{\text{relax}}, \boldsymbol{y}_{\text{round}})$ as an estimate of the optimal solution of the MIP with cost $f_{\text{round}} := f(\boldsymbol{x}_{\text{relax}}, \boldsymbol{y}_{\text{round}})$.

Potentially, we can also add an extra step where the relaxation is solved a second time with the additional constraint $\boldsymbol{y} = \boldsymbol{y}_{\text{round}}$. In this case we return the optimal solution $(\boldsymbol{x}_{\text{round}}, \boldsymbol{y}_{\text{round}})$ of this second problem, which has value $f_{\text{round}} := f(\boldsymbol{x}_{\text{round}}, \boldsymbol{y}_{\text{round}})$.

Rounding is not guaranteed to identify the optimal solution of the MIP. In fact, it is not even guaranteed to find a feasible solution. On the other hand, if it finds a feasible solution, then it automatically provides us with the following simple bound on its distance from the optimum:

$$\delta_{\text{opt}} := \frac{f_{\text{round}} - f_{\text{opt}}}{f_{\text{opt}}} \leq \frac{f_{\text{round}} - f_{\text{relax}}}{f_{\text{relax}}} =: \delta_{\text{relax}}, \tag{3.4}$$

where δ_{opt} is the true optimality gap, δ_{relax} is the upper bound on the optimality gap automatically obtained from the rounding, and where we assumed that the optimal value f_{relax} of the relaxation is nonnegative.

General purpose rounding strategies (e.g., round the entries of $\boldsymbol{y}_{\text{relax}}$ to the nearest integer) are typically ineffective. Rounding works especially well when:

- a feasible solution of the MIP can be found easily,

- the problem has some structure that can be leveraged.

For example, as we will see in Section 9.5, if the integer variables parameterize a path through a graph, then many fast graph-search algorithms are natural candidates for

the rounding stage. Note also, that the rounding needs not to be a deterministic process. One common approach is to apply a randomized rounding multiple times and select the rounded solution with the lowest cost.

3.3.2 Branch and bound

BB is a general-purpose optimization algorithm that systematically shrinks its search space around the optimal solution of a problem. The BB search space is composed by multiple sets of simple shape. At each iteration, we select one of these sets and minimize our objective function over it. If this subproblem has optimal value larger than a known upper bound on the optimal value of the original problem, then the selected set cannot contain an optimal solution and is discarded from the BB search space. Otherwise, the set is split into two smaller sets that better approximate the feasible set of the original problem, and the process is repeated.

Let us describe the application of a very basic BB algorithm to the MIP (3.1). For simplicity of presentation, let us assume that either the MIP (3.1) is infeasible or an optimal solution exists. More complicated versions of BB can handle corner cases such as unbounded problems (i.e., feasible problems with no optimal solutions). During the iterations of the BB method, we update three objects:

- An estimate $(\boldsymbol{x}_0, \boldsymbol{y}_0)$ of the optimal solution of the MIP. The initial value of this estimate is irrelevant.
- An upper bound f_0 on the optimal value f_{opt} of the MIP. After a feasible estimate $(\boldsymbol{x}_0, \boldsymbol{y}_0)$ is found, this upper bound will be set to $f_0 := f(\boldsymbol{x}_0, \boldsymbol{y}_0)$. Before that, we simply let $f_0 := \infty$.
- A collection $\mathcal{L}$ of subsets of $\mathbb{R}^{n+m}$. These are portions of the search space where we could potentially find a solution with objective smaller than f_0. Initially, we simply let $\mathcal{L} := \{\mathcal{S}\}$.

At each iteration $i = 1, 2, \ldots$, we apply the following steps:

1. If the collection $\mathcal{L}$ is empty, we have nowhere to look for a new solution. Therefore we return the solution estimate $(\boldsymbol{x}_0, \boldsymbol{y}_0)$ and the corresponding objective value f_0.
2. We take one set out of $\mathcal{L}$, call it $\mathcal{S}_i$, and we solve the subproblem

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}, \boldsymbol{y}) && \text{(3.5a)} \\ \text{subject to} \quad & (\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}_i. && \text{(3.5b)} \end{aligned}$$

If this subproblem is infeasible, we move to the next iteration. Otherwise, we denote the optimal solution and value of this problem as $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ and f_i.

3. If $f_i \geq f_0$, the set $\mathcal{S}_i$ cannot contain an optimal solution of the MIP, and we move to the next iteration. If $f_i < f_0$, we analyze at the entries of the vector $\boldsymbol{y}_i$.

4. If al the entries of $\boldsymbol{y}_i$ are all integer, we update our solution estimate and cost upper bound: $\boldsymbol{x}_0 := \boldsymbol{x}_i$, $\boldsymbol{y}_0 := \boldsymbol{y}_i$, and $f_0 := f_i$. Otherwise, we select one entry of $\boldsymbol{y}_i$ that is fractional (call it entry $j \in \{1, \ldots, m\}$) and we insert two sets in the collection $\mathcal{L}$:

$$\mathcal{S}_i^0 := \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}_i : y_j \leq \lfloor y_{i,j} \rfloor\}, \qquad \mathcal{S}_i^1 := \{(\boldsymbol{x}, \boldsymbol{y}) \in \mathcal{S}_i : y_j \geq \lceil y_{i,j} \rceil\},$$

where $\lfloor a \rfloor := \max\{b \in \mathbb{Z} : b \leq a\}$ and $\lceil a \rceil := \min\{b \in \mathbb{Z} : b \geq a\}$. Note that the union of $\mathcal{S}_i^0$ and $\mathcal{S}_i^1$ is smaller than $\mathcal{S}_i$, and that this subdivision eliminates the point $(\boldsymbol{x}_i, \boldsymbol{y}_i)$ from the BB search space.

When the BB algorithm terminates, if the returned value f_0 is infinity then the MIP is infeasible. Otherwise, we have $f_0 = f_{\text{opt}}$, $\boldsymbol{x}_0 = \boldsymbol{x}_{\text{opt}}$, and $\boldsymbol{y}_0 = \boldsymbol{y}_{\text{opt}}$.

The finite termination of the BB method is easily shown when the vector $\boldsymbol{y}$ can only take a finite number of values (because of the constraint set $\mathcal{S}$). In these cases, the BB process leads to an exhaustive enumeration in the worst case. When $\boldsymbol{y}$ can take infinite values, the convergence analysis is more involved [140, Section II.4.2].

The one described above is a very basic BB. For example, a simple modification allows us to keep track of a lower bound on the optimal value f_{opt}. This lower bound can be compared to the upper bound f_0 to terminate the algorithm early, when a prescribed tolerance is met. The BB method involves also many heuristics. For example, a variety of strategies can be employed for selecting a set $\mathcal{S}_i$ in step 1, or for deciding the splitting index j in step 4. BB can also be augmented with cutting-plane methods to form a class of algorithms called branch-and-cut. We point the reader to [140, Section II.4.2] and [42, Section 9.2] for all these details.

The efficiency of BB critically hinges on the quality of the relaxation. A relaxation that is easy to solve (e.g., a conic program) leads to fast BB iterations. While a strong MIP formulation gives tight lower bounds f_i and allows us to quickly discard large portions of the search space. In addition, a strong formulation is also likely to yield integer solutions in step 4. Another important factor for the efficiency of BB is the ability of reusing the solution of a subproblem to speed up the next subproblems. MILPs are especially efficient in this sense: with the simplex method, the (dual)

optimal solution of the subproblem (3.5) can be efficiently used as an initial guess for the two subproblems with constraint sets $\mathcal{S}_i^0$ and $\mathcal{S}_i^1$.

Chapter 4

Graphs

In this chapter we introduce some standard graph definitions and notation. We then illustrate a few classical problems that can be solved over a graph that will serve as running examples for this thesis. A great introductory reference for these topics is [146]. A more advanced and very detailed treatment of the subject can be found in [167] and [108].

4.1 Graphs

A **graph** G is an ordered pair $(\mathcal{V}, \mathcal{E})$ of finite sets. The elements $v \in \mathcal{V}$ of the first set are called vertices. The elements $e \in \mathcal{E}$ of the second set are called edges and are pairs of distinct vertices. If the edges are ordered pairs, denoted as $e = (v, w)$ for some $v, w \in \mathcal{V}$, then the graph is called **directed**. If the edges are unordered pairs, denoted as $e = \{v, w\}$, then the graph is **undirected**. In this thesis, when stating facts that hold both for directed and undirected graphs, we will use the neutral notation $e = [v, w]$ to represent edges that are either ordered or unordered pairs.

We call **weighted graph** a graph whose vertices and edges are labelled with scalars that represent costs (or weights). These costs are denoted as $c_v \in \mathbb{R}$ for all $v \in \mathcal{V}$ and $c_e \in \mathbb{R}$ for all $e \in \mathcal{E}$.

For an undirected graph $G = (\mathcal{V}, \mathcal{E})$, we let $\mathcal{I}_v$ be the set of edges that are incident with vertex $v \in \mathcal{V}$:

$$\mathcal{I}_v := \{e \in \mathcal{E} : v \in e\}.$$

We use a similar notation for the edges that connect a given subset $\mathcal{W} \subseteq \mathcal{V}$ of the vertices to its complement $\mathcal{V} \backslash \mathcal{W}$:

$$\mathcal{I}_{\mathcal{W}} := \{\{v, w\} \in \mathcal{E} : v \in \mathcal{W}, w \notin \mathcal{W}\}.$$

For a directed graph, we distinguish between edges that are outgoing, incoming, and incident with vertex v, or with the vertex subset $\mathcal{W}$:

$$\begin{aligned} \mathcal{I}_v^{\text{out}} &:= \{(v,w) \in \mathcal{E}\}, & \mathcal{I}_{\mathcal{W}}^{\text{out}} &:= \{(v,w) \in \mathcal{E} : v \in \mathcal{W}, w \notin \mathcal{W}\}, \\ \mathcal{I}_v^{\text{in}} &:= \{(w,v) \in \mathcal{E}\}, & \mathcal{I}_{\mathcal{W}}^{\text{in}} &:= \{(w,v) \in \mathcal{E} : v \in \mathcal{W}, w \notin \mathcal{W}\}, \\ \mathcal{I}_v &:= \mathcal{I}_v^{\text{in}} \cup \mathcal{I}_v^{\text{out}}, & \mathcal{I}_{\mathcal{W}} &:= \mathcal{I}_{\mathcal{W}}^{\text{in}} \cup \mathcal{I}_{\mathcal{W}}^{\text{out}}. \end{aligned}$$

4.2 Subgraphs

We say that a graph $H = (\mathcal{W}, \mathcal{F})$ is a **subgraph** of $G = (\mathcal{V}, \mathcal{E})$ if

$$\mathcal{W} \subseteq \mathcal{V}, \qquad \mathcal{F} \subseteq \mathcal{E}.$$

(Note that the condition $\mathcal{F} \subseteq \mathcal{W}^2$ is implicit in this definition.) We will use the notation $H \subseteq G$ to indicate that H is a subgraph of G.

A subgraph H can be represented as an element of the set $\{0,1\}^{\mathcal{V}\cup\mathcal{E}}$ of vectors with binary entries indexed by the elements of $\mathcal{V} \cup \mathcal{E}$. This vector is called the **incidence vector** of H, is denoted as $\boldsymbol{y}^H$, and has entries

$$y_v^H := \begin{cases} 1 & \text{if } v \in \mathcal{W} \\ 0 & \text{if } v \notin \mathcal{W} \end{cases}, \qquad y_e^H := \begin{cases} 1 & \text{if } e \in \mathcal{F} \\ 0 & \text{if } e \notin \mathcal{F} \end{cases},$$

for all $v \in \mathcal{V}$ and $e \in \mathcal{E}$.

Remark 4.1. Sometimes it can be helpful to think of the set $\{0,1\}^{\mathcal{V}\cup\mathcal{E}}$ as $\{0,1\}^{|\mathcal{V}|+|\mathcal{E}|}$. The only difference is that the vectors in the first set are indexed by vertices and edges, while the vectors in the second set are indexed by the integers $1, 2, \ldots, |\mathcal{V}| + |\mathcal{E}|$.

Given a subset $\mathcal{W} \subset \mathcal{V}$ of the vertices of a graph $G = (\mathcal{V}, \mathcal{E})$, we define the subgraph **induced** by $\mathcal{W}$ as

$$G_{\mathcal{W}} := (\mathcal{W}, \mathcal{E}_{\mathcal{W}}), \qquad \mathcal{E}_{\mathcal{W}} := \{[v,w] \in \mathcal{E} : v, w \in \mathcal{W}\}.$$

In words, this graph has vertices $\mathcal{W}$ and includes all the edges in G that have both ends in $\mathcal{W}$.

4.3 Special classes of graphs

This section introduces a few special classes of graphs that will be frequently mentioned in this thesis.

A **path** is a graph whose vertices can be ordered in such a way that each edge connects one vertex to the next. Formally, it is a graph $G = (\mathcal{V}, \mathcal{E})$, either directed or undirected, with

$$\mathcal{V} = \{v_0, v_1, \ldots, v_l\}, \qquad \mathcal{E} = \{[v_0, v_1], [v_1, v_2], \ldots, [v_{l-1}, v_l]\},$$

where $v_i \neq v_j$ for all $i, j \in \{0, \ldots, l\}$. The scalar $l \geq 0$ is equal to the number of edges in the path and is called the **path length** (for $l = 0$ we have $\mathcal{V} = \{v_0\}$ and $\mathcal{E} = \emptyset$). The vertex v_0 is called the source of the path and the vertex v_l is called the target. We will often denote these two vertices with the letters s and t.

A path in a graph $G = (\mathcal{V}, \mathcal{E})$ is a subgraph of G that is also a path. We use the term s-t path in G when we want to specify the source s and the target t of the path. If G contains at least one s-t path, we say that the vertices s and t are **connected**. We call a graph **connected** if all pairs of vertices in it are connected. A path in G is said to be **Hamiltonian** if it visits all the vertices in the graph, i.e., if $l = |\mathcal{V}| - 1$.

A **cycle** is similar to a path, but its first and last vertices coincide. Formally, it is a graph $G = (\mathcal{V}, \mathcal{E})$ with

$$\mathcal{V} = \{v_1, v_2, \ldots, v_l\}, \qquad \mathcal{E} = \{[v_1, v_2], \ldots, [v_{l-1}, v_l], [v_l, v_1]\},$$

where $v_i \neq v_j$ for all $i, j \in \{1, \ldots, l\}$. The length l of a cycle is again equal to the number of edges in the cycle, and is assumed to be greater than or equal to two. A cycle in a graph G is a subgraph of G that is also a cycle. A graph is **acyclic** if it does not contain cycles. A **tour** (or **Hamiltonian cycle**) in G is a cycle that visits every vertex, i.e., has length $l = |\mathcal{V}|$.

An undirected graph $G = (\mathcal{V}, \mathcal{E})$ is called a **tree** if it is both connected and acyclic. It is called a **matching** if every pair of edges $e, f \in \mathcal{E}$ are disjoint, i.e., $e \cap f = \emptyset$. A tree (respectively, a matching) in a graph G is a subgraph of G that is also a tree (respectively, a matching). If this subgraph covers all the vertices $\mathcal{V}$ of G, we call it a **spanning tree** (respectively, a **perfect matching**).

An undirected graph $G = (\mathcal{V}, \mathcal{E})$ is **bipartite** if its vertices $\mathcal{V}$ can be partitioned into two disjoint sets $\mathcal{V}_1$ and $\mathcal{V}_2$ such that every edge has the form $e = \{v, w\} \in \mathcal{E}$ for some $v \in \mathcal{V}_1$ and $w \in \mathcal{V}_2$. In this graph, we call an **assignment** of $\mathcal{V}_1$ a subgraph $H = (\mathcal{W}, \mathcal{F}) \subseteq G$ such that each vertex in $\mathcal{V}_1$ is covered by exactly one edge in $\mathcal{F}$.

4.4 Graph optimization problems

Many problems in combinatorial optimization and graph theory can be stated as follows: given a weighted graph G and a set $\mathcal{H}$ of admissible subgraphs of G, find a subgraph $H \in \mathcal{H}$ of minimum cost. In formulas,

$$\text{minimize} \quad \sum_{v \in \mathcal{W}} c_v + \sum_{e \in \mathcal{F}} c_e \tag{4.1a}$$

$$\text{subject to} \quad H = (\mathcal{W}, \mathcal{F}) \in \mathcal{H}, \tag{4.1b}$$

where the variable is the subgraph H. The objective function is the cost of H, defined as the sum of the costs of its vertices and edges. In this thesis we use the generic term **graph optimization problem** for a problem of the form (4.1).

A simple (and often effective) way of solving a graph optimization problem is to formulate it as an Integer Linear Program (ILP), and then use a Branch and Bound (BB) method like the one described in Section 3.3.2. To formulate (4.1) as an ILP, we first parameterize the subgraph H through its incidence vector $\boldsymbol{y}^H \in \{0,1\}^{\mathcal{V} \cup \mathcal{E}}$. Secondly, we describe the set $\mathcal{H}$ of admissible subgraphs through a polytope $\mathcal{Y}$ such that

$$\mathcal{Y} \cap \{0,1\}^{\mathcal{V} \cup \mathcal{E}} = \{\boldsymbol{y}^H : H \in \mathcal{H}\}.$$

In words, the vectors with integer coordinates in the polytope $\mathcal{Y}$ are the incidence vectors of the admissible subgraphs. (Without loss of generality, in the future sections and chapters we will assume that $\mathcal{Y} \subseteq [0,1]^{\mathcal{V} \cup \mathcal{E}}$.) Problem (4.1) can be then reformulated as the ILP

$$\text{minimize} \quad \sum_{v \in \mathcal{V}} c_v y_v + \sum_{e \in \mathcal{E}} c_e y_e \tag{4.2a}$$

$$\text{subject to} \quad \boldsymbol{y} \in \mathcal{Y} \cap \{0,1\}^{\mathcal{V} \cup \mathcal{E}}, \tag{4.2b}$$

where the only variable is $\boldsymbol{y}$, with entries y_v for all $v \in \mathcal{V}$ and y_e for all $e \in \mathcal{E}$. This ILP has optimal value equal to the optimal value of problem (4.1), and an optimal solution $\boldsymbol{y}$ is the incidence vector of an optimal subgraph H of problem (4.1).

Note that, for each instance of problem (4.1), there are multiple (infinitely many) polytopes $\mathcal{Y}$ that we can use to describe the set $\mathcal{H}$. Ideally, as discussed in Section 3.2, we would like to find a good compromise between a polytope with few facets and a polytope such that the inclusion

$$\text{conv}(\mathcal{T}) \subseteq \mathcal{Y} \tag{4.3}$$

is as tight as possible, where $\mathcal{T} := \mathcal{Y} \cap \{0,1\}^{\mathcal{V}\cup\mathcal{E}}$ is the feasible set of the ILP (4.2). In fact, if the inclusion (4.3) holds with equality we have a perfect formulation, and the ILP can be efficiently solved as a Linear Program (LP).

We conclude this chapter with a list of famous graph problems and the corresponding ILP formulations.

4.4.1 Shortest path

Let $G = (\mathcal{V}, \mathcal{E})$ be a weighted graph. Given a source vertex $s \in \mathcal{V}$ and target vertex $t \in \mathcal{V}$, in the **Shortest-Path Problem** (SPP) we seek an s-t path of minimum cost in G. We observe that the SPP is a special case of the graph optimization problem (4.1), where the set $\mathcal{H}$ of admissible subgraphs contains all the s-t paths in G.

For a directed graph G, the SPP can be formulated as an ILP of the form (4.2) by defining the polytope $\mathcal{Y} \subseteq [0,1]^{\mathcal{V}\cup\mathcal{E}}$ through the following linear constraints:

$$y_v = \sum_{e\in\mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e\in\mathcal{I}_v^{\text{out}}} y_e + \delta_{tv} \leq 1, \qquad \forall v \in \mathcal{V}, \tag{4.4a}$$

$$\sum_{e\in\mathcal{E}_\mathcal{U}} y_e \leq |\mathcal{U}| - 1, \qquad \forall \mathcal{U} \subset \mathcal{V} : |\mathcal{U}| \geq 2, \tag{4.4b}$$

$$y_e \geq 0, \qquad \forall e \in \mathcal{E}. \tag{4.4c}$$

For two vertices $v, w \in \mathcal{V}$, the scalar δ_{vw} is equal to one if $v = w$ and is zero otherwise. To interpret the first constraint, it helps to think of the SPP as the problem of shipping one unit of water from the source s to the target t. The binary variable y_e represents the flow of water carried by the edge e, while y_v is the total flow that goes through vertex v. Constraint (4.4a) enforces then the **flow conservation**: one unit of flow is injected in the source and ejected from the target, while the flow at every other vertex is at most one and conserved (flow in equals flow out). Satisfying the flow conservation does not ensure that the subgraph H parameterized by $\boldsymbol{y}$ is a path, since H might include isolated cycles. The role of the second constraint is to eliminate such cycles: if $\mathcal{U}$ is the set of vertices traversed by a cycle, then the left-hand side of (4.4b) is equal to $|\mathcal{U}|$ and this constraint is violated. Note that these constraints are exponential in number: typically, they are not included in the optimization problem from the beginning, but are added iteratively only if the obtained solution violates them.

The cycle-elimination constraints (4.4b) have been originally proposed by Dantzig, Fulkerson, and Johnson [45]. Despite the exponentially many constraints, the ILP formulation (4.4) is not perfect, i.e., its linear relaxation needs not be exact [178, Proposition 4]. Many alternative cycle-elimination constraints can be found in the

literature, each leading to a different ILP formulation of the SPP [178, Section 2]. A stronger, but still not perfect and even larger, formulation of the SPP is obtained by replacing the constraints in (4.4b) with the following constraints [178, Propositions 2]:

$$\sum_{e \in \mathcal{E}_{\mathcal{U}}} y_e \leq \sum_{u \in \mathcal{U} \setminus \{v\}} y_u, \qquad \forall v \in \mathcal{U},\ \mathcal{U} \subset \mathcal{V} : |\mathcal{U}| \geq 2. \tag{4.5}$$

This alternative exponential-size formulation can be verified to be as strong as the polynomial-size formulation proposed in [95] (see [178, Theorem 5]).

The picture changes dramatically if we assume that the costs c_v for $v \in \mathcal{V}$ and c_e for $e \in \mathcal{E}$ are nonnegative. Under this assumption, the objective function (4.2a) can only "push" the variables $\boldsymbol{y}$ in the direction of the nonpositive orthant. Therefore, instead of condition (4.3), the tightness of the linear relaxation is quantified by the inclusion

$$\operatorname{conv}(\mathcal{T}) + \mathbb{R}^{\mathcal{V} \cup \mathcal{E}}_{\geq 0} \subseteq \mathcal{Y} + \mathbb{R}^{\mathcal{V} \cup \mathcal{E}}_{\geq 0}. \tag{4.6}$$

In words, we neglect any difference that the polytopes $\operatorname{conv}(\mathcal{T})$ and $\mathcal{Y}$ might have in the direction of the nonnegative orthant. If the costs are nonnegative, the cycle-elimination constraint (4.4b) can be safely removed. The resulting polytope $\mathcal{Y}$ has a number of facets (number of constraints) that is only linear in the size of the graph G. Furthermore, for this polytope the inclusion (4.6) holds with equality (see, e.g., [167, Section 13.1]). Therefore, the linear relaxation of the ILP formulation of the directed SPP with nonnegative costs is always exact.

For an undirected graph G, the SPP could be formulated as an ILP, e.g., by replacing the flow-conservation constraint (4.4a) with

$$y_v = \frac{1}{2} \left(\sum_{e \in \mathcal{I}_v} y_e + \delta_{sv} + \delta_{tv} \right) \leq 1, \qquad \forall v \in \mathcal{V}.$$

However, this substitution significantly deteriorates the tightness of the linear relaxation. In practice, we then prefer to reduce an undirected SPP to a directed SPP by replacing each unordered edge $\{v, w\} \in \mathcal{E}$ with two ordered edges (v, w) and (w, v), and rely on the ILP formulation (4.4) of the directed problem.

If negative weights are allowed, then the SPP (directed or undirected) is well known to be NP-hard. If all the weights are nonnegative, then the SPP can be solved in polynomial time, e.g., through an LP.

4.4.2 Travelling salesperson

In the **Travelling-Salesperson Problem** (TSP) we seek a tour of minimum cost in a given weighted graph G. This problem is also a special case of the graph optimization problem (4.1), where $\mathcal{H}$ is the set of all the tours in G. A variety of interesting applications of the TSP can be found in [133].

For a directed graph G, the TSP is formulated as an ILP of the form (4.2) by letting the polytope $\mathcal{Y}$ be such that

$$y_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} y_e = \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e = 1, \qquad \forall v \in \mathcal{V}, \tag{4.7a}$$

$$\sum_{e \in \mathcal{E}_{\mathcal{U}}} y_e \leq |\mathcal{U}| - 1, \qquad \forall \mathcal{U} \subset \mathcal{V} : |\mathcal{U}| \geq 2, \tag{4.7b}$$

$$y_e \geq 0, \qquad e \in \mathcal{E}. \tag{4.7c}$$

The first constraint ensures that each vertex is visited by the tour exactly once. The second constraint is identical to (4.4b) and prevents cycles in the optimal solution.

If the graph G is undirected, we simply replace constraint (4.7a) with

$$y_v = \frac{1}{2} \sum_{e \in \mathcal{I}_v} y_e = 1, \qquad \forall v \in \mathcal{V}. \tag{4.8}$$

Note that in case of an undirected graph the cycle-elimination constraint (4.7b) is redundant for $|\mathcal{U}| = 2$.

These ILP formulations of the directed and undirected TSP have exponential size and are not perfect (a simple example of this can be found in [108, Section 21.4]). Similarly to the SPP, there are many well-known formulation of the TSP that have polynomial size, but the one given above is typically more effective in practice (provided that the cycle-elimination constraints are added iteratively).

The TSP is one of the most famous NP-complete problems [104], both in the directed and undirected case and regardless the sign of the cost weights.

4.4.3 Minimum spanning tree

Given an undirected weighted graph G, the **Minimum-Spanning-Tree Problem** (MSTP) asks for a spanning tree of minimum cost. This problem is a special case of (4.1), where the set $\mathcal{H}$ contains all the spanning trees.

By defining the polytope $\mathcal{Y}$ through the following linear constraints, we obtain an

ILP formulation of the MSTP:

$$y_v = 1, \qquad \forall v \in \mathcal{V}, \tag{4.9a}$$

$$\sum_{e \in \mathcal{E}} y_e = |\mathcal{V}| - 1, \tag{4.9b}$$

$$\sum_{e \in \mathcal{E}_\mathcal{U}} y_e \leq |\mathcal{U}| - 1, \qquad \forall \mathcal{U} \subset \mathcal{V} : |\mathcal{U}| \geq 3, \tag{4.9c}$$

$$y_e \geq 0, \qquad \forall e \in \mathcal{E}. \tag{4.9d}$$

The first constraint requires the tree to span every vertex. The second says that the number of edges in the tree must be one less than the number of vertices. The third ensures that the tree does not have cycles.

The constraints in (4.9) are exponential in number, but in [58] it has been shown that they yield a perfect formulation. An alternative exponential-size formulation of the MSTP has the following constraints in place of (4.9c):

$$\sum_{e \in \mathcal{I}_\mathcal{U}} y_e \geq 1, \qquad \forall \mathcal{U} \subset \mathcal{V} : |\mathcal{U}| \geq 1. \tag{4.10}$$

These ensure that the tree is connected. It can be verified that this formulation is weaker than the one in (4.9) [16, Theorem 10.1]. However, for some specific problems, this family of constraints can be more effective.

An extended formulation of this polytope has been presented in [132, Section 3.1], i.e., a description of $\mathcal{Y}$ as the projection of a higher-dimensional polytope that is defined by a number of variables and linear constraints that is only polynomial in the size of the graph. This also implies that the MSTP can be solved in polynomial time.

4.4.4 Facility location

In the **Facility-Location Problem** (FLP) the weighted graph G is undirected and bipartite, with vertices $\mathcal{V} = \mathcal{V}_1 \cup \mathcal{V}_2$. The vertices $\mathcal{V}_1$ are called clients and the vertices $\mathcal{V}_2$ are called facilities. The goal is to find a minimum-cost assignment of each client to a facility. By including in the set $\mathcal{H}$ all the assignments of $\mathcal{V}_1$, we see that also the FLP is a subclass of problem (4.1).

The FLP is formulated as an ILP (4.2) by letting the polytope $\mathcal{Y}$ enforce the

constraints

$$y_v = \sum_{e \in \mathcal{I}_v} y_e = 1, \qquad \forall v \in \mathcal{V}_1, \tag{4.11a}$$

$$y_v \geq y_e \geq 0, \qquad \forall v \in \mathcal{V}_2,\ e \in \mathcal{I}_v, \tag{4.11b}$$

$$y_v \leq 1, \qquad \forall v \in \mathcal{V}_2. \tag{4.11c}$$

The first constraint ensures that each client is assigned to exactly one facility. The second constraint says that a facility needs to be open in order to cover a client. Note that the binary variables y_v for $v \in \mathcal{V}_2$ decide whether the facility v is open or not. The ILP formulation defined by (4.11) is compact, but its convex relaxation can be loose.

The FLP is well-known to be NP-hard (see, e.g., [108, Proposition 22.1]). The **Minimum-Set-Cover Problem** (MSCP) is another widely studied problem that is easily seen to be equivalent to the FLP.

4.4.5 Minimum perfect matching

In the **Minimum-Perfect-Matching Problem** (MPMP) we seek a perfect matching in an undirected weighted graph G. Letting $\mathcal{H}$ be the set of all perfect matchings in G, shows that the MPMP is a special case of problem (4.1).

The MPMP is formulated as an ILP of the form (4.2) by defining the polytope $\mathcal{Y}$ through the constraints

$$y_v = \sum_{e \in \mathcal{I}_v} y_e = 1, \qquad \forall v \in \mathcal{V}, \tag{4.12a}$$

$$y_e \geq 0, \qquad e \in \mathcal{E}. \tag{4.12b}$$

The first constraint ensures that every vertex is matched.

The ILP formulation defined by (4.12) has a loose convex relaxation. In fact, for a complete graph with three vertices the MPMP is infeasible, but the solution $y_e = 1/2$ for all $e \in \mathcal{E}$ and $y_v = 1$ for all $v \in \mathcal{V}$ is feasible for the constraints above. By adding the following set of (exponentially many) constraints, the convex relaxation of the ILP becomes perfect [57, Section 2]:

$$\sum_{e \in \mathcal{I}_\mathcal{U}} y_e \geq 1, \qquad \forall \mathcal{U} \subset \mathcal{V} : |\mathcal{U}| \geq 3,\ |\mathcal{U}| \text{ is odd}. \tag{4.13}$$

The MPMP is solvable in polynomial time using Edmonds' matching algorithm [57].

On the other hand, contrary to the MSTP, it does not admit an extended formulation [163].

Part II

Framework and methodology

Chapter 5

Graphs of convex sets

In this chapter we introduce the class of problems and the framework at the core of this thesis. We start by formally defining what a Graph of Convex Sets (GCS) is. Then we show how the graph optimization problems described in Section 4.4 are naturally extended to GCSs. Finally, we introduce a general methodology to formulate a GCS problem as an efficient Mixed-Integer Convex Program (MICP).

The results in this chapter generalize the techniques presented in [131] for finding shortest paths in GCSs.

5.1 What is a graph of convex sets?

A GCS is a graph $G = (\mathcal{V}, \mathcal{E})$ where each vertex $v \in \mathcal{V}$ is paired with a convex program, and each edge $e = [v, w] \in \mathcal{E}$ corresponds to convex costs and constraints that couple the programs of vertices v and w. As the notation $[v, w]$ suggests, the graph G can be directed or undirected. The convex program of vertex v has variables $\boldsymbol{x}_v \in \mathbb{R}^{n_v}$, constraint set $\mathcal{X}_v \subset \mathbb{R}^{n_v}$, and objective function $f_v : \mathbb{R}^{n_v} \to \mathbb{R}$. While the constraint set and cost function paired with edge $e = [u, v]$ are $\mathcal{X}_e \subseteq \mathbb{R}^{n_v+n_w}$ and $f_e : \mathbb{R}^{n_v+n_w} \to \mathbb{R}$, respectively. We assume that the sets $\mathcal{X}_v$ and $\mathcal{X}_e$ are nonempty, closed, and convex. The sets $\mathcal{X}_v$ are also assumed to be bounded. The functions f_v and f_e are convex.

Remark 5.1. We can extend the definition of a GCS to include additional variables $\boldsymbol{x}_e \in \mathbb{R}^{n_e}$ that are paired with every edge $e \in \mathcal{E}$. In this case, the constraint set and the cost function of the edge e would be $\mathcal{X}'_e \subseteq \mathbb{R}^{n_v+n_w+n_e}$ and $f'_e : \mathbb{R}^{n_v+n_w+n_e} \to \mathbb{R}$, respectively. These extra variables can make the modelling easier but, from a mathematical point of view, this case is easily reduced to the one above. In fact, we

can define the set $\mathcal{X}_e$ by projecting $\mathcal{X}'_e$ onto the space of the variables $\boldsymbol{x}_v$ and $\boldsymbol{x}_w$, i.e.,

$$\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w) : (\boldsymbol{x}_v, \boldsymbol{x}_w, \boldsymbol{x}_e) \in \mathcal{X}'_e \text{ for some } \boldsymbol{x}_e \in \mathbb{R}^{n_e}\}.$$

Then we can define the edge cost function by partial minimization over the extra variable $\boldsymbol{x}_e$:

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \inf\{f'_e(\boldsymbol{x}_v, \boldsymbol{x}_w, \boldsymbol{x}_e) : (\boldsymbol{x}_v, \boldsymbol{x}_w, \boldsymbol{x}_e) \in \mathcal{X}'_e\}.$$

It is not hard to see that the set $\mathcal{X}_e$ and the function f_e verify all the convexity, closure, and boundedness assumptions required by our framework (see, e.g., [25, Section 3.2.5] for the convexity part). Therefore we can replace $\mathcal{X}'_e$ and f'_e with $\mathcal{X}_e$ and f_e, and eliminate the extra variables $\boldsymbol{x}_e$ from the GCS.

5.2 GCS problems

We consider a generalization of the graph optimization problem (4.1) where the graph G is substituted with a GCS. This substitution leads naturally to the following optimization problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{v \in \mathcal{W}} f_v(\boldsymbol{x}_v) + \sum_{e=[v,w] \in \mathcal{F}} f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) && && \text{(5.1a)}\\
\text{subject to} \quad & H = (\mathcal{W}, \mathcal{F}) \in \mathcal{H}, && && \text{(5.1b)}\\
& \boldsymbol{x}_v \in \mathcal{X}_v, && \forall v \in \mathcal{W}, && \text{(5.1c)}\\
& (\boldsymbol{x}_v, \boldsymbol{x}_w) \in \mathcal{X}_e, && \forall e = [v, w] \in \mathcal{F}. && \text{(5.1d)}
\end{aligned}$$

Here the variables are the discrete subgraph $H \subseteq G$ and the continuous vectors $\boldsymbol{x}_v$ for all $v \in \mathcal{V}$. The objective function is equal to the cost of the subgraph H, defined as the sum of the cost functions paired with the vertices and the edges in H. The first constraint is identical to (4.1b), and states that the subgraph H can only be selected within a given class $\mathcal{H}$ of admissible subgraphs of G. The second and third constraints enforce the convex constraints paired with the vertices and edges selected by H.

The GCS problem (5.1) has a combinatorial component (choosing the subgraph H) and a convex component (optimizing the variables $\boldsymbol{x}_v$). We observe that if the subgraph H is fixed, then the GCS problem reduces to a convex optimization problem. Conversely, when we fix the value of the continuous variables $\boldsymbol{x}_v$, we get back a graph optimization problem of the form (4.1); over a graph with vertices $\{v \in \mathcal{V} : \boldsymbol{x}_v \in \mathcal{X}_v\}$,

edges $\{e = [v, w] \in \mathcal{E} : (\boldsymbol{x}_v, \boldsymbol{x}_w) \in \mathcal{X}_e\}$, vertex weights $c_v := f_v(\boldsymbol{x}_v)$, and edge weights $c_e := f_e(\boldsymbol{x}_v, \boldsymbol{x}_w)$. This observation can be used to devise a variety of heuristic methods to solve the GCS problem (5.1). For example, given an initial guess for the variables $\boldsymbol{x}_v$, we could try to find a locally optimal solution by alternating a graph optimization problem and a convex program. The effectiveness of these heuristics, however, is very instance dependent. The goal of the next section is to derive a general methodology that allows us to formulate any GCS problem as a strong and compact MICP. This MICP can then be reliably solved to global optimality using, for example, the branch-band-bound algorithm from Section 3.3.2.

5.3 Mixed-integer formulation

We take two steps to formulate the GCS problem (5.1) as an MICP. In the first step, we formulate the GCS problem as a simple MINCP. Because of its nonconvex costs and constraints, this problem will not be efficiently solvable yet. In the second step, we use a convexification process tailored to the structure of our problem, which yields an MICP that is equivalent to, but much easier to solve than, the MINCP.

5.3.1 Nonconvex formulation

We formulate the MINCP by building on the ILP formulation (4.2) of the purely discrete problem that underlies our GCS problem. We parameterize the subgraph H through its incidence vector $\boldsymbol{y}^H \in \{0,1\}^{\mathcal{V}\cup\mathcal{E}}$. While the set $\mathcal{H}$ is represented by a polytope $\mathcal{Y} \subseteq [0,1]^{\mathcal{V}\cup\mathcal{E}}$ that constrains the incidence vectors of the admissible subgraphs. Given this parameterization, the GCS problem (5.1) is formulated as the following optimization problem:

$$\begin{aligned}
\text{minimize} \quad & \sum_{v\in\mathcal{V}} y_v f_v(\boldsymbol{x}_v) + \sum_{e=[v,w]\in\mathcal{E}} y_e f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) && && \text{(5.2a)}\\
\text{subject to} \quad & \boldsymbol{y} \in \mathcal{Y} \cap \{0,1\}^{\mathcal{V}\cup\mathcal{E}}, && && \text{(5.2b)}\\
& y_v \boldsymbol{x}_v \in y_v \mathcal{X}_v, && \forall v \in \mathcal{V}, && \text{(5.2c)}\\
& y_e(\boldsymbol{x}_v, \boldsymbol{x}_w) \in y_e \mathcal{X}_e, && \forall e = [v,w] \in \mathcal{E}. && \text{(5.2d)}
\end{aligned}$$

The first constraint ensures that the vector $\boldsymbol{y}$ is the incidence vector of a subgraph $H \in \mathcal{H}$. The other constraints use the binary variables in $\boldsymbol{y}$ to select the appropriate constraints from the vertices and edges of the GCS. For example, the second constraint simplifies to $\boldsymbol{x}_v \in \mathcal{X}_v$ if $y_v = 1$, and gives the redundant constraint $\mathbf{0} \in 0\mathcal{X}_v = \{\mathbf{0}\}$

when $y_v = 0$. The objective function operates similarly, and selects only the cost functions of the vertices and edges that belong to our subgraph.

Problem (5.2) is an MINCP, since the objective function and the last two constraints are not convex. Our next goal is to manipulate this problem and isolate its nonconvexity into a single collection of bilinear equality constraints.

For each vertex $v \in \mathcal{V}$, we introduce the auxiliary variable $\boldsymbol{z}_v := y_v \boldsymbol{x}_v$. Using these new variables, the nonnegativity of y_v, and the formula (2.2), the nonconvex constraint (5.2c) is rewritten as a convex constraint in $(\boldsymbol{z}_v, y_v)$:

$$y_v \geq 0,\ y_v \boldsymbol{x}_v \in y_v \mathcal{X}_v \quad \Longleftrightarrow \quad (\boldsymbol{z}_v, y_v) \in \tilde{\mathcal{X}}_v, \tag{5.3}$$

where $\tilde{\mathcal{X}}_v$ denotes the (closure of the) homogenization of $\mathcal{X}_v$.[1] Similarly, the nonconvex vertex costs in (5.2a) can be rewritten as the convex costs $\tilde{f}_v(\boldsymbol{z}_v, y_v)$. In fact, for $y_v > 0$, we have

$$y_v f_v(\boldsymbol{x}_v) = y_v f_v(y_v \boldsymbol{x}_v / y_v) = y_v f_v(\boldsymbol{z}_v / y_v) = \tilde{f}_v(\boldsymbol{z}_v, y_v).$$

While, for $y_v = 0$, constraint (5.3) gives us $\boldsymbol{z}_v = \mathbf{0}$ and we have

$$y_v f_v(\boldsymbol{x}_v) = 0 f_v(\boldsymbol{x}_v) = 0 = \tilde{f}_v(\mathbf{0}, 0),$$

where the last equality follows from (2.11).

The operation just done for the vertices of the GCS can be repeated for the edges. For each edge $e \in \mathcal{I}_v$ incident with vertex v, we define an auxiliary variable $\boldsymbol{z}_v^e := y_e \boldsymbol{x}_v$. Then we substitute constraint (5.2d) with the convex constraint $(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) \in \tilde{\mathcal{X}}_e$, and the edge costs in (5.2a) with the convex costs $\tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e)$.[2]

Overall, we have transformed the MINCP (5.2) into the equivalent optimization

[1] Everywhere in this and the following chapters we will omit writing the closure in front of the homogenization explicitly. When using the symbol tilde, we will always mean the closure of the homogenization, both for sets and for functions.

[2] We are slightly abusing notation here. To be precise, we should write $((\boldsymbol{z}_v^e, \boldsymbol{z}_w^e), y_e) \in \tilde{\mathcal{X}}_e$ and $\tilde{f}_e((\boldsymbol{z}_v^e, \boldsymbol{z}_w^e), y_e)$.

problem

$$\begin{aligned}
\text{minimize} \quad & \sum_{v\in\mathcal{V}} \tilde{f}_v(\boldsymbol{z}_v, y_v) + \sum_{e\in\mathcal{E}} \tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) && && \text{(5.4a)}\\
\text{subject to} \quad & \boldsymbol{y} \in \mathcal{Y} \cap \{0,1\}^{\mathcal{V}\cup\mathcal{E}}, && && \text{(5.4b)}\\
& (\boldsymbol{z}_v, y_v) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V}, && \text{(5.4c)}\\
& (\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) \in \tilde{\mathcal{X}}_e, && \forall e = [v,w] \in \mathcal{E}, && \text{(5.4d)}\\
& \boldsymbol{z}_v = y_v \boldsymbol{x}_v,\ \boldsymbol{z}_v^e = y_e \boldsymbol{x}_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v. && \text{(5.4e)}
\end{aligned}$$

This is also an MINCP, but has the advantage that its only nonconvexity (except for the integrality constraints) are the bilinear constraints (5.4e). Our next step is to design a convex relaxation tailored to these bilinear constraints.

Remark 5.2. Recall that, for the convex sets $\mathcal{X}_v$ and $\mathcal{X}_e$ and the convex costs f_v and f_e that are typically encountered in convex optimization, the (closure of the) homogenization can be computed very easily using the formula (2.5).

5.3.2 Convex formulation

There are many ways to reformulate the MINCP (5.4) as an MICP. Here we show a simple and effective approach that carefully balances the MICP size and strength. First we add as many valid convex constraints as we can to the MINCP (we say that a constraint is valid for an optimization problem if it does not affect the problem optimal value). Then we obtain our MICP simply by dropping the bilinear constraints (5.4e). The goal of the additional convex constraints is twofold:

- First, they need to exactly replicate the effects of the bilinear constraints (5.4e) when the entries of $\boldsymbol{y}$ take binary value. This ensures that our MICP is a correct formulation of the original GCS problem (5.1).
- Secondly, they need to envelop the bilinear constraints (5.4e) as tightly as possible, so that the convex relaxation of our MICP gives tight lower bounds.

The following lemma gives us an algorithmic way of generating a family of valid convex constraints for the MINCP (5.4).

Lemma 5.1. *For some vertex $v \in \mathcal{V}$, assume that the linear inequality*

$$a y_v + \sum_{e\in\mathcal{I}_v} a_e y_e + b \geq 0 \tag{5.5}$$

is valid for the polytope $\mathcal{Y}$. Then the convex constraint

$$\left(a\boldsymbol{z}_v + \sum_{e\in\mathcal{I}_v} a_e \boldsymbol{z}_v^e + b\boldsymbol{x}_v,\ ay_v + \sum_{e\in\mathcal{I}_v} a_e y_e + b\right) \in \tilde{\mathcal{X}}_v \tag{5.6}$$

is valid for problem (5.4).

Proof. Constraint (5.6) requires two conditions to hold. One is (5.5), which is assumed. For the second, we multiply both sides of the constraint $\boldsymbol{x}_v \in \mathcal{X}_v$ (which is easily seen to be valid for the MINCP (5.4)) by the left-hand side of (5.5). Then we use the bilinear constraints (5.4e) to linearize all the variable products. □

Let us illustrate the usage of Lemma 5.1 through some simple examples. Recall that the polytope $\mathcal{Y}$ is assumed to be contained in the unit hypercube $[0,1]^{\mathcal{V}\cup\mathcal{E}}$. Therefore, for any given vertex v, the inequalities $0 \le y_v \le 1$ and $0 \le y_e \le 1$ for all $e \in \mathcal{I}_v$ are certainly valid for $\mathcal{Y}$. We can then apply Lemma 5.1 to these inequalities to derive the following valid convex constraints:

$$y_v \ge 0 \quad \Rightarrow \quad (\boldsymbol{z}_v, y_v) \in \tilde{\mathcal{X}}_v, \qquad \forall v \in \mathcal{V}, \tag{5.7a}$$

$$1 - y_v \ge 0 \quad \Rightarrow \quad (\boldsymbol{x}_v - \boldsymbol{z}_v, 1 - y_v) \in \tilde{\mathcal{X}}_v, \qquad \forall v \in \mathcal{V}, \tag{5.7b}$$

$$y_e \ge 0 \quad \Rightarrow \quad (\boldsymbol{z}_v^e, y_e) \in \tilde{\mathcal{X}}_v, \qquad \forall v \in \mathcal{V},\ e \in \mathcal{I}_v, \tag{5.7c}$$

$$1 - y_e \ge 0 \quad \Rightarrow \quad (\boldsymbol{x}_v - \boldsymbol{z}_v^e, 1 - y_e) \in \tilde{\mathcal{X}}_v, \qquad \forall v \in \mathcal{V},\ e \in \mathcal{I}_v. \tag{5.7d}$$

We note that the first condition above is equal to (5.4c), while the remaining three conditions are not part of (but implied by) the constraints of our MINCP.

Remark 5.3. Lemma 5.1 is easily specialized to linear equality constraints of the form

$$ay_v + \sum_{e\in\mathcal{I}_v} a_e y_e + b = 0.$$

In this case, the implied valid constraint is also a linear equality:

$$a\boldsymbol{z}_v + \sum_{e\in\mathcal{I}_v} a_e \boldsymbol{z}_v^e + b\boldsymbol{x}_v = \boldsymbol{0}.$$

In Chapter 6 we will analyze specific classes of GCS problems, and we will apply Lemma 5.1 to derive additional convex constraints tailored to those problems. However, as shown in the next theorem, the constraints in (5.7) are already sufficient to achieve our first goal above, i.e., give us a correct MICP formulation of the GCS problem.

Theorem 5.1. *The MICP*

$$\begin{aligned}
\text{minimize} \quad & \sum_{v\in\mathcal{V}} \tilde{f}_v(\boldsymbol{z}_v, y_v) + \sum_{e\in\mathcal{E}} \tilde{f}_e(\boldsymbol{z}^e_v, \boldsymbol{z}^e_w, y_e) && && (5.8a)\\
\text{subject to} \quad & \boldsymbol{y} \in \mathcal{Y} \cap \{0,1\}^{\mathcal{V}\cup\mathcal{E}}, && && (5.8b)\\
& (\boldsymbol{z}_v, y_v) \in \tilde{\mathcal{X}}_v,\ (\boldsymbol{x}_v - \boldsymbol{z}_v, 1 - y_v) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V}, && (5.8c)\\
& (\boldsymbol{z}^e_v, y_e) \in \tilde{\mathcal{X}}_v,\ (\boldsymbol{x}_v - \boldsymbol{z}^e_v, 1 - y_e) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v, && (5.8d)\\
& (\boldsymbol{z}^e_v, \boldsymbol{z}^e_w, y_e) \in \tilde{\mathcal{X}}_e, && \forall e = [v, w] \in \mathcal{E}. && (5.8e)
\end{aligned}$$

is a correct formulation of the GCS problem (5.1).

Proof. Since the MICP (5.8) is a relaxation of the MINCP (5.4), it suffices to show that the feasible points of the MICP satisfy the bilinear equalities $\boldsymbol{z}_v = y_v\boldsymbol{x}_v$ and $\boldsymbol{z}^e_v = y_e\boldsymbol{x}_v$ for all $v \in \mathcal{V}$ and $e \in \mathcal{I}_v$. If $y_v = 0$ ($y_v = 1$) the bilinear equalities give us $\boldsymbol{z}_v = \boldsymbol{0}$ ($\boldsymbol{z}_v = \boldsymbol{x}_v$), and this is also ensured by the first (second) constraint in (5.8c). If $y_e = 0$ ($y_e = 1$) the bilinear equalities give us $\boldsymbol{z}^e_v = \boldsymbol{0}$ ($\boldsymbol{z}^e_v = \boldsymbol{x}_v$), and this is also ensured by the first (second) constraint in (5.8d). □

In its current form, the MICP (5.8) can have a very loose convex relaxation. What will tighten the relaxation is the application of Lemma 5.1 to the problem specific constraints (as shown Chapter 6). The MICP (5.8) has small size: in its design we did not introduce additional variables with respect to the original MINCP (5.4). Furthermore, the usage of Lemma 5.1 gives us a number of additional constraints that is at most equal to the number of inequalities defining the polytope $\mathcal{Y}$.

5.4 Discussion

The core idea behind Lemma 5.1 (i.e., deriving a valid constraint by multiplying a valid convex constraint by a valid linear inequality) is very general. We will extend and analyze in depth this procedure in Chapter 8. What is peculiar about Lemma 5.1 is the assumption that the linear inequality (5.5) involves only the binary variables related to a single vertex v. The reason for this is that, if we were to multiply the vector $\boldsymbol{x}_v$ by a generic linear expression

$$\sum_{w\in\mathcal{V}} a_w y_w + \sum_{e\in\mathcal{E}} a_e y_e + b,$$

we would get products of variables that cannot be linearized using the variables in the MINCP (5.4). Therefore we would have to introduce additional variables (potentially

all possible products $y_w \boldsymbol{x}_v$ for $v, w \in \mathcal{V}$ and $y_e \boldsymbol{x}_v$ for $v \in \mathcal{V}$ and $e \in \mathcal{E}$), but this would make our optimization problems much larger and slower to solve.

In Section 4.4 we have seen that efficient ILP formulations of some graph optimization problems require the use of auxiliary variables (so-called extended formulations). In these formulations, the polytope $\mathcal{Y}$ is described as the projection onto the space of the variables $\boldsymbol{y}$ of a higher-dimensional polytope. Starting from such a formulation, we face a decision similar to the one just discussed: either we exclude the constraints involving the auxiliary variables from the convexification process, or we include them at the price of introducing extra variables that represent the products of the auxiliary variables and the vertex positions $\boldsymbol{x}_v$. The first route yields smaller but weaker formulations, the second gives us larger but stronger formulations.

We observe that if each set $\mathcal{X}_v$ contains only one point $\bar{\boldsymbol{x}}_v \in \mathbb{R}^{n_v}$, then our MICP reduces to the ILP formulation (4.2) of the graph optimization problem underlying the GCS problem. In fact, under this assumption, the constraints (5.8c) and (5.8d) imply $\boldsymbol{z}_v = y_v \bar{\boldsymbol{x}}_v$ and $\boldsymbol{z}_v^e = y_e \bar{\boldsymbol{x}}_v$ for all $v \in \mathcal{V}$ and $e \in \mathcal{I}_v$. Substituting these values in the objective function, we obtain

$$\tilde{f}_v(y_v \bar{\boldsymbol{x}}_v, y_v) = y_v \tilde{f}_v(\bar{\boldsymbol{x}}_v, 1) = y_v f_v(\bar{\boldsymbol{x}}_v) = y_v c_v$$

where $c_v := f_v(\bar{\boldsymbol{x}}_v) \in \mathbb{R}$ is the fixed cost of vertex $v \in \mathcal{V}$. Similarly, for each edge $e \in \mathcal{E}$, we have

$$\tilde{f}_e(y_e \bar{\boldsymbol{x}}_v, y_e \bar{\boldsymbol{x}}_w, y_e) = y_e \tilde{f}_e(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{x}}_w, 1) = y_e f_e(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{x}}_w) = y_e c_e$$

where $c_e := f_e(\bar{\boldsymbol{x}}_v, \bar{\boldsymbol{x}}_w) \in \mathbb{R}$ is the fixed cost of edge e. Informally, this shows that the strength of the initial ILP formulation has a fundamental effect on the strength of our MICP. It also suggests that sets $\mathcal{X}_v$ of large volume can lead to loose convex relaxations.

For the convex sets $\mathcal{X}_v$ and $\mathcal{X}_e$ that typically appear in practice, the MICP (5.8) can be solved to global optimality with standard solvers. However, problem (5.8) can be tackled numerically even when the sets in our GCS are not defined by explicit constraints (e.g., convex inequalities). For example, each sets $\mathcal{X}_v$ and $\mathcal{X}_e$ may be very complex and accessible only through an oracle that, given a point, either certifies that the point is in the set or returns a separating hyperplane. In fact, such an oracle is easily adapted to checking membership to the homogenizations of $\mathcal{X}_v$ and $\mathcal{X}_e$, and this black-box access to the problem constraints is sufficient for efficient optimization algorithms like the ellipsoid method [86].

Chapter 6

Examples of GCS problems

In the previous chapter we have introduced a general methodology to formulate a Graph of Convex Sets (GCS) problem as a Mixed-Integer Convex Program (MICP). Lemma 5.1 takes valid linear constraints for a graph optimization problem and automatically translates them into valid convex constraints for the corresponding GCS problem. In Theorem 5.1 we have applied this lemma to formulate a general-purpose MICP (5.8), that allows us to solve numerically any GCS problem. However, given a specific class of GCS problems, Lemma 5.1 can be used to derive additional valid constraints and increase the efficiency of our MICPs. This is shown in this chapter for a variety of practically relevant GCS problems. In addition, in this chapter we also illustrate multiple examples that demonstrate the wide applicability of the techniques developed in this thesis.

6.1 Shortest path

We have briefly described the classical Shortest-Path Problem (SPP) in Section 4.4.1. Here we study its extension in GCS. We anticipate that the SPP in GCS is NP-hard even for very simple cost functions and constraint sets (see Section 9.2).

The linear constraints (4.4) describe the polytope $\mathcal{Y}$ that allows us to formulate the directed SPP as an Integer Linear Program (ILP). Here we apply Lemma 5.1 to the inequalities defining this polytope to specialize the general-purpose MICP (5.8) to the directed SPP in GCS. This MICP was proposed in [131]. The convex constraints implied by the bound $y_v \leq 1$ and $y_e \geq 0$ are already included in the MICP (5.8) (see the second constraint in (5.8c) and the first constraint in (5.8d), respectively). The equalities in the flow conservation (4.4a) are amenable to Lemma 5.1 as described in

Remark 5.3. This yields the valid linear equalities

$$\boldsymbol{z}_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} \boldsymbol{z}_v^e + \delta_{sv} \boldsymbol{x}_v = \sum_{e \in \mathcal{I}_v^{\text{out}}} \boldsymbol{z}_v^e + \delta_{tv} \boldsymbol{x}_v, \qquad \forall v \in \mathcal{V}. \tag{6.1}$$

Lemma 5.1 can be applied to the cycle-elimination constraints in (4.4b) only when the subset $\mathcal{U}$ contains two vertices, i.e., the graph G has pairs of opposite edges $e = (v, w)$ and $f = (w, v)$. (This is the case, for example, if we formulate an undirected SPP as a directed SPP as discussed at the end of Section 4.4.1.) The resulting convex constraints are

$$(\boldsymbol{x}_v - \boldsymbol{z}_v^e - \boldsymbol{z}_v^f, 1 - y_e - y_f) \in \tilde{\mathcal{X}}_v, \qquad \forall e = (v, w) \in \mathcal{E} : f = (w, v) \in \mathcal{E}. \tag{6.2}$$

We can also apply Lemma 5.1 to the alternative cycle-elimination constraints (4.5) with $|\mathcal{U}| = 2$. We obtain

$$(\boldsymbol{z}_v - \boldsymbol{z}_v^e - \boldsymbol{z}_v^f, y_v - y_e - y_f) \in \tilde{\mathcal{X}}_v,\ (\boldsymbol{z}_w - \boldsymbol{z}_v^e - \boldsymbol{z}_v^f, y_w - y_e - y_f) \in \tilde{\mathcal{X}}_w, \\ \forall e = (v, w) \in \mathcal{E} : f = (w, v) \in \mathcal{E}. \tag{6.3}$$

The constraint in (6.2) is implied by the two constraints in (6.3): we simply sum to the latter the conditions $(\boldsymbol{x}_v - \boldsymbol{z}_v, 1 - y_v) \in \tilde{\mathcal{X}}_v$ and $(\boldsymbol{x}_w - \boldsymbol{z}_w, 1 - y_w) \in \tilde{\mathcal{X}}_w$ from (5.8c). This shows that the MICP featuring the constraints (6.3) is as strong or stronger than the one featuring (6.2).

Our overall MICP formulation of the SPP in GCS is obtained by adding to problem (5.8) the constraint (6.1) and one between constraint (6.2) and (6.3) (the second is typically more effective).

In Section 4.4.1 we have discussed how, if we assume the vertex and edge costs to be nonnegative, the cycle-elimination constraints are redundant for the ILP formulation of the ordinary SPP. For the SPP in GCS things are a little more complicated. If we assume that the functions f_v and f_e for $v \in \mathcal{V}$ and $e \in \mathcal{E}$ take nonnegative values, then the cycle-elimination constraints are unnecessary for the correctness of our mixed-integer formulation (as well as the corresponding constraints derived using Lemma 5.1). In fact, if they are unnecessary for the original ILP, then they are also unnecessary for the corresponding MINCP (5.2), which in turn is equivalent to the MICP (5.8) by Theorem 5.1. On the other hand, even under the nonnegativity assumption, these constraints can help making the convex relaxation of our MICP tighter.

6.1.1 Example: helicopter flight

We have an helicopter that is powered by solar energy and has to fly across an archipelago. The battery level over time is described by the function $b : \mathbb{R} \to [0, 1]$. The helicopter flies at constant speed $\sigma = 100$ and, when flying, its battery level decreases at rate $\alpha = 5$. It can stop at any time on any of the islands to recharge the battery. During the recharging breaks, the battery level increases at rate $\beta = 1$. The helicopter starts from the island indexed by $i = 1$ with full battery. This island is taken to be a circle with center $\boldsymbol{c}_1 = (0, 0)$ and zero radius $r_1 = 0$. The final island has index $i = 2$ and is a circle with center $\boldsymbol{c}_2 = (100, 100)$ and radius $r_2 = 0$. The other islands are indexed by $i = 3, \ldots, N$, with $N := 25$, and they are full dimensional circles

$$\mathcal{C}_i := \{\boldsymbol{q} \in \mathbb{R}^2 : \|\boldsymbol{q} - \boldsymbol{c}_i\|_2 \leq r_i\}.$$

Their centers $\boldsymbol{c}_i \in \mathbb{R}^2$ are drawn uniformly at random from the interval $[\boldsymbol{c}_1, \boldsymbol{c}_2]$ and the radii r_i are drawn uniformly at random from the interval $[0, 10]$. (Whenever we sample an island that intersects with one of the previous islands we reject it.) The goal is to complete the fight in minimum time. Time passes both when the helicopter is flying and when it is recharging the batteries.

The top panel in Figure 6-1 shows the optimal trajectory of the helicopter; the bottom panel shows the battery level b as a function of time. The optimal flight requires eight recharging breaks and takes a total time of 8.45.

We formulate the problem just described as an SPP in GCS. We construct a graph $G = (\mathcal{V}, \mathcal{E})$ with one vertex per island, i.e., $|\mathcal{V}| = N$. The source is the first island, $s := 1$, and the target is the second island, $t := 2$. We draw an edges between any pair of islands that are close enough for the helicopter to fly between them, assuming full battery at the beginning of the flight. In formulas, we have $(i, j) \in \mathcal{E}$ if

$$\|\boldsymbol{c}_j - \boldsymbol{c}_i\|_2 < r_i + r_j + \sigma/\alpha, \tag{6.4}$$

for all $i, j = 1, \ldots, N$ such that $i \neq j$.

Each vertex $i = 1, \ldots, N$ has two continuous variables:

- $\boldsymbol{q}_i \in \mathcal{C}_i$: recharge point in case the helicopter decides to stop on the ith island,
- $\boldsymbol{b}_i \in [0, 1]^2$: battery level before and after recharging on the island.

These variables are stacked to form the vector $\boldsymbol{x}_i := (\boldsymbol{q}_i, \boldsymbol{b}_i) \in \mathbb{R}^4$. The recharging time spent on the ith island is $h(\boldsymbol{b}_i) := (b_{i,2} - b_{i,1})/\beta$. The convex set $\mathcal{X}_i \subset \mathbb{R}^4$ paired with vertex i is equal to $\mathcal{C}_i \times [0, 1]^2$ with the additional constraint that $h(\boldsymbol{b}_i) \geq 0$. The

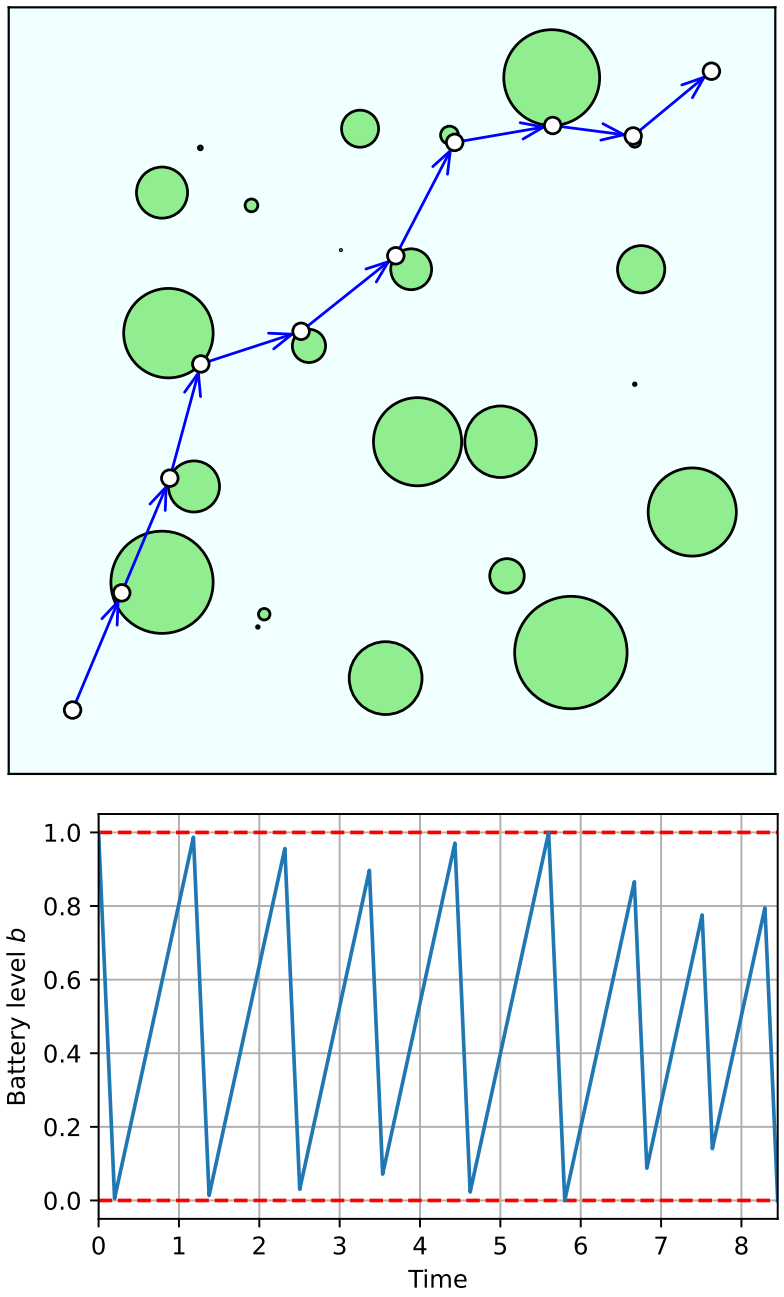

Figure 6-1: Example of an SPP in GCS. A helicopter powered by solar energy flies through an archipelago in minimum time. *Top:* optimal trajectory of the helicopter. *Bottom:* optimal battery level as a function of time.

set $\mathcal{X}_1$ also ensures that the battery is fully charged at the beginning of the flight, i.e., $b_{1,2} = 1$. The cost of vertex i is the linear function $f_i(\boldsymbol{x}_i) := h(\boldsymbol{b}_i)$. (Note that the variables $b_{1,1}$ and $b_{2,2}$ are actually irrelevant, and have the only role of simplifying the notation.)

Each edge $e = (i, j)$ has a cost equal to the flight time $f_e(\boldsymbol{x}_i, \boldsymbol{x}_j) := \|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2/\sigma$, and is paired with the convex set

$$\mathcal{X}_e := \{(\boldsymbol{x}_i, \boldsymbol{x}_j) : b_{j,1} \leq b_{i,2} - \alpha\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2/\sigma\}.$$

The latter constraint states that the battery level before charging on island j cannot be greater than the battery level after charging on island i, minus the battery consumed to fly between the islands. (Note that the slack in this inequality will always be zero at optimality, and enforcing the equality directly would not be convex.)

The convex relaxation of the SPP in GCS, without enforcing any cycle elimination constraint, has optimal value equal to 8.33 time units. Therefore the relaxation gap (cost of the MICP minus cost of the relaxation, normalized by the cost of the MICP) is only 1.4%. Essentially, the problem is solved through a single convex program; a Second-Order Cone Program (SOCP) in this case.

6.2 Travelling salesperson

We have introduced the Travelling-Salesperson Problem (TSP) in Section 4.4.2, here we consider its extension in GCS. The TSP in GCS is a generalization of the ordinary TSP, therefore it is NP-hard. Our goal is to formulate this problem as a practical MICP. As for the SPP, we start from the base MICP (5.8), and we derive specialized constraints for this problem using Lemma 5.1 and the ILP formulation of the TSP from Section 4.4.2.

We start by considering the directed TSP. We apply Lemma 5.1 to all the linear constraints in (4.7). The base MICP (5.8) already contains the convex constraints implied by the bounds $y_v \leq 1$ and $y_e \geq 0$. The equality constraints (4.7a) is amenable to Lemma 5.1 as described in Remark (5.3), and it gives us the following valid linear equalities for the TSP in GCS:

$$\boldsymbol{z}_v = \sum_{e \in \mathcal{I}_v^{\text{in}}} \boldsymbol{z}_v^e = \sum_{e \in \mathcal{I}_v^{\text{out}}} \boldsymbol{z}_v^e = \boldsymbol{x}_v, \qquad \forall v \in \mathcal{V}. \tag{6.5}$$

As seen for the SPP, the cycle-elimination constraint (4.7b) is amenable to Lemma 5.1 provided that the set $\mathcal{U}$ has only two elements. In particular, the constraints (6.2)

and (6.3) are also valid for the TSP in GCS.

For the undirected TSP we proceed similarly. The only difference is the linear constraint (6.5), which is now substituted with

$$\boldsymbol{z}_v = \frac{1}{2} \sum_{e \in \mathcal{I}_v} \boldsymbol{z}_v^e = \boldsymbol{x}_v, \qquad \forall v \in \mathcal{V}.$$

The latter constraint is obtained from the equality (4.8) as in Remark (5.3).

6.2.1 Example: optimal car pooling

There is a party in Manhattan and the host is going to pick up $N := 12$ guests with a van. The location of the party is $\boldsymbol{\theta}_0 := (45, 7)$, where the first coordinate is the street and the second is the avenue. The guests have initial coordinates $\boldsymbol{\theta}_i \in \mathbb{Z}^2$ for $i = 1, \dots, N$, and they are willing to move with a car to shorten the ride of the host. The goal is to find the optimal pick-up location of each guest so that the total gas consumption is minimized. The van of the host consumes two units of gas per unit of distance (distances are measured using the $\mathcal{L}_1$ norm). The cars of the guests consume half of that. Figure 6-2 depicts the optimal solution of this problem for a particular choice of the initial positions of the guests.

The problem can be formulated as a TSP in GCS as follows. We construct an undirected graph $G = (\mathcal{V}, \mathcal{E})$ with $|\mathcal{V}| = 13$ vertices; one represents the party, the remaining are the guests. The party vertex is labelled as zero and has continuous variable $\boldsymbol{x}_0$. The set $\mathcal{X}_0$ enforces the equality $\boldsymbol{x}_0 = \boldsymbol{\theta}_0$. The cost function f_0 is zero. The guest vertices are $i = 1, \dots, N$, and have variables $\boldsymbol{x}_i \in \mathbb{R}^2$. The sets $\mathcal{X}_i$ enforce the conservative bounds $\boldsymbol{x}_i \in [\boldsymbol{\theta}_{\min}, \boldsymbol{\theta}_{\max}]$, where $\boldsymbol{\theta}_{\min}$ and $\boldsymbol{\theta}_{\max}$ are the elementwise minimum and maximum of $\{\boldsymbol{\theta}_1, \dots, \boldsymbol{\theta}_N\}$. The cost function f_i is equal to the distance travelled by the guest, i.e., $\|\boldsymbol{x}_i - \boldsymbol{\theta}_i\|_1$. The graph is fully connected, and the cost of each edge $e = \{v, w\}$ is $f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := 2\|\boldsymbol{x}_v - \boldsymbol{x}_w\|_1$, where the coefficient takes into account the higher gas consumption of the host van.

Note that with the formulation just described the pick-up locations $\boldsymbol{x}_i$ of the guests are allowed to be fractional. However, it can be verified that the optimal value of these variables is always integer valued. This is because of the $\mathcal{L}_1$ metric used in the cost functions, and the integral values of the initial positions $\boldsymbol{\theta}_i$.

The optimal value of the problem is 39, and the one of the convex relaxation (including the exponentially many cycle-elimination constraints) is 24.1. For this problem the relaxation is not as tight as for the helicopter-flight example, since the ILP formulation of the TSP is much weaker than the one of the SPP (which is exact

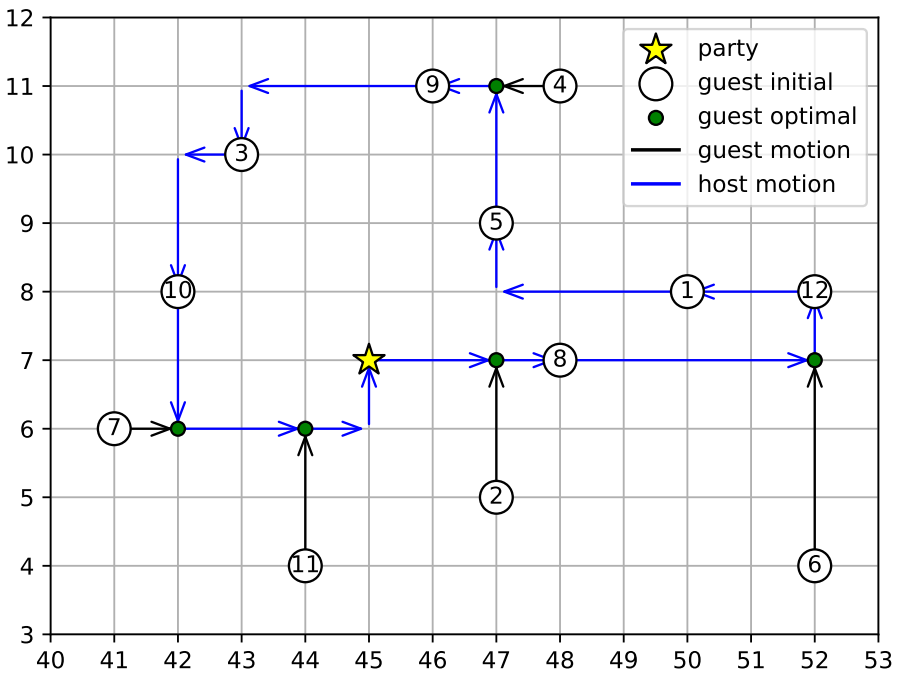

Figure 6-2: Optimal car pooling as a TSP in GCS. A host picks up $N := 12$ guests for a party. The goal is to minimize the total fuel consumption, and the guests are willing to move towards the host in order to reduce the total cost of the trip.

for nonnegative weights).

6.3 Minimum spanning tree

We extend the Minimum-Spanning-Tree Problem (MSTP) introduced in Section 4.4.3 to the GCS setting, and we formulate it as an MICP. The MSTP in GCS is a generalization of the MSTP with neighborhoods, therefore it inherits the NP-hardness of the latter problem [193, Theorem 1].

As for the previous problems we start from the base formulation (5.8). By applying Remark (5.3) on the equality constraint (4.9a) we find the additional linear equality

$$\boldsymbol{z}_v = \boldsymbol{x}_v, \qquad \forall v \in \mathcal{V}.$$

Constraint (4.10) for $|\mathcal{U}| = 1$ is amenable to Lemma 5.1 and gives:

$$\left(\sum_{e \in \mathcal{I}_v} \boldsymbol{z}_v^e - \boldsymbol{x}_v, \sum_{e \in \mathcal{I}_v} y_e - 1 \right) \in \tilde{\mathcal{X}}_v, \qquad \forall v \in \mathcal{V}.$$

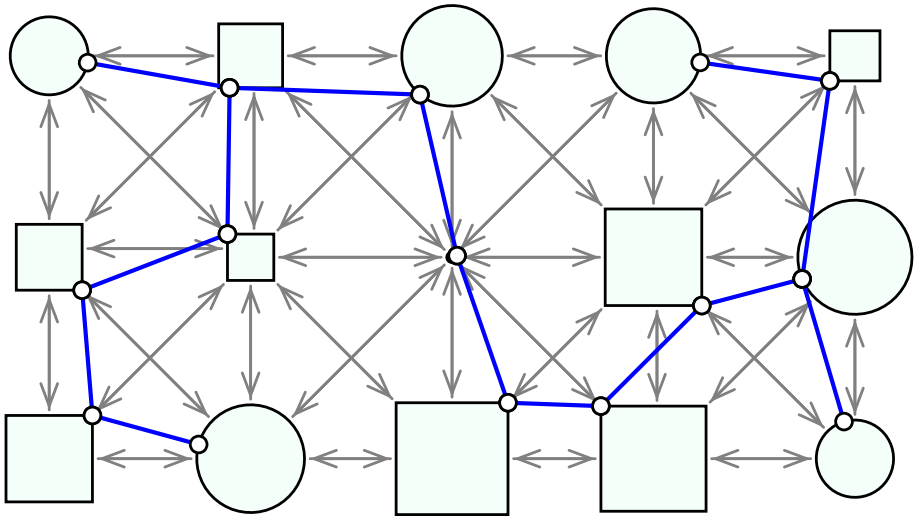

Figure 6-3: Design of a power network using the MSTP in GCS. Cities are convex sets and are connected to the other cities through electricity cables (blue). The goal is to minimize the total length of the electricity grid.

All the other constraints that are not variables bounds (constraints (4.9b) and (4.9c)) involve more than one vertex, and are not suitable for Lemma 5.1.

6.3.1 Example: power network design

We consider an MSTP in GCS where $N := 15$ cities need to be connected by electricity cables, and we need to decide where each city should be latched to the grid. The cities are disposed on a uniform gird with integer coordinates. A city is represented as a two-dimensional set: a circle or a square with equal probability. The diameter of the circles and the sides of the squares are chosen uniformly at random from the interval $[0, 0.6]$. The goal is to minimize the total length of the electricity grid. Figure 6-3 shows the optimal solution of this problem.

The problem just described is formulated as an MSTP in GCS very easily. The vertices are the cities, and the edges connect the cities as shown in Figure 6-3. The continuous variables are the latching locations within each city. Vertices have zero cost and edges have cost equal to the Euclidean distance of the points that they connect. Edges are not associated with any constraint.

The optimal cost of the MICP is 10.00, and the one of the convex relaxation (including all the cycle-elimination constraints) is 9.00. As for the SPP, the convex relaxation is very accurate for this problem. Recall that both the SPP and the MSTP

can be solved in polynomial time. Additionally, the base ILP formulations of these problems, which our MICP builds on, have exact convex relaxation.

6.4 Facility location

The Facility-Location Problem (FLP) was described in Section 4.4.4. Since the FLP is NP-hard, so is the FLP in GCS. The application of Remark (5.3) and Lemma 5.1 to the constraints (4.11a) and (4.11b) gives

$$\begin{aligned} &\boldsymbol{z}_v = \sum_{e \in \mathcal{I}_v} \boldsymbol{z}_v^e = \boldsymbol{x}_v, && \forall v \in \mathcal{V}_1, \\ &(\boldsymbol{z}_v - \boldsymbol{z}_v^e, y_v - y_e) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V}_2,\ e \in \mathcal{I}_v. \end{aligned}$$

Added to the base MICP (5.8), these constraints yield our MICP formulation of the FLP in GCS.

6.4.1 Example: sphere cover for robot collision checking

Collision checks are a very common problem in robotics. Consider, for example, a robot arm with multiple revolute joints. Given a configuration of the arm (i.e., given the angle of every joint), we want to be sure that every pair of robot links is not in collision. Exact collision checks can be computationally expensive, since the robot links might have complex shape. A common workaround is then to construct an outer approximation of each link using simpler shapes (typically spheres). We formulate this outer-approximation problem as an FLP in GCS.

Consider the simple two-dimensional robot link illustrated in Figure 6-4. As it is often the case, the link of our robot is described by a triangular mesh: $\mathcal{T}_i \subset \mathbb{R}^2$ for $i = 1, \dots, N$. The number of triangles in the figure is $N = 17$. Given that the problem is in two dimensions, we seek a collection of circles $\mathcal{C}_j \subset \mathbb{R}^2$ for $j = 1, \dots, M$. Each triangle $\mathcal{T}_i$ must be entirely contained in at least one circle $\mathcal{C}_j$. Among all the possible solutions, we seek the one that minimizes the sum of the areas of the circles. We optimize both the shape of the circles and their number. The optimal solution is reported in Figure 6-4, and uses only $M = 5$ circles to cover all the triangles in the mesh.

We formulate the problem above as an FLP in GCS. Each triangle $\mathcal{T}_1, \dots, \mathcal{T}_N$ is a client that needs to be covered by a facility. Each facility represents a potential circle used to cover the triangles. Conservatively, we take the number of facilities to be equal to the number N of clients, since we might need one circle for each triangle

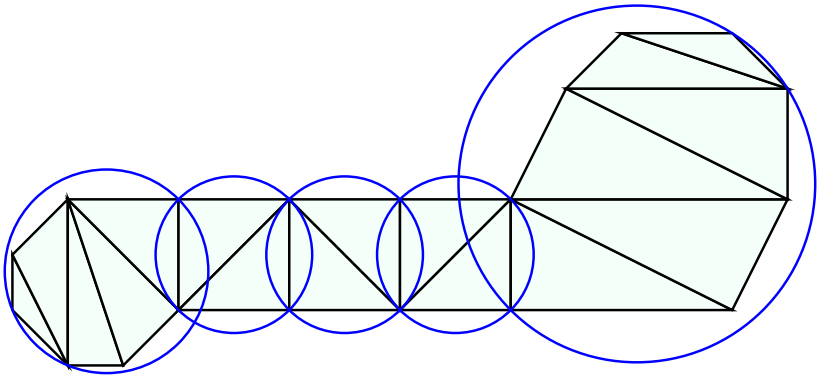

Figure 6-4: Minimum-volume cover of a robot link using circles.

in the worst case. The solution of the FLP in GCS will then automatically select the optimal number of facilities (i.e., circles) to employ. The continuous variable $\boldsymbol{x}_i \in \mathbb{R}^6$ paired with a client has fixed value, and stacks the three two-dimensional vertices of the triangle $\mathcal{T}_i$. Each facility is paired with the vector $\boldsymbol{x}_j \in \mathbb{R}^3$, that stacks the center $\boldsymbol{c}_j \in \mathbb{R}^2$ and the radius $r_j \in \mathbb{R}_{\geq 0}$ of the circle $\mathcal{C}_j$. The positions of the centers and the radii are bounded by computing the smallest axis-aligned bounding box that contains all the triangles. The costs of the clients are zero, while the cost of the jth facility is equal to πr_j^2 (the area of the corresponding circle). For every edge connecting a client i to a facility j, we enforce the convex constraint that all the vertices of the triangle $\mathcal{T}_i$ must lie in the circle $\mathcal{C}_j$. This ensures that $\mathcal{T}_i \subseteq \mathcal{C}_j$. The costs of all the edges are set to zero.

In this case, the MICP and the convex relaxation have optimal value 62.1 and 19.6, respectively. This relatively large gap is due to the convex relaxation of the ILP formulation (4.11) of the ordinary FLP, which can be quite loose to start with.

6.5 Minimum perfect matching

The Minimum-Perfect-Matching Problem (MPMP) was described in Section 4.4.5. The MPMP in GCS can be formulated as an MICP by proceeding as for the previous problems. In particular, the application of Remark 5.3 to constraint (4.12a) gives us

$$\boldsymbol{z}_v = \sum_{e \in \mathcal{I}_v} \boldsymbol{z}_v^e = \boldsymbol{x}_v, \qquad \forall v \in \mathcal{V}.$$

The constraints in (4.13) cannot be leveraged by Lemma 5.1.

Mixed-integer optimization is not a practical way of solving the MPMP in GCS. In fact, the MPMP in GCS is easily reduced to an ordinary MPMP, with scalar costs on the vertices and edges, and solved using, e.g., Edmonds' matching algorithm. For each vertex $v \in \mathcal{V}$ in our graph we define $c_v := 0$. For every edge $e = (v, w) \in \mathcal{E}$, we let c_e be the optimal value of the following convex program:

$$\begin{aligned} \text{minimize} \quad & f_v(\boldsymbol{x}_v) + f_w(\boldsymbol{x}_w) + f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) \\ \text{subject to} \quad & \boldsymbol{x}_v \in \mathcal{X}_v, \\ & \boldsymbol{x}_w \in \mathcal{X}_w, \\ & (\boldsymbol{x}_v, \boldsymbol{x}_w) \in \mathcal{X}_e. \end{aligned}$$

(If this program is infeasible we remove the edge e from the graph.) These scalar vertex and edge costs give us a weighted graph whose minimum perfect matching is the same (and has the same cost) as the minimum perfect matching in the GCS. This because for any fixed perfect matching, the continuous variables $\boldsymbol{x}_v$ and $\boldsymbol{x}_w$ are completely decoupled if $e = (v, w)$ is not in the matching, and are optimal if they solve the convex program above. Solving the convex programs above takes a time that is only linear in the number $|\mathcal{E}|$ of edges. Therefore, contrarily to all the GCS problems seen above, the MPMP in GCS is efficiently solvable.

Given that solving an MPMP in GCS does not require the techniques developed in this thesis, we skip the illustrative example for this problem.

6.6 Inspection problem

Our technique for solving GCS problems is fully algorithmic. Given the GCS and the ILP formulation of the discrete problem, our MICP is assembled and solved automatically. Approaching a new GCS problem is then very easy: our modelling effort is limited to the definition of the GCS and the design of the ILP. We demonstrate this point through the following example.

Consider the inspection problem shown in Figure 6-5. We have a floor plant composed of $N := 19$ rooms. Solid black lines represent walls and dotted linear are open doors. We seek a shortest closed trajectory that visits all the $M := 7$ rooms in red. Green rooms need not to be visited by the trajectory. Formulating this problem directly as an MICP would be very tedious and error prone: with GCS this problem is easily formulated and solved.

The inspection problem in Figure 6-5 is reminiscent of a TSP in GCS, where each room is a vertex in a graph. However, the optimal inspection trajectory must not visit

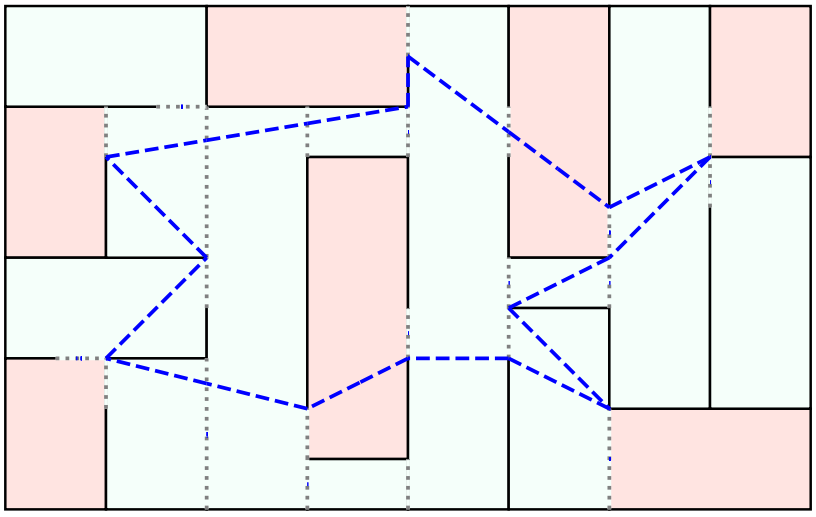

Figure 6-5: Inspection problem. We seek a continuous closed trajectory (dashed blue) that visits all the red rooms in the floor plant. Solid black lines represent walls and dotted lines are doorways. Green rooms need not to be visited by the trajectory.

all the rooms in the floor plan, and might also go through the same room multiple times. Therefore, the GCS problem we formulate is slightly more involved than a TSP.

We construct our GCS as follows. We imagine a fictitious building where the floor plan in Figure 6-5 is repeated on M floors. We start from the ground floor in the first red room (picked arbitrarily). Every time that we reach a new red room we move one floor up. Our building has a total of MN rooms, each of which is a vertex in our GCS. Each vertex is paired with two points in the corresponding room: the first is the entry point and the second is the exit point. The cost of each vertex is the distance between the two points (i.e., the distance travelled inside the room). For each floor, we draw an edge between any pair of communicating rooms. When we travel along such an edge, the exit point of the last room must be equal to the entry point of the next room. We also add edges that connect consecutive floors: any red room (that is not the first) is connected to the corresponding room upstairs. The first room at the top floor is then connected to the first room at the ground floor. Along all these edges, we enforce the same continuity constraints as before.

The ILP is formulated through the following steps. Every vertex must verify the flow conservation: if we enter in a room we must also leave the room. The constraint that each room must be visited at least once is enforced as follows: the sum over all floors of the binary variables that connect that room to its copy upstairs must

be greater than or equal to one. The binary variable of the edge connecting the top floor to the ground floor must also be at least one. This last constraint closes (i.e., connects the start and the end of) our trajectory.

The GCS and the ILP just constructed are processed algorithmically to solve the inspection problem and compute the trajectory in in Figure 6-5.

Chapter 7

Software implementation

In this chapter we describe the software package `gcspy` that we have developed for solving problems in Graphs of Convex Sets (GCS). This package is written in Python and based on `cvxpy` [50]. It is freely available at

`https://github.com/TobiaMarcucci/gcspy`.

It is not a fully mature code base yet, but its structure is very simple and well illustrates the easiness with which the techniques introduced in this thesis can be implemented on a computer. Optimizing the implementation of `gcspy` will be object of future works. We also highlight that a mature implementation of our techniques is available within the open-source software `Drake` [183]. This second implementation is currently specialized to the Shortest-Path Problem (SPP) in GCS, and its extension to any GCS problem is under development.

7.1 Interface

We start by describing the syntax of `gcspy`. We take the example of the helicopter flight described in Section 6.1.1, and we show how to solve this SPP in GCS with our package one step at the time.

We start by importing the packages `gcspy` and `cvxpy`.

```
import gcspy as gp # our package
import cvxpy as cp # convex-optimization package
```

We define the numerical parameters of our problem following the notation from Section 6.1.1.

```
alpha = 5 # battery discharge rate
beta = 1 # battery charge rate
sigma = 100 # moving speed
```

Next we specify the center and the radius of each island. Except for the first two islands (the source and the target), these are originally generated at random. Here, for simplicity, we directly define their values rounded to the second digit.

```
C = ((0, 0), (100, 100), (78,  9), (37, 57), (89, 69),
     (42, 72), (30, 15), (19, 35), (54, 42), (20, 88),
     (67, 42), (14, 20), (97, 31), (88, 89), (53, 69),
     (88, 51), (75, 99), (28, 79), (45, 91), (29, 13),
     (68, 21), (49,  5), (15, 59), (59, 90), (14, 81)) # centers
r = (0.0, 0.0, 8.8, 2.6, 3.7,
     0.1, 0.9, 4.0, 6.9, 0.3,
     5.6, 8.0, 6.9, 0.9, 3.2,
     0.2, 7.5, 1.0, 2.9, 0.2,
     2.7, 5.7, 7.0, 1.4, 4.0) # radii
```

Next we instantiate our GCS.

```
gcs = gp.GraphOfConvexSets()
```

We add the vertices to the GCS. Each vertex has a continuous variable $\boldsymbol{q}_i \in \mathcal{C}_i$ that represents where the helicopter should stop for a recharging break on the island. The second variable $\boldsymbol{b}_i \in [0,1]^2$ is the battery level before and after the recharging break. The recharging time $(b_{i,2} - b_{i,1})/\beta$ is nonnegative and equal to the cost of each vertex. Finally, we also ensure that the battery is fully charged at the beginning of the flight.

```
N = len(C) # number of islands
for i in range(N):
    v = gcs.add_vertex(i) # ith vertex
    q = v.add_variable(2) # recharging point
    b = v.add_variable(2) # battery levels
    v.add_constraint(cp.norm2(q - C[i]) <= r[i]) # point in island
    v.add_constraint(b >= 0)
    v.add_constraint(b <= 1)
    h = (b[1] - b[0]) / beta # recharging time
    v.add_constraint(h >= 0)
    v.add_cost(h) # penalize recharging time
    if i == 0: # source island
        v.add_constraint(b[1] == 1) # initial battery is full
```

We now specify the edges of our graph (currently, `gcspy` supports only directed graphs). As in (6.4), we connect every pair of distinct vertices such that the minimum distance between the corresponding islands can be covered by the helicopter. To facilitate this check, we define the following helper function.

```
autonomy = sigma / alpha # maximum flying range
def reach(i, j): # checks if island j is reachable from island i
    dist = (C[j][0] - C[i][0]) ** 2
    dist += (C[j][1] - C[i][1]) ** 2
    dist **= .5 # distance between island centers
    return autonomy >= dist - r[i] - r[j]
```

Using the function above, we iterate over all pairs of vertices and decide whether to connect them with an edge or not. The cost of traversing an edge (i, j) is equal to the duration of the corresponding flight segment, i.e., $\|\boldsymbol{q}_j - \boldsymbol{q}_i\|_2/\sigma$. We also add a constraint that relates the battery levels on the islands i and j.

```
for i in range(N):
    vi = gcs.get_vertex_by_name(i) # retrieve ith vertex
    qi, bi = vi.variables # retrieve vertex variables
    for j in range(N):
        if i != j and reach(i, j): # island j is reachable from i
            vj = gcs.get_vertex_by_name(j) # retrieve jth vertex
            qj, bj = vj.variables # retrieve vertex variables
            e = gcs.add_edge(vi, vj) # edge from i to j
            te = cp.norm2(qi - qj) / sigma # flight time
            be = alpha * te # battery consumption
            e.add_constraint(bj[0] <= bi[1] - be)
            e.add_cost(te) # penalize flight time
```

Now we can solve the SPP in GCS and print, e.g., the optimal value of the problem.

```
s = gcs.get_vertex_by_name(0) # source vertex
t = gcs.get_vertex_by_name(1) # target vertex
sol = gcs.solve_shortest_path(s, t) # assembles MICP and solves it
print(sol.value) # optimal value of the SPP in GCS
```

The optimal value of the binary variables y_v and y_e paired with the vertices $v \in \mathcal{V}$ and edges $e \in \mathcal{E}$ can be printed as follows.

```
for v in gcs.vertices:
    print(v.y.value) # vertex binary
for e in gcs.edges:
    print(e.y.value) # edge binary
```

The optimal value of the variables $\boldsymbol{x}_v$ paired with the vertices $v \in \mathcal{V}$ are retrieved similarly. (Note that every vertex is paired with a list of decision variables in `gcspy`. The variable $\boldsymbol{x}_v$ can be thought of as the concatenation of this list.)

```
for v in gcs.vertices:
    for xv in v.variables:
        print(xv.value) # vertex continuous variable
```

7.1.1 Solving new GCS problems

In the code snippets above, we called the method `solve_shortest_path` of the class `GraphOfConvexSets` to compute the shortest path between the source and the target. This method formulates the MICP described in Section 6.1, solves it, and gives us back the solution in the form of the object `sol`. Methods that can be called to solve other GCS problems include `solve_traveling_salesperson`, `solve_spanning_tree`, and `solve_facility_location`. On the other hand, the techniques introduced in this thesis are fully algorithmic and apply to any graph optimization problem, provided that we are given an Integer-Linear Programming (ILP) formulation of it, i.e., the polytope $\mathcal{Y}$.

In the following code snippet we pretend that the method `solve_shortest_path` was not implemented in `gcspy`, and show how we can still solve the SPP in GCS by providing a description of the polytope $\mathcal{Y}$.

```
Y = [] # list of linear inequalities of ILP
for v, yv in zip(gcs.vertices, gcs.vertex_binaries):
    edges_in = gcs.incoming_indices(v) # edges incoming v
    edges_out = gcs.outgoing_indices(v) # edges outgoing v
    flow_in = gcs.edge_binaries[edges_in] # flows incoming v
    flow_out = gcs.edge_binaries[edges_out] # flows outgoing v
    delta_sv = 1 if v == s else 0
    delta_tv = 1 if v == t else 0
    Y.append(yv == sum(flow_in) + delta_sv) # flow conservation
    Y.append(yv == sum(flow_out) + delta_tv) # flow conservation
sol = gcs.solve_from_ilp(Y)
```

In these lines of code we describe the polytope $\mathcal{Y}$ from (4.4) by listing the linear constraints that define its facets (i.e., the constraints of the ILP formulation of the SPP). For the SPP, these constraints are the flow conservation (which is computed within the for loop for each vertex) and the bounds $y_v \in [0, 1]$ and $y_e \in [0, 1]$ (which are omitted since they hold for any problem and they are automatically added by `gcspy`). Then we call the method `solve_from_ilp` which parses these constraints, automatically formulates our MICP using Lemma 5.1, solves it, and returns the solution object.

7.2 Behind the scenes

The software implementation described above is made very simple by the DCP framework introduced in [85], and briefly described in Section 2.3.3. DCP allows us to specify the convex sets and functions in a GCS using the high-level syntax of a software like `cvxpy`. Then it computes conic representations of these sets and functions. Once all the pieces of our problem are in conic form, it is very easy for us to apply all the necessary homogenization transformations and assemble our MICP. Let us describe this process in more detail.

Every vertex $v \in \mathcal{V}$ in a GCS is paired with a convex program, specified by a convex set $\mathcal{X}_v \subset \mathbb{R}^{n_v}$ and a convex function $f_v : \mathbb{R}^{n_v} \to \mathbb{R}$. This program is coded using the DCP ruleset, and is translated in conic form automatically (potentially through the addition of auxiliary variables and constraints). Therefore, without loss of generality, we can assume that the vertex cost is linear,

$$f_v(\boldsymbol{x}_v) := \boldsymbol{c}_v^\top \boldsymbol{x}_v$$

for some $\boldsymbol{c}_v \in \mathbb{R}^{n_v}$, and the vertex set is in conic form,

$$\mathcal{X}_v := \{\boldsymbol{x}_v : \boldsymbol{A}_v \boldsymbol{x}_v + \boldsymbol{b}_v \in \mathcal{K}_v\}$$

for some matrix $\boldsymbol{A}_v \in \mathbb{R}^{m_v \times n_v}$, vector $\boldsymbol{b}_v \in \mathbb{R}^{m_v}$, and convex cone $\mathcal{K}_v \subset \mathbb{R}^{m_v}$.

The same holds for the edges. Each edge $e = [v, w] \in \mathcal{E}$ is paired with a convex program, specified by the set $\mathcal{X}_e \subseteq \mathbb{R}^{n_v + n_w}$ and the function $f_e : \mathbb{R}^{n_v+n_w} \to \mathbb{R}$. Again, these convex sets and functions are transformed into conic form. Therefore, we can assume that

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \boldsymbol{c}_e^\top (\boldsymbol{x}_v, \boldsymbol{x}_w)$$

for some $\boldsymbol{c}_e \in \mathbb{R}^{n_v+n_w}$, as well as

$$\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w) : \boldsymbol{A}_e(\boldsymbol{x}_v, \boldsymbol{x}_w) + \boldsymbol{b}_e \in \mathcal{K}_e\}$$

for some matrix $\boldsymbol{A}_e \in \mathbb{R}^{m_e \times (n_v+n_w)}$, vector $\boldsymbol{b}_e \in \mathbb{R}^{m_e}$, and convex cone $\mathcal{K}_e \subset \mathbb{R}^{m_e}$.

With all the costs and constraints expressed in conic form, our MICP (5.8) is formulated as follows. We define the binary variables $y_v \in \{0, 1\}$ and $y_e \in \{0, 1\}$ for each vertex $v \in \mathcal{V}$ and edge $e \in \mathcal{E}$. We introduce the auxiliary variables $\boldsymbol{z}_v \in \mathbb{R}^{n_v}$ and $\boldsymbol{z}_v^e \in \mathbb{R}^{n_v}$ for each vertex $v \in \mathcal{V}$ and edge $e \in \mathcal{I}_v$. Since all the cost functions are linear, the objective function of our MICP is simply a linear function of the auxiliary

variables:

$$\sum_{v\in\mathcal{V}} \tilde{f}_v(\boldsymbol{z}_v, y_v) + \sum_{e\in\mathcal{E}} \tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) = \sum_{v\in\mathcal{V}} \boldsymbol{c}_v^\top \boldsymbol{z}_v + \sum_{e\in\mathcal{E}} \boldsymbol{c}_e^\top (\boldsymbol{z}_v^e, \boldsymbol{z}_w^e).$$

(Recall that the homogenization of a linear function is equal to the function itself.)

The constraint set $\mathcal{Y}$ is either constructed within the specific method called by the user (e.g., `solve_shortest_path`) or is provided by the user as shown in Section 7.1.1. Therefore the constraint $\boldsymbol{y} \in \mathcal{Y}$ from (5.8b) is easily assembled.

Given the conic representation of the sets $\mathcal{X}_v$, their homogenizations take the form

$$\tilde{\mathcal{X}}_v = \{(\boldsymbol{x}, y) : y \geq 0,\ \boldsymbol{A}_v\boldsymbol{x} + \boldsymbol{b}_v y \in \mathcal{K}_v\}.$$

The constraints $(\boldsymbol{z}_v, y_v) \in \tilde{\mathcal{X}}_v$ and $(\boldsymbol{x}_v - \boldsymbol{z}_v, 1 - y_v) \in \tilde{\mathcal{X}}_v$ from (5.8c) are then enforced simply as

$$\boldsymbol{A}_v\boldsymbol{z}_v + \boldsymbol{b}_v y_v \in \mathcal{K}_v, \qquad \boldsymbol{A}_v(\boldsymbol{x}_v - \boldsymbol{z}_v) + \boldsymbol{b}_v(1 - y_v) \in \mathcal{K}_v.$$

(Note that the bounds $0 \leq y_v \leq 1$ are already enforced by the constraint $\boldsymbol{y} \in \mathcal{Y}$.) The constraints from (5.8d) and (5.8e) are implemented similarly:

$$\begin{aligned}
&\boldsymbol{A}_v\boldsymbol{z}_v^e + \boldsymbol{b}_v y_e \in \mathcal{K}_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v,\\
&\boldsymbol{A}_v(\boldsymbol{x}_v - \boldsymbol{z}_v^e) + \boldsymbol{b}_v(1 - y_e) \in \mathcal{K}_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v,\\
&\boldsymbol{A}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e) + \boldsymbol{b}_e y_e \in \mathcal{K}_e, && \forall e = [v, w] \in \mathcal{E}.
\end{aligned}$$

Finally, in Chapter 6 we have seen how Lemma 5.1 can be used to generate constraints that are specialized to a given graph problem. These are also enforced as above. Thanks to the conic representation of the GCS convex programs, any constraint of the form

$$\left(a_v\boldsymbol{z}_v + \sum_{e\in\mathcal{I}_v} a_e\boldsymbol{z}_v^e + b\boldsymbol{x}_v, a_v y_v + \sum_{e\in\mathcal{I}_v} a_e y_e + b\right) \in \tilde{\mathcal{X}}_v$$

becomes

$$\boldsymbol{A}_v\left(a_v\boldsymbol{z}_v + \sum_{e\in\mathcal{I}_v} a_e\boldsymbol{z}_v^e + b\boldsymbol{x}_v\right) + \boldsymbol{b}_v\left(a_v y_v + \sum_{e\in\mathcal{I}_v} a_e y_e + b\right) \in \mathcal{K}_v.$$

7.2.1 Edge variables

The package `gcspy` allows the explicit use of auxiliary variables $\boldsymbol{x}_e \in \mathbb{R}^{n_e}$ for each edge $e \in \mathcal{E}$, as discussed in Remark 5.1. These can be very useful to define certain classes

of convex costs and constraints. Therefore, the edge cost function really handled by `gcspy` is

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_w, \boldsymbol{x}_e) := \boldsymbol{c}_e^\top(\boldsymbol{x}_v, \boldsymbol{x}_w, \boldsymbol{x}_e)$$

with $\boldsymbol{c}_e \in \mathbb{R}^{n_v+n_w+n_e}$. While the constraint set is

$$\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w, \boldsymbol{x}_e) : \boldsymbol{A}_e(\boldsymbol{x}_v, \boldsymbol{x}_w, \boldsymbol{x}_e) + \boldsymbol{b}_e \in \mathcal{K}_e\}$$

with $\boldsymbol{A}_e \in \mathbb{R}^{m_e \times (n_v+n_w+n_e)}$. All the steps above are essentially unchanged after the introduction of a variable $\boldsymbol{z}_e$ that represents the product of y_e and $\boldsymbol{x}_e$.

Chapter 8

Analysis of the convex relaxation

In this chapter we describe and analyze at a more abstract level the method introduced in Chapter 5 to formulate a Graph of Convex Sets (GCS) problem as a Mixed-Integer Convex Program (MICP). We show that Lemma 5.1 can be used to design convex relaxations of a large class of bilinear constraints, and we connect this result to existing relaxation techniques for nonconvex optimization. Finally, we give a simple geometric proof of the validity of our MICP (already shown in Theorem 5.1). Most of the material in this chapter is taken from [131, Section 7].

8.1 Set-based relaxation of bilinear constraints

Our first step in this analysis is to show that Lemma 5.1 is, in fact, a general-purpose relaxation technique for nonconvex sets of the form

$$\mathcal{S} := \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z}) : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} \in \mathcal{Y},\ \boldsymbol{Z} = \boldsymbol{x}\boldsymbol{y}^\top\}, \tag{8.1}$$

where $\mathcal{X} \subseteq \mathbb{R}^n$ and $\mathcal{Y} \subseteq \mathbb{R}^m$ are closed convex sets. In particular, here $\mathcal{X}$ and $\mathcal{Y}$ represent generic constraint sets on the continuous and binary variables in the Mixed-Integer NonConvex Program (MINCP) (5.4), respectively. See Section 8.5 below for more details.

A natural approach to construct a convex relaxation of the set $\mathcal{S}$ is to multiply all the valid inequalities $\boldsymbol{a}^\top\boldsymbol{x} + b \geq 0$ for the set $\mathcal{X}$ by all the valid inequalities $\boldsymbol{c}^\top\boldsymbol{y} + d \geq 0$ for the set $\mathcal{Y}$, and then use the bilinear equality $\boldsymbol{Z} = \boldsymbol{x}\boldsymbol{y}^\top$ to linearize these products. This gives us an infinite family of valid linear inequalities for $\mathcal{S}$, which

form the convex set

$$\mathcal{C} := \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z}) : \boldsymbol{a}^\top \boldsymbol{Z}\boldsymbol{c} + d\boldsymbol{a}^\top \boldsymbol{x} + b\boldsymbol{c}^\top \boldsymbol{y} + bd \geq 0 \\ \text{for all } (\boldsymbol{a}, b) \in \mathcal{X}^\circ \text{ and } (\boldsymbol{c}, d) \in \mathcal{Y}^\circ\}, \quad (8.2)$$

where $\mathcal{X}^\circ$ and $\mathcal{Y}^\circ$ are the polar sets defined in Section 2.1.2. By construction, the set $\mathcal{C}$ is a convex relaxation of the nonconvex set $\mathcal{S}$:

$$\mathcal{S} \subseteq \mathcal{C}.$$

Note also that the conditions $\boldsymbol{x} \in \mathcal{X}$ and $\boldsymbol{y} \in \mathcal{Y}$ are implied by the linear inequalities in (8.2). In fact, for the valid inequality $(\boldsymbol{c}, d) = (\boldsymbol{0}, 1) \in \mathcal{Y}^\circ$, the conditions in (8.2) reduce to $\boldsymbol{a}^\top \boldsymbol{x} + b \geq 0$ for all $(\boldsymbol{a}, b) \in \mathcal{X}^\circ$, which is equivalent to $\boldsymbol{x} \in \mathcal{X}$, as we have seen in (2.8). The condition $\boldsymbol{y} \in \mathcal{Y}$ is recovered similarly by considering the valid inequality $(\boldsymbol{a}, b) = (\boldsymbol{0}, 1) \in \mathcal{X}^\circ$.

The relaxation (8.2) is not implementable on a computer yet, since it involves an infinite number of constraints. However, if one of the two sets is a polytope then the convex set $\mathcal{C}$ can be efficiently described by a finite number of convex constraints.

Theorem 8.1. *Let the set*

$$\mathcal{Y} := \{\boldsymbol{y} : \boldsymbol{c}_j^\top \boldsymbol{y} + d_j \geq 0 \text{ for all } j \in \mathcal{J}\}$$

be a polyhedron. We have

$$\mathcal{C} = \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z}) : \boldsymbol{x} \in \mathcal{X},\ (\boldsymbol{Z}\boldsymbol{c}_j + d_j\boldsymbol{x},\ \boldsymbol{c}_j^\top \boldsymbol{y} + d_j) \in \tilde{\mathcal{X}} \text{ for all } j \in \mathcal{J}\}. \quad (8.3)$$

Proof. We start from the definition in (8.2). First we add the redundant constraint $\boldsymbol{x} \in \mathcal{X}$. Then we notice that listing all the valid inequalities $(\boldsymbol{c}, d) \in \mathcal{Y}^\circ$ is equivalent to listing only the valid inequalities $(\boldsymbol{c}_j, d_j)$ for $j \in \mathcal{J}$. In fact, by Lemma 2.3, for any vector $(\boldsymbol{c}, d) \in \mathcal{Y}^\circ$ there exist nonnegative coefficients α_j such that $\boldsymbol{c} = \sum_{j\in\mathcal{J}} \alpha_j \boldsymbol{c}_j$ and $d \geq \sum_{j\in\mathcal{J}} \alpha_j d_j$. Using these coefficients, we have

$$\begin{aligned} 0 &\leq \sum_{j\in\mathcal{J}} \alpha_j (\boldsymbol{a}^\top \boldsymbol{Z}\boldsymbol{c}_j + d_j \boldsymbol{a}^\top \boldsymbol{x} + b\boldsymbol{c}_j^\top \boldsymbol{y} + bd_j) \\ &= \boldsymbol{a}^\top \boldsymbol{Z}\boldsymbol{c} + b\boldsymbol{c}^\top \boldsymbol{y} + \sum_{j\in\mathcal{J}} \alpha_j d_j (\boldsymbol{a}^\top \boldsymbol{x} + b) \\ &\leq \boldsymbol{a}^\top \boldsymbol{Z}\boldsymbol{c} + b\boldsymbol{c}^\top \boldsymbol{y} + d\boldsymbol{a}^\top \boldsymbol{x} + bd, \end{aligned}$$

where the last inequality uses the nonnegativity of $\boldsymbol{a}^\top \boldsymbol{x} + b$, which is ensured by $(\boldsymbol{a}, b) \in \mathcal{X}^\circ$ and $\boldsymbol{x} \in \mathcal{X}$. Finally, for each $j \in \mathcal{J}$, we rewrite the infinite family of linear inequalities

$$0 \leq \boldsymbol{a}^\top \boldsymbol{Z}\boldsymbol{c}_j + d_j \boldsymbol{a}^\top \boldsymbol{x} + b\boldsymbol{c}_j^\top \boldsymbol{y} + bd_j = \boldsymbol{a}^\top (\boldsymbol{Z}\boldsymbol{c}_j + d_j \boldsymbol{x}) + b(\boldsymbol{c}_j^\top \boldsymbol{y} + d_j),$$

for all $(\boldsymbol{a}, b) \in \mathcal{X}^\circ$, as a single convex constraint $(\boldsymbol{Z}\boldsymbol{c}_j + d_j \boldsymbol{x}, \boldsymbol{c}_j^\top \boldsymbol{y} + d_j) \in \tilde{\mathcal{X}}$. This step uses again the equality (2.8). □

We then have two descriptions of the relaxation $\mathcal{C}$: the symmetric one (8.2) that clearly exposes the logic behind our technique, and the asymmetric one (8.3) that is computationally efficient, provided that one of the two sets is polyhedral and has a small number of facets. Note that the asymmetric relaxation generalizes Lemma 5.1, with $\boldsymbol{c}_j^\top \boldsymbol{y} + d_j \geq 0$ taking the place of the linear inequality (5.5).

The asymmetric relaxation (8.3) can be efficiently described and implemented on a computer when the set $\mathcal{X}$ is described in conic form. In this case, the homogenization $\tilde{\mathcal{X}}$ can be computed explicitly using the formula (2.4). However, the relaxation (8.3) is also usable when an explicit description of the set $\mathcal{X}$ is not available. For example, the set $\mathcal{X}$ may be very complex and accessible only through an oracle that, given a point $\boldsymbol{x}$, either certifies that $\boldsymbol{x} \in \mathcal{X}$ or returns a separating hyperplane. In fact, if $\mathcal{X}$ is bounded, such an oracle is easily adapted to checking membership to the homogenization $\tilde{\mathcal{X}}$, and this black-box access to the problem constraints is sufficient for efficient convex-optimization algorithms like the ellipsoid method [86, Chapters 3 and 4]. Since the asymmetric relaxation (8.3) works directly with its abstract set representation of $\mathcal{X}$, we call it **set based**.

8.2 Tightness of the relaxation

Ideally, we would like our convex relaxation $\mathcal{C}$ to be as tight as possible and coincide with the convex hull of the nonconvex set $\mathcal{S}$. A simple case in which this is true is when the set $\mathcal{Y}$ is an interval on the real line (see Proposition 8.2 below). However, the next proposition shows that the equality $\mathcal{C} = \mathrm{conv}(\mathcal{S})$ does not hold in general.

Proposition 8.1. *There exists sets $\mathcal{X}$ and $\mathcal{Y}$ such that the convex relaxation $\mathcal{C}$ from* (8.3) *is strictly larger than the convex hull of the set $\mathcal{S}$ from* (8.1).

Proof. Consider $\mathcal{X} := \mathcal{Y} := [-1, 1]^2$. With this choice, the constraints defining the

set $\mathcal{C}$ in (8.3) read

$$\begin{aligned}
&\boldsymbol{x} \in [-1,1]^2, \\
&(\boldsymbol{z}_i + \boldsymbol{x}, y_i + 1) \in \tilde{\mathcal{X}}, && i = 1,2, \\
&(\boldsymbol{x} - \boldsymbol{z}_i, 1 - y_i) \in \tilde{\mathcal{X}}, && i = 1,2,
\end{aligned}$$

where $\boldsymbol{z}_i$ is the ith column of $\boldsymbol{Z}$ and y_i is the ith entry of $\boldsymbol{y}$. These constraints are satisfied if we let

$$\boldsymbol{x} = \boldsymbol{y} = \boldsymbol{0}, \quad \boldsymbol{Z} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}.$$

However, this point $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z})$ does not belong to the convex hull of $\mathcal{S}$. If it was, the matrix $\boldsymbol{Z}$ should have been a convex combination of rank-one matrices in the set $[-1,1]^{2\times 2}$, but $\boldsymbol{Z}$ is an extreme point of this set and has rank equal to two. □

It is important to highlight that it was fully expected that our relaxation $\mathcal{C}$ is not always equal to the convex hull of $\mathcal{S}$. In fact, the bilinear program

$$\begin{aligned}
\text{minimize} \quad & \boldsymbol{p}^\top \boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{y} + \boldsymbol{x}^\top \boldsymbol{R} \boldsymbol{y} && \text{(8.4a)} \\
\text{subject to} \quad & \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} \in \mathcal{Y} && \text{(8.4b)}
\end{aligned}$$

is equivalent to minimizing a linear function over the nonconvex set $\mathcal{S}$, and this problem is NP-hard (see, e.g., [158, Section 1]). Therefore, the equality $\mathcal{C} = \operatorname{conv}(\mathcal{S})$ would give us a compact LP formulation of this problem, and allow us to solve an NP-hard problem in polynomial time.

The next lemma shows that the convex relaxation $\mathcal{C}$ is perfectly tight to the nonconvex set $\mathcal{S}$ at all the extreme points of the polyhedron $\mathcal{Y}$. We will use this lemma later in the chapter to give a simpler proof of the correctness of our MICP 5.8.

Lemma 8.1. *Let $\mathcal{Y}$ be a polytope and $\hat{\boldsymbol{y}}$ one of its extreme points. A point $(\boldsymbol{x}, \hat{\boldsymbol{y}}, \boldsymbol{Z})$ belongs to the set $\mathcal{S}$ in* (8.1) *if and only if it belongs to the set $\mathcal{C}$ in* (8.3)*.*

Proof. One direction follows from $\mathcal{S} \subseteq \mathcal{C}$. For the other direction we show that if $(\boldsymbol{x}, \hat{\boldsymbol{y}}, \boldsymbol{Z}) \in \mathcal{C}$ then $\boldsymbol{Z} = \boldsymbol{x}\hat{\boldsymbol{y}}^\top$. Since $\hat{\boldsymbol{y}}$ is an extreme point of $\mathcal{Y}$, there are m linearly independent inequalities that are active at $\hat{\boldsymbol{y}}$. Let $\boldsymbol{C} \in \mathbb{R}^{m\times m}$ and $\boldsymbol{d} \in \mathbb{R}^m$ collect the coefficients $(\boldsymbol{c}_i, d_i)$ of these inequalities, so that $\boldsymbol{C}\hat{\boldsymbol{y}} + \boldsymbol{d} = \boldsymbol{0}$. For the same inequalities, the constraints in (8.3) give us $\boldsymbol{Z}\boldsymbol{c}_i + d_i\boldsymbol{x} = \boldsymbol{0}$ or, equivalently, $\boldsymbol{Z}\boldsymbol{C}^\top + \boldsymbol{x}\boldsymbol{d}^\top = \boldsymbol{0}$. We then have $\boldsymbol{Z}\boldsymbol{C}^\top = \boldsymbol{x}\hat{\boldsymbol{y}}^\top\boldsymbol{C}^\top$ and, since $\boldsymbol{C}$ is invertible, $\boldsymbol{Z} = \boldsymbol{x}\hat{\boldsymbol{y}}^\top$. □

8.3 Explicit description of the convex hull

The next lemma shows that if the polyhedron $\mathcal{Y}$ is bounded (i.e., is a polytope) and described through its extreme points, then the convex hull of $\mathcal{S}$ can be easily described in a convex fashion.

Lemma 8.2. *Assume that $\mathcal{Y}$ is a polytope with extreme points $\{\hat{\boldsymbol{y}}_k\}_{k\in\mathcal{K}}$. We have*

$$\operatorname{conv}(\mathcal{S}) = \left\{ \sum_{k\in\mathcal{K}} (\boldsymbol{x}_k, \lambda_k \hat{\boldsymbol{y}}_k, \boldsymbol{x}_k \hat{\boldsymbol{y}}_k^\top) : \sum_{k\in\mathcal{K}} \lambda_k = 1,\ (\boldsymbol{x}_k, \lambda_k) \in \tilde{\mathcal{X}} \text{ for all } k \in \mathcal{K} \right\}. \quad (8.5)$$

Proof. For any $k \in \mathcal{K}$, the set $\mathcal{C}_k := \{(\boldsymbol{x}, \hat{\boldsymbol{y}}_k, \boldsymbol{x}\hat{\boldsymbol{y}}_k^\top) : \boldsymbol{x} \in \mathcal{X}\}$ is convex and contained in $\mathcal{S}$. Its homogenization is equal to $\tilde{\mathcal{C}}_k = \{(\boldsymbol{x}, \lambda\hat{\boldsymbol{y}}_k, \boldsymbol{x}\hat{\boldsymbol{y}}_k^\top, \lambda) : (\boldsymbol{x}, \lambda) \in \tilde{\mathcal{X}}\}$. We use Lemma 2.1 to compute the convex hull of the union of the sets $\mathcal{C}_k$ and, after a few manipulations, we obtain the set on the right-hand side of (8.5). Since this set is the convex hull of subsets of $\mathcal{S}$, it is contained in the convex hull of $\mathcal{S}$. On the other hand this set is convex, therefore to prove that it is equal to the convex hull of $\mathcal{S}$ we are left to show that $\mathcal{S}$ is contained in it.

Let $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z})$ be a point in $\mathcal{S}$. We have $\boldsymbol{y} = \sum_{k\in\mathcal{K}} \lambda_k \hat{\boldsymbol{y}}_k$ for some coefficients $\lambda_k \geq 0$ such that $\sum_{k\in\mathcal{K}} \lambda_k = 1$. We define $\boldsymbol{x}_k := \lambda_k \boldsymbol{x}$ for all $k \in \mathcal{K}$. This gives us

$$\begin{aligned}
&\sum_{k\in\mathcal{K}} \boldsymbol{x}_k = \boldsymbol{x}, \\
&\sum_{k\in\mathcal{K}} \boldsymbol{x}_k \hat{\boldsymbol{y}}_k^\top = \boldsymbol{x} \sum_{k\in\mathcal{K}} \lambda_k \hat{\boldsymbol{y}}_k^\top = \boldsymbol{x}\boldsymbol{y}^\top = \boldsymbol{Z}, \\
&(\boldsymbol{x}_k, \lambda_k) \in \tilde{\mathcal{X}}, \qquad\qquad \forall k \in \mathcal{K}.
\end{aligned}$$

We conclude that $(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z})$ belongs to the set on the right-hand side of (8.5), and that $\mathcal{S}$ is contained in this set. □

This proposition gives us a lifted (i.e., higher-dimensional) description of the convex hull of $\mathcal{S}$ that is convex and also set based (according to our informal definition above). While our relaxation $\mathcal{C}$ has size proportional to the number $|\mathcal{J}|$ of facets of $\mathcal{Y}$, this description of the convex hull has size proportional to the number $|\mathcal{K}|$ of extreme points of $\mathcal{Y}$. For most of the polytopes $\mathcal{Y}$ that we encounter in practice, the number of facets $|\mathcal{J}|$ is significantly smaller than the number of extreme points $|\mathcal{K}|$, and our relaxation is more effective overall. However, for polytopes $\mathcal{Y}$ with low number of extreme points, also the explicit description of the convex hull of $\mathcal{S}$ can be computationally efficient.

The next proposition shows that in the simple case where the polytope $\mathcal{Y}$ is a one-dimensional interval, then our relaxation $\mathcal{C}$ is actually equal to the convex hull of $\mathcal{S}$.

Proposition 8.2. *Let $\mathcal{Y}$ be an interval of the form $[a, b] \subset \mathbb{R}$, with $b > a$. Then the set $\mathcal{C}$ from* (8.3) *is equal to the convex hull of the set $\mathcal{S}$ in* (8.5)*.*

Proof. With $\mathcal{Y} := [a, b]$ the constraint defining $\mathcal{C}$ read

$$\mathcal{C} = \{(\boldsymbol{x}, y, \boldsymbol{z}) : \boldsymbol{x} \in \mathcal{X},\ (\boldsymbol{z} - a\boldsymbol{x}, y - a) \in \tilde{\mathcal{X}},\ (b\boldsymbol{x} - \boldsymbol{z}, b - y) \in \tilde{\mathcal{X}}\}.$$

While the constraints that define the convex hull of $\mathcal{S}$ are

$$\begin{aligned}\operatorname{conv}(\mathcal{S}) = \{(\boldsymbol{x}_0 + \boldsymbol{x}_1, \lambda_0 a + \lambda_1 b, \boldsymbol{x}_0 a + \boldsymbol{x}_1 b) :& \\ \lambda_0 + \lambda_1 = 1,\ (\boldsymbol{x}_0, \lambda_0) \in \tilde{\mathcal{X}},\ (\boldsymbol{x}_1, \lambda_1) \in \tilde{\mathcal{X}}\}.&\end{aligned}$$

The mutual inclusion of these two sets can be verified by using the change of variables

$$\begin{aligned}(\boldsymbol{x}, y, \boldsymbol{z}) &= (\boldsymbol{x}_0 + \boldsymbol{x}_1, \lambda_0 a + \lambda_1 b, \boldsymbol{x}_0 a + \boldsymbol{x}_1 b), \\ (\boldsymbol{x}_0, \boldsymbol{x}_1, \lambda_0, \lambda_1) &= \frac{1}{b-a}(b\boldsymbol{x} - \boldsymbol{z}, \boldsymbol{z} - a\boldsymbol{x}, b - y, y - a).\end{aligned}$$

□

8.4 Related relaxation techniques

The basic idea of generating new valid constraints by multiplying existing ones is classical, and has many incarnations: from the simple McCormick envelope [134] to semidefinite hierarchies for polynomial optimization [148, 149, 110], passing through the Reformulation-Linearization Technique (RLT) [169]. Among this family of techniques, the Lovász-Schrijver hierarchy [120] is the closest to ours, since it is set based and includes constraints of the form (8.3) (see Theorem 1.6 and Conditions (iii) through (iii") in [120]). However, this hierarchy focuses on binary optimization and symmetric quadratic maps, and its naive application to the bilinear set $\mathcal{S}$ would produce multiple redundant variables and constraints. Our approach leverages the bilinear structure of the set $\mathcal{S}$ to construct a relaxation $\mathcal{C}$ that is smaller and as tight as the first level of the Lovász-Schrijver hierarchy, without semidefinite constraints. Our practical experience is that higher levels of the hierarchy and semidefinite constraints lead to MICPs that, although stronger, are significantly slower to solve.

Consider the case in which both the sets $\mathcal{X}$ and $\mathcal{Y}$ are polyhedral: $\mathcal{X} := \{\boldsymbol{x} : \boldsymbol{a}_i^\top \boldsymbol{x} + b_i \geq 0 \text{ for all } i \in \mathcal{I}\}$ and $\mathcal{Y} := \{\boldsymbol{y} : \boldsymbol{c}_j^\top \boldsymbol{y} + d_j \geq 0 \text{ for all } j \in \mathcal{J}\}$. After a few manipulations, our relaxation simplifies to

$$\mathcal{C} = \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z}) : \boldsymbol{x} \in \mathcal{X},\ \boldsymbol{y} \in \mathcal{Y},\\ \boldsymbol{a}_i^\top \boldsymbol{Z} \boldsymbol{c}_j + d_j \boldsymbol{a}_i^\top \boldsymbol{x} + b_i \boldsymbol{c}_j^\top \boldsymbol{y} + b_i d_j \geq 0 \text{ for all } i \in \mathcal{I} \text{ and } j \in \mathcal{J}\}.$$

Therefore, in this simple case, our approach simply amounts to multiplying the inequalities that define the polyhedra $\mathcal{X}$ and $\mathcal{Y}$. This specific relaxation was originally proposed in [170], as a variant of the RLT. The fact that our relaxation is not the convex hull of $\mathcal{S}$ is therefore well known. In the even simpler case where $\mathcal{X}$ and $\mathcal{Y}$ are intervals on the real line, our relaxation simplifies to the McCormick envelope [134]. Proposition 8.2 generalizes the result that the McCormick envelop is the convex hull of the nonconvex set $\mathcal{S}$ when $\mathcal{X}$ and $\mathcal{Y}$ are one-dimensional intervals.

Consider the bilinear program (8.4) with polytopic sets $\mathcal{X}$ and $\mathcal{Y}$, and the additional constraint $\boldsymbol{y} \in \{0,1\}^m$. Assuming $\mathcal{Y} \subseteq [0,1]^m$, the first-level RLT is known to yield a valid MILP formulation of this program [1, Theorem 1]. Lemma 8.1 extends this result to generic closed convex sets $\mathcal{X}$. In fact, $\mathcal{Y} \subseteq [0,1]^m$ ensures that any vector $\boldsymbol{y} \in \mathcal{Y} \cap \{0,1\}^m$ is an extreme point of $\mathcal{Y}$, and the relaxation $\mathcal{C}$ is exact in correspondence of these points.

The recent work [196] shows how perspective functions can be used to allow the multiplication of nonlinear convex constraints in the RLT algorithm. However, the relaxation in that work is not set based, and requires an explicit analysis of all the possible products of basic cone inequalities.

8.5 Back to graphs of convex sets

As already noticed, Theorem 8.1 generalizes Lemma 5.1, which is the backbone of our MICP formulations of the GCS problems. Specifically, the following is an equivalent way of deriving our MICPs. For each vertex $v \in \mathcal{V}$ in our GCS, we stack in a vector $\boldsymbol{y}_v \in \{0,1\}^{\{v\} \cup \mathcal{I}_v}$ the binary variables y_v and y_e for $e \in \mathcal{I}_v$. We define a polytope $\mathcal{Y}_v \subset [0,1]^{\{v\} \cup \mathcal{I}_v}$ by collecting all the linear inequalities (5.5) that only affect the entries of $\boldsymbol{y}_v$. Then we define a nonconvex set

$$\mathcal{S}_v := \{(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z}) : \boldsymbol{x} \in \mathcal{X}_v,\ \boldsymbol{y} \in \mathcal{Y}_v,\ \boldsymbol{Z} = \boldsymbol{x}\boldsymbol{y}^\top\}.$$

for each vertex $v \in \mathcal{V}$. Letting $\boldsymbol{Z}_v$ be the matrix that stacks the auxiliary variables $\boldsymbol{z}_v$ and $\boldsymbol{z}_v^e$ for $e \in \mathcal{I}_v$, we can replace all the bilinear constraints in the MINCP (5.4) with constraints of the form $(\boldsymbol{x}_v, \boldsymbol{y}_v, \boldsymbol{Z}_v) \in \mathcal{S}_v$. Finally, we apply Theorem 8.1 to get a convex relaxation $\mathcal{C}_v$ of each set $\mathcal{S}_v$, and we get an MICP by replacing $\mathcal{S}_v$ with $\mathcal{C}_v$.

Theorem 5.1 established the validity of the MICP obtained using Lemma 5.1. Using the results in this section, the proof of this theorem much simpler. Constraint (5.8b) forces the vectors $\boldsymbol{y}_v$ to have binary entries and, consequently, to be extreme points of the corresponding polytopes $\mathcal{Y}_v$. The MICP (5.8) is valid since by Lemma 8.1 the sets $\mathcal{C}_v$ and $\mathcal{S}_v$ are equal at the extreme points of $\mathcal{Y}_v$.

Finally, observe that we applied Theorem 8.1 to each vertex independently. An alternative (more straightforward) way of reformulating the MINCP (5.4) as an MICP is to stack all the continuous and binary variables into two large vectors $\boldsymbol{x} \in \mathbb{R}^{\sum_{v\in\mathcal{V}} n_v}$ and $\boldsymbol{y} \in \{0,1\}^{\mathcal{V}\cup\mathcal{E}}$, respectively. Then the MINCP (5.4) can be restated as a large program of the form

$$\begin{aligned} \text{minimize} \quad & f(\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z}) \\ \text{subject to} \quad & \boldsymbol{y} \in \{0,1\}^{\mathcal{V}\cup\mathcal{E}}, \\ & (\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{Z}) \in \mathcal{S}, \end{aligned}$$

where the nonconvex set $\mathcal{S}$ is defined as in (8.1). By applying Theorem 8.1 to the whole set $\mathcal{S}$, we now obtain a second MICP formulation of the GCS problem. While correct, in our experience, this MICP is too large and slow to solve for most of the problems we encounter in practice. Conversely, our method operates on each vertex independently, and leads to slightly weaker but significantly smaller MICPs that are solved much more quickly.

Part III

Shortest-path problem and its applications

Chapter 9

Shortest-path problem

In the last part of this thesis we focus on the directed Shortest-Path Problem (SPP) in Graphs of Convex Sets (GCS). This problem is particularly important since it generalizes many fundamental problems in optimal control, multi-stage decision making, and robotics. In addition, the structure of this problem is particularly well suited to the optimization techniques that we introduced in this thesis.

In this chapter we dive deep in the analysis of the SPP in GCS and its numerical solution. Applications in optimal control and robotics will be discussed in the following chapters. Most of the material presented in this chapter is taken from [131].

9.1 Problem statement

The ordinary directed SPP and its extension in GCS have been discussed in Sections 4.4.1 and 6.1, respectively. We start this chapter by briefly recapping the statements of these problems and some important definitions.

In the directed SPP, we are given a directed graph $G = (\mathcal{V}, \mathcal{E})$ with vertex costs $c_v \in \mathbb{R}$ for $v \in \mathcal{V}$ and edge costs $c_e \in \mathbb{R}$ for $e \in \mathcal{E}$. We seek a path in G of minimum cost from a specified source vertex $s \in \mathcal{V}$ to a specified target vertex $t \in \mathcal{V}$. Recall that a path in G is a subgraph $(\mathcal{W}, \mathcal{F}) \subseteq G$ with

$$\mathcal{W} = \{s = v_0, v_1, \dots, v_{l-1}, v_l = t\}, \qquad \mathcal{E} = \{(v_0, v_1), (v_1, v_2), \dots, (v_{l-1}, v_l)\},$$

and $v_i \neq v_j$ for all $i, j = 1, \dots, l$. Let $\mathcal{P}_{s,t}$ be the set of all s-t paths in our graph G.

We formulate the ordinary SPP as the optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in \mathcal{W}} c_v + \sum_{e \in \mathcal{F}} c_e && \text{(9.1a)} \\ \text{subject to} \quad & (\mathcal{W}, \mathcal{F}) \in \mathcal{P}_{s,t}. && \text{(9.1b)} \end{aligned}$$

In the SPP in GCS each vertex $v \in \mathcal{V}$ is paired with a continuous variable $\boldsymbol{x}_v \in \mathbb{R}^{n_v}$. The scalar costs in (9.1a) are replaced by the convex functions $f_v : \mathbb{R}^{n_v} \to \mathbb{R}$ for all $v \in \mathcal{V}$ and $f_e : \mathbb{R}^{n_v+n_w} \to \mathbb{R}$ for all $e = (v, w) \in \mathcal{E}$. The selection of a path decides both the cost functions that we need to minimize and the constraints $\mathcal{X}_v \subseteq \mathbb{R}^{n_v}$ for $v \in \mathcal{V}$ and $\mathcal{X}_e \subseteq \mathbb{R}^{n_v+n_w}$ for $e = (v, w) \in \mathcal{E}$ that we need to satisfy. This leads to the optimization problem

$$\begin{aligned} \text{minimize} \quad & \sum_{v \in \mathcal{W}} f_v(\boldsymbol{x}_v) + \sum_{e=(v,w) \in \mathcal{F}} f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) && && \text{(9.2a)} \\ \text{subject to} \quad & (\mathcal{W}, \mathcal{F}) \in \mathcal{P}_{s,t}, && && \text{(9.2b)} \\ & \boldsymbol{x}_v \in \mathcal{X}_v, && \forall v \in \mathcal{W}, && \text{(9.2c)} \\ & (\boldsymbol{x}_v, \boldsymbol{x}_w) \in \mathcal{X}_e, && \forall e = (v, w) \in \mathcal{F}. && \text{(9.2d)} \end{aligned}$$

9.2 Complexity analysis

The ordinary SPP (9.1) is NP-hard if we do not make specific assumptions on the structure of the graph or the signs of the vertex and edge costs. In fact, the **Hamiltonian-Path Problem** (HPP), which asks whether a graph G contains an s-t Hamiltonian path, is well known to be NP-complete for a cyclic graph [104], and reduces to the SPP if we let $c_v = -1$ for all $v \in \mathcal{V}$ and $c_e = 0$ for all $e \in \mathcal{E}$. In order for the SPP to be efficiently solvable, we need to rule out the presence of cycles with negative cost (see, e.g., [167, Chapter 8]). There are two obvious, but practically very relevant, scenarios in which negative-cost cycles are certainly absent:

- The graph G is acyclic. In this case a shortest path can be found in a time that grows linearly with $|\mathcal{V}| + |\mathcal{E}|$ (see, e.g., [43, Section 22.2]).
- All the costs c_v for $v \in \mathcal{V}$ and c_e for $e \in \mathcal{E}$ are nonnegative. In this case we can use the variant of Dijkstra's method [52] proposed in [69] to find a shortest path in a time proportional to $|\mathcal{V}| \log(|\mathcal{V}|) + |\mathcal{E}|$. Alternatively, we can solve the problem as an LP (see the discussion at the end of Section 4.4.1).

The SPP in GCS is obviously NP-hard when no special assumptions are made on

the graph structure or the signs of the cost functions f_v and f_e, since it generalizes the ordinary SPP. Given the observations above, two practically relevant special cases of the SPP in GCS to consider, with the hope that they are efficiently solvable, are:

- The SPP in GCS with acyclic graph G.
- The SPP in GCS with nonnegative costs, i.e., $f_v : \mathbb{R}^{n_v} \to \mathbb{R}_{\geq 0}$ for all $v \in \mathcal{V}$ and $f_e : \mathbb{R}^{n_v+n_w} \to \mathbb{R}_{\geq 0}$ for all $e = (v, w) \in \mathcal{E}$.

The next theorem shows that, in contrast to the ordinary SPP, the SPP in GCS is still NP-hard under these assumptions.

Theorem 9.1. *Assume that the graph G is acyclic. Assume that the vertex costs f_v and the edge costs f_e are nonnegative for all $v \in \mathcal{V}$ and $e \in \mathcal{E}$. The SPP in GCS* (9.2) *is NP-hard.*

The proof of this theorem is a reduction of the **3SAT** problem (Boolean-satisfiability problem with clauses of three literals), which is another well-known NP-complete problem [104]. Let us quickly recall the statement of this problem (without introducing the jargon typically used for satisfiability problems). For each $i = 1, \ldots, m$ and $j = 1, 2, 3$, we are given a convex set

$$\mathcal{C}_{i,j} := \{\boldsymbol{x} \in [0, 1]^n : x_{k_{i,j}} = a_{i,j}\},$$

where $a_{i,j} \in \{0, 1\}$ and $k_{i,j} \in \{1, \ldots, n\}$. In words, each set $\mathcal{C}_{i,j}$ allows the vector $\boldsymbol{x}$ to have entries between zero and one, and forces the entry of $\boldsymbol{x}$ indexed by $k_{i,j}$ to be equal to $a_{i,j}$ (i.e., either zero or one). The 3SAT problem asks whether the set

$$\bigcap_{i=1}^{m} (\mathcal{C}_{i,1} \cup \mathcal{C}_{i,2} \cup \mathcal{C}_{i,3})$$

is empty or not.

Remark 9.1. More commonly, the 3SAT problem asks for a vector $\boldsymbol{x}$ with only binary entries, i.e., the sets $\mathcal{C}_{i,j}$ should be defined as $\{\boldsymbol{x} \in \{0, 1\}^n : x_{k_{i,j}} = a_{i,j}\}$. This problem is easily seen to be equivalent to the one above: any feasible solution $\boldsymbol{x}$ of this second problem is feasible for the problem above, and by rounding the fractional entries of any feasible solution $\boldsymbol{x}$ of the problem above we obtain a feasible solution for this second problem.

Proof of Theorem 9.1. We reduce the 3SAT problem to the SPP in GCS. Given the sets $\mathcal{C}_{i,j}$, we construct the layered GCS illustrated in Figure 9-1. The leftmost and

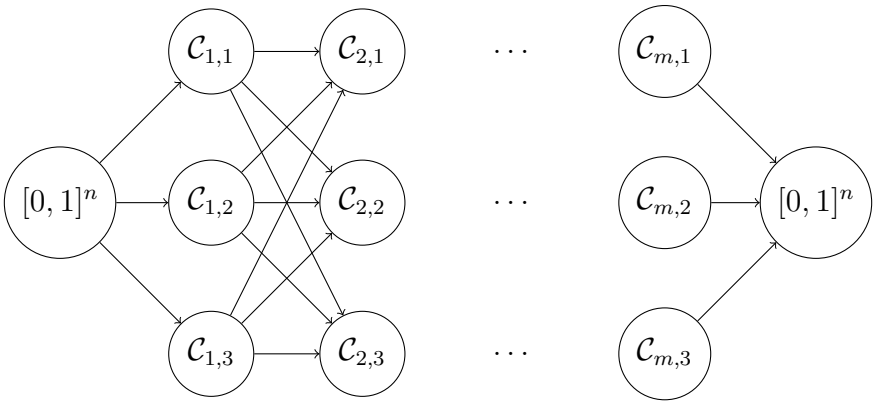

Figure 9-1: Reduction of the 3SAT problem to the SPP in GCS.

rightmost vertices are the source s and the target t, respectively. We pair these vertices with the convex sets $\mathcal{X}_s := \mathcal{X}_t := [0,1]^n$. Between the source and the target, we have m layers with three vertices each. The jth vertex in the ith layer, denoted as v, is paired with the convex set $\mathcal{X}_v := \mathcal{C}_{i,j}$. The directed edges connect the source to all the vertices in the first layer, and all the vertices in the last layer to the target. The layers in between are fully connected. We define the edge constraints as $\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w) : \boldsymbol{x}_v = \boldsymbol{x}_w\}$ for all $e = (v, w) \in \mathcal{E}$. All the vertex and edge costs are equal to zero.

The edge constraints force all the continuous variables $\boldsymbol{x}_v$ along the selected path (i.e., for all $v \in \mathcal{W}$) to be equal. We call this common value $\boldsymbol{x}$. Thanks to the layered structure of our graph, the vector $\boldsymbol{x}$ lies in at least one of the sets $\mathcal{C}_{i,1}$, $\mathcal{C}_{i,2}$, or $\mathcal{C}_{i,3}$ for all $i = 1, \ldots, m$. Therefore the SPP in GCS we have constructed is feasible if and only if the 3SAT problem is. □

The proof of Theorem 9.1 shows that even deciding the feasibility of an SPP in GCS is NP-hard. A simple modification of the same proof shows that solving a feasible SPP in GCS is equally hard. Specifically, if we let $\mathcal{X}_e := \mathbb{R}^n$ and $f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \|\boldsymbol{x}_v - \boldsymbol{x}_w\|$ for all $e = (v, w) \in \mathcal{E}$, then the SPP in GCS is feasible (we can let $\boldsymbol{x}_v$ be a any point in $\mathcal{X}_v$ for all $v \in \mathcal{V}$ and take any path through the graph). In addition, the SPP in GCS has optimal value equal to zero if and only if the initial 3SAT problem is feasible.

9.2.1 Alternative proofs of NP-hardness

The proof of Theorem 9.1 makes minimal assumptions on the structure of the GCS, but requires the sets $\mathcal{X}_v$ to be high dimensional. We include here two alternative proofs of NP-hardness of the SPP in GCS that use low-dimensional sets $\mathcal{X}_v$. The first

uses intervals on the real line, the second disjoint rectangles in two dimensions. On the other hand, these alternative proofs rely on the graph G having cycles (and the edge costs f_e growing superlinearly).

Theorem 9.2. *Assume that $\mathcal{X}_v \subset \mathbb{R}$ for all $v \in \mathcal{V}$. The SPP in GCS* (9.2) *is NP-hard.*

Proof. We reduce the HPP, which is NP-complete [104], to the SPP in GCS. We construct an SPP in GCS that shares the same graph G as the given HPP. We let the source $\mathcal{X}_s := \{0\}$ and target $\mathcal{X}_t := \{1\}$ sets be singletons on the real line, and we define $\mathcal{X}_v := [0, 1]$ for all $v \neq s, t$. The length of each edge $e = (v, w)$ is the Euclidean distance squared, $f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \|\boldsymbol{x}_w - \boldsymbol{x}_v\|_2^2$. Given these choices, the optimal positioning of the vertices for a fixed path with vertices $s = v_0, v_1, \ldots, v_l = t$ is $\boldsymbol{x}_{v_k} = k/l$ for $k = 0, \ldots, l$. The cost of this solution is $l(1/l)^2 = 1/l$. We conclude that an optimal path is one for which l is maximized, and is Hamiltonian if and only if G contains a Hamiltonian path. □

Theorem 9.3. *Assume that the sets $\mathcal{X}_v$ for $v \in \mathcal{V}$ are disjoint rectangles in two dimensions. The SPP in GCS* (9.2) *is NP-hard.*

Proof. We repeat the proof of Theorem 9.2, but we embed the sets $\mathcal{X}_v$ defined above in a two-dimensional space. Specifically, we define very small scalars ε_v for all $v \in \mathcal{V}$. We assume that $\varepsilon_v \neq \varepsilon_w$ for all $v, w \in \mathcal{V}$ such that $v \neq w$. Then we let $\mathcal{X}_s := \{(0, \varepsilon_s)\}$, $\mathcal{X}_t := \{(1, \varepsilon_t)\}$, and $\mathcal{X}_v := [0, 1] \times \{\varepsilon_v\}$ for all $v \neq s, t$. This modification ensures that the sets $\mathcal{X}_v$ are disjoint, but does not affect the optimal path, which is still Hamiltonian if and only if G has a Hamiltonian path. □

9.3 Mixed-integer convex formulation

In Section 6.1 we have shown how the SPP in GCS is formulated as a Mixed-Integer Convex Program (MICP). Here we consider special classes of SPPs in GCS, and we show how the MICP from Section 6.1 can be made computationally lighter.

9.3.1 Nonnegative costs

We start by assuming that the vertex costs f_v and the edge costs f_e are nonnegative functions. Under this assumption any cycle in the GCS has nonnegative cost and, therefore, the cycle-elimination constraints are not necessary for the validity of our MICP (although in some cases they can lead to a tighter convex relaxation). Specifically, the validity of our MICP is intact if we remove the constraint (4.4b), as well as

the corresponding constraints obtained by using Lemma 5.1, namely (6.2) and (6.3). Collecting all the other constraints, our MICP reads

$$\begin{aligned}
\text{minimize} \quad & \sum_{v\in\mathcal{V}} \tilde{f}_v(\boldsymbol{z}_v, y_v) + \sum_{e=(v,w)\in\mathcal{E}} \tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) && && \text{(9.3a)}\\
\text{subject to} \quad & y_e \in \{0,1\}, && \forall e \in \mathcal{E}, && \text{(9.3b)}\\
& y_v \in \{0,1\}, && \forall v \in \mathcal{V}, && \text{(9.3c)}\\
& (\boldsymbol{z}_v, y_v) = \sum_{e\in\mathcal{I}_v^{\text{in}}} (\boldsymbol{z}_v^e, y_e) + \delta_{sv}(\boldsymbol{x}_v, 1), && \forall v \in \mathcal{V}, && \text{(9.3d)}\\
& (\boldsymbol{z}_v, y_v) = \sum_{e\in\mathcal{I}_v^{\text{out}}} (\boldsymbol{z}_v^e, y_e) + \delta_{tv}(\boldsymbol{x}_v, 1), && \forall v \in \mathcal{V}, && \text{(9.3e)}\\
& (\boldsymbol{z}_v, y_v) \in \tilde{\mathcal{X}}_v,\ (\boldsymbol{x}_v - \boldsymbol{z}_v, 1 - y_v) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V}, && \text{(9.3f)}\\
& (\boldsymbol{z}_v^e, y_e) \in \tilde{\mathcal{X}}_v,\ (\boldsymbol{x}_v - \boldsymbol{z}_v^e, 1 - y_e) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v, && \text{(9.3g)}\\
& (\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) \in \tilde{\mathcal{X}}_e, && \forall e = (v,w) \in \mathcal{E}. && \text{(9.3h)}
\end{aligned}$$

The convex relaxation of this problem is obtained simply by dropping the constraints (9.3b) and (9.3c).

It can be verified that multiple components (variables and constraints) of problem (9.3) are redundant, in the sense that they can be removed from the problem without affecting the validity of the MICP and the optimal value of its convex relaxation. This reduced-size MICP reads as follows:

$$\begin{aligned}
\text{minimize} \quad & \sum_{v\in\mathcal{V}} \tilde{f}_v(\boldsymbol{z}_v, y_v) + \sum_{e=(v,w)\in\mathcal{E}} \tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) && && \text{(9.4a)}\\
\text{subject to} \quad & y_e \in \{0,1\}, && \forall e \in \mathcal{E}, && \text{(9.4b)}\\
& y_v \in \{0,1\}, && \forall v \in \mathcal{V}, && \text{(9.4c)}\\
& y_v = \sum_{e\in\mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e\in\mathcal{I}_v^{\text{out}}} y_e + \delta_{tv} \le 1, && \forall v \in \mathcal{V}, && \text{(9.4d)}\\
& \boldsymbol{z}_v = \sum_{e\in\mathcal{I}_v^{\text{in}}} \boldsymbol{z}_v^e + \delta_{sv}\boldsymbol{z}_v = \sum_{e\in\mathcal{I}_v^{\text{out}}} \boldsymbol{z}_v^e + \delta_{tv}\boldsymbol{z}_v, && \forall v \in \mathcal{V}, && \text{(9.4e)}\\
& (\boldsymbol{z}_v^e, y_e) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v, && \text{(9.4f)}\\
& (\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) \in \tilde{\mathcal{X}}_e, && \forall e = (v,w) \in \mathcal{E}. && \text{(9.4g)}
\end{aligned}$$

Specifically, we have removed:

- the variables $\boldsymbol{x}_v$ for all $v \in \mathcal{V}$,
- the constraint $(\boldsymbol{z}_v, y_v) \in \tilde{\mathcal{X}}_v$ from (9.3f) for all $v \in \mathcal{V}$,

- the constraint $\boldsymbol{x}_v - \boldsymbol{z}_v \in (1 - y_v)\mathcal{X}_v$ from (9.3f) for all $v \in \mathcal{V}$,
- the constraint $y_e \leq 1$ from (9.3g) for all $e \in \mathcal{E}$,
- the constraint $\boldsymbol{x}_v - \boldsymbol{z}_v^e \in (1 - y_e)\mathcal{X}_v$ from (9.3g) for all $v \in \mathcal{V}$ and $e \in \mathcal{I}_v$.

Again the convex relaxation of this problem is obtained by dropping the constraints (9.4b) and (9.4c).

Given a feasible solution for the convex relaxation of one of the two MICPs above it is always possible to reconstruct a feasible solution with equal cost for the other MICP. From the larger MICP to the reduced MICP it suffices to discard the variables $\boldsymbol{x}_v$ for all $v \in \mathcal{V}$. For the other direction, we can let $\boldsymbol{x}_v$ be any point that satisfies the constraint in the third bullet above. It is easily verified that this value of $\boldsymbol{x}_v$ satisfies all the other constraints that we have removed.

9.3.2 No costs on the vertices

The MICP (9.4) can be simplified further if the vertex costs f_v are zero for all $v \in \mathcal{V}$, and the edge costs f_e are still assumed to be nonnegative. In this case, we obtain the small MICP

$$\begin{aligned}
\text{minimize} \quad & \sum_{e=(v,w)\in\mathcal{E}} \tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) && && \text{(9.5a)}\\
\text{subject to} \quad & y_e \in \{0,1\}, && \forall e \in \mathcal{E}, && \text{(9.5b)}\\
& \sum_{e\in\mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e\in\mathcal{I}_v^{\text{out}}} y_e + \delta_{tv} \leq 1, && \forall v \in \mathcal{V}, && \text{(9.5c)}\\
& \sum_{e\in\mathcal{I}_v^{\text{in}}} \boldsymbol{z}_v^e = \sum_{e\in\mathcal{I}_v^{\text{out}}} \boldsymbol{z}_v^e, && \forall v \in \mathcal{V}\backslash\{s,t\}, && \text{(9.5d)}\\
& (\boldsymbol{z}_v^e, y_e) \in \tilde{\mathcal{X}}_v, && \forall v \in \mathcal{V},\ e \in \mathcal{I}_v, && \text{(9.5e)}\\
& (\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) \in \tilde{\mathcal{X}}_e, && \forall e = (v,w) \in \mathcal{E}. && \text{(9.5f)}
\end{aligned}$$

Here we have removed:

- the variables y_v and $\boldsymbol{z}_v$ for all $v \in \mathcal{V}$,
- the constraint (9.4e) for $v = s$ and $v = t$.

Given a feasible solution for the convex relaxation (9.5) or (9.4) we can always recover a feasible solution with equal cost for the other convex relaxation.

Remark 9.2. An SPP in GCS with nonzero vertex costs can always be reduced to an SPP in GCS with zero vertex costs. We can do this by distributing the cost f_v of vertex v on the edges that are incident with v. For all the vertices v that are not the target, we add the term $f_v(\boldsymbol{x}_v)$ to the costs $f_e(\boldsymbol{x}_v, \boldsymbol{x}_w)$ of all the edges $e = (v, w)$ that are outgoing v. For the target t, we add the same term to the costs of the edges that are incoming t. We also note that the MICP that we obtain after distributing the vertex costs on the edges is stronger than the original one (i.e., has a tighter convex relaxation). To see this, consider for example a vertex v that is not the target. The cost term $\tilde{f}_v(\boldsymbol{z}_v, y_v)$ in the original MICP is replaced with

$$\sum_{e \in \mathcal{I}_v^{\text{out}}} \tilde{f}_v(\boldsymbol{z}_v^e, y_e) \geq \tilde{f}_v \left(\sum_{e \in \mathcal{I}_v^{\text{out}}} \boldsymbol{z}_v^e, \sum_{e \in \mathcal{I}_v^{\text{out}}} y_e \right) = \tilde{f}_v(\boldsymbol{z}_v, y_v),$$

where the inequality is due to the convexity of $\tilde{f}_v$ as in (2.10). An analogous inequality is easily verified for the target vertex. On the other hand, the MICP with vertex costs distributed on the edges is computationally more expensive, since it has more addends in the objective function. Which MICP solves faster depends on the specific problem instance.

9.3.3 Acyclic graphs

If the graph G is acyclic then the constraint $y_v \leq 1$ is not necessary. This so-called **degree constraint** can be eliminated from (9.4d) and also from the right-hand side of (9.5c).

9.4 Dual problem

We now analyze the dual of the convex relaxation of our MICP, and we draw additional parallels between this problem and the Linear Programming (LP) formulation of the SPP discussed in Section 4.4.1. The goal of this section is to get a better understanding of the various components of our problem. With this goal in mind, we make our problem as simple as possible: we assume that the vertex costs f_v are zero and the graph G is acyclic. Therefore our primal problem is (9.5), without the degree constraint in (9.5c).

9.4.1 Dual of the SPP

In Section 4.4.1 we have shown that the SPP with nonnegative weights ca be formulated as the LP:

$$\begin{aligned}
\text{minimize} \quad & \sum_{e\in\mathcal{E}} c_e y_e && && (9.6a)\\
\text{subject to} \quad & \sum_{e\in\mathcal{I}_v^{\text{in}}} y_e + \delta_{sv} = \sum_{e\in\mathcal{I}_v^{\text{out}}} y_e + \delta_{tv}, && \forall v \in \mathcal{V}, && (9.6b)\\
& y_e \geq 0, && \forall e \in \mathcal{E}, && (9.6c)
\end{aligned}$$

where we have set the vertex costs to zero, as by our assumption, and we have removed the variables y_v. Note also that we have removed the degree constraints $y_v \leq 1$, which is easily seen to be redundant for this problem.

The dual of the LP above can be derived as in Section 2.3.2 and reads

$$\begin{aligned}
\text{maximize} \quad & p_s - p_t && && (9.7a)\\
\text{subject to} \quad & p_v - p_w \leq c_e, && \forall e = (v,w) \in \mathcal{E}. && (9.7b)
\end{aligned}$$

Here $p_v \in \mathbb{R}$ is the multiplier of the flow conservation in (9.6b) for all $v \in \mathcal{V}$. These multipliers are interpretable as potentials: the objective maximizes the potential jump between source and target, and the constraints ensure that the potential jump along each edge does not exceed the edge cost.

For the LPs (9.6) and (9.7), complementary slackness reads $(c_e - p_v + p_w)y_e = 0$ for all edges $e = (v,w)$. Therefore, at optimality, each edge $e \in \mathcal{F}$ along the shortest path must have a potential jump equal to its edge cost.

9.4.2 Dual of the SPP in GCS

The convex relaxation of the MICP (9.5) is a conic program, and its dual is also derived as shown in Section 2.3.2. Neglecting the degree constraint, the dual problem reads

$$\begin{aligned}
\text{maximize} \quad & p_s - p_t && && (9.8a)\\
\text{subject to} \quad & \boldsymbol{r}_v^\top \boldsymbol{x}_v + p_v - \boldsymbol{r}_w^\top \boldsymbol{x}_w - p_w \leq f_e(\boldsymbol{x}_v, \boldsymbol{x}_w), && && \\
& \qquad \forall (\boldsymbol{x}_v, \boldsymbol{x}_w) \in (\mathcal{X}_v \times \mathcal{X}_w) \cap \mathcal{X}_e,\ e = (v,w) \in \mathcal{E}, && && (9.8b)\\
& \boldsymbol{r}_s = \boldsymbol{r}_t = 0. && && (9.8c)
\end{aligned}$$

The dual variables are $p_v \in \mathbb{R}$ and $\boldsymbol{r}_v \in \mathbb{R}^{n_v}$ for all $v \in \mathcal{V}$. The first are paired with the flow conservation as above. The second correspond to (9.5c). The additional variables $\boldsymbol{r}_s$ and $\boldsymbol{r}_t$ have only the role of simplifying the presentation.

Similarly to the LP (9.7), the dual (9.8) can be interpreted in terms of potentials. For each vertex $v \in \mathcal{V}$, the linear function $\boldsymbol{r}_v^\top \boldsymbol{x}_v + p_v$ defines the potential of the point $\boldsymbol{x}_v \in \mathcal{X}_v$. Because of (9.8c), these functions are constant over the source and target sets, and the objective (9.8a) maximizes the potential jump between s and t, as in the ordinary SPP. Like (9.7b), constraint (9.8b) asks the potential jump along an edge to be smaller than the edge cost. By setting all the potential functions to zero, we see that the dual problem is always feasible and has nonnegative optimal value.

For the primal-dual pair (9.4) and (9.8), complementary slackness requires

$$\boldsymbol{r}_v^\top \boldsymbol{z}_v^e + p_v y_e - \boldsymbol{r}_w^\top \boldsymbol{z}_w^e - p_w y_e = \tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e)$$

for all edges $e = (v, w)$. As for the ordinary SPP, this is trivially satisfied if $y_e = 0$. While, for $y_e > 0$, we get $\boldsymbol{r}_v^\top \bar{\boldsymbol{z}}_v^e + p_v - \boldsymbol{r}_w^\top \bar{\boldsymbol{z}}_w^e - p_w = f_e(\bar{\boldsymbol{z}}_v^e, \bar{\boldsymbol{z}}_w^e)$, with $\bar{\boldsymbol{z}}_v^e := \boldsymbol{z}_v^e / y_e$ and $\bar{\boldsymbol{z}}_w^e := \boldsymbol{z}_w^e / y_e$. In words, at optimality, the potential jump along edge e is tight to the edge cost f_e at the point $(\bar{\boldsymbol{z}}_v^e, \bar{\boldsymbol{z}}_w^e)$.

9.5 Heuristic solution via rounding

In many practical cases, we do not necessarily seek an optimal solution of the SPP in GCS but, instead, we are interested in finding a feasible solution of low cost quickly. In these cases, a simple and effective way of tackling the SPP in GCS is to solve its convex relaxation and then try to obtain a feasible solution using a rounding strategy. As discussed in Section 3.3.1, rounding is not always guaranteed to work, but can be very effective for some problems with special structure. In addition, rounding automatically provides us with the bound (3.4) on the optimality gap of the solution that it finds.

The rounding strategy that we propose is based on the observation that, if we fix a path $(\mathcal{W}, \mathcal{F}) \in \mathcal{P}_{s,t}$ through the graph G, then the SPP in GCS (9.2) reduces to the following convex program (which we will call **convex restriction** [51]):

$$\begin{aligned}
\text{minimize} \quad & \sum_{v \in \mathcal{W}} f_v(\boldsymbol{x}_v) + \sum_{e=(v,w) \in \mathcal{F}} f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) && && \text{(9.9a)} \\
\text{subject to} \quad & \boldsymbol{x}_v \in \mathcal{X}_v, && \forall v \in \mathcal{W}, && \text{(9.9b)} \\
& (\boldsymbol{x}_v, \boldsymbol{x}_w) \in \mathcal{X}_e, && \forall e = (v, w) \in \mathcal{F}, && \text{(9.9c)}
\end{aligned}$$

where the only variables are $\boldsymbol{x}_v$ for $v \in \mathcal{W}$. This program is small in size and has the structure of an optimal-control problem, with sparse (banded) cost and constraints. Thanks to this, it can be solved in a time that increases only linearly with the length l of the path [189].

Our rounding strategy is a randomized method that generates a collection of candidate paths, for each path solves the corresponding convex restriction (9.9), and then simply takes the solution with lowest optimal value. To generate a collection of candidate paths we observe that the value of each variable y_e is naturally interpreted as the probability of the edge e being along the shortest path. When we solve the convex relaxation, these probabilities can be fractional, and induce a probability distribution over the set of all possible paths through the graph G. To sample from this distribution we use a simple randomized depth-first search. Starting from the source vertex $v = s$, at each iteration, this search crosses an edge $e \in \mathcal{I}_v^{\text{out}}$ with probability y_e/y_v. If a dead end occurs (i.e., if all the outgoing edges with nonzero probability lead to a vertex that we have already visited) we backtrack to the latest vertex in the path that admits a way out. The algorithm terminates when the target t is reached and a path is identified. We repeat this algorithm until we identify a given number of distinct paths (say five or ten) or we reach a given maximum number of random trials (say one hundred).

Our rounding strategy is particularly effective when the convex restriction (9.9) is guaranteed to be feasible, no matter the path that we select to cross the graph. In this case, it is easily verified that our convex relaxation of the SPP in GCS is feasible if and only there is an s-t path in the graph G, and the randomized rounding is guaranteed to identify a solution of the SPP in GCS. This will be the case for some of the motion-planning problems that we will study in Chapter 11.

Remark 9.3. Making our rounding strategy deterministic by selecting at each iteration the edge e with largest probability y_e is, in general, a bad idea. To see this, imagine that we have solved the convex relaxation, we have extracted the path with largest probability mass, and this path is in fact optimal for the SPP in GCS. Then we can construct a new graph where we add multiple copies of the vertices and edges of the optimal path. If we solve our convex relaxation with this new graph, it can be verified that the probability mass will be divided evenly between the multiple copies of the optimal path. Therefore, if we add a sufficient number of copies, a greedy deterministic search will eventually select a path that is not optimal. Conversely, repeating the same experiment with our randomized strategy, the probability of sampling an optimal path is independent of the presence of multiple optimal paths.

9.6 Numerical experiments

This section collects multiple numerical experiments. We start in Section 9.6.1 with a simple two-dimensional problem. Section 9.6.2 presents a statistical analysis of the performance of our MICP on large-scale instances of the SPP in GCS. The same large-scale instances are used in Section 9.6.3 to analyze the effectiveness of our rounding algorithm. Finally, in Section 9.6.4, we use a carefully designed problem to show how symmetries in the GCS can loosen the relaxation of our MICP.

All the experiments in this section use the commercial solver `MOSEK 10.0` with default options on a laptop computer with processor 2.4 GHz 8-Core Intel Core i9 and memory 64 GB 2667 MHz DDR4.

9.6.1 Two-dimensional example

We consider the two-dimensional problem in Figure 9-2. We have a graph G with $|\mathcal{V}| = 9$ vertices, $|\mathcal{E}| = 22$ edges, and multiple cycles. The source $\mathcal{X}_s := \{\boldsymbol{\theta}_s \in \mathbb{R}^2\}$ and target $\mathcal{X}_t := \{\boldsymbol{\theta}_t \in \mathbb{R}^2\}$ sets are single points, while the remaining regions are full dimensional. The geometry of the sets $\mathcal{X}_v$ and the edge set $\mathcal{E}$ can be deduced from Figure 9-2. We do not enforce any edge constraint $\mathcal{X}_e$. All the vertex costs f_v are zero. For the edge costs we consider the Euclidean distance

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \|\boldsymbol{x}_w - \boldsymbol{x}_v\|_2, \qquad \forall e = (v, w) \in \mathcal{E}, \tag{9.10}$$

and the Euclidean distance squared

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \|\boldsymbol{x}_w - \boldsymbol{x}_v\|_2^2, \qquad \forall e = (v, w) \in \mathcal{E}. \tag{9.11}$$

The shortest paths corresponding to the two edge costs are shown in Figure 9-2 in orange and blue. As expected, the first trajectory is almost straight while the lengths of the segments in the second are better balanced.

In Figure 9-3 we analyze the optimal values of the SPP in GCS and our convex relaxation for different values of a parameter $\sigma > 0$ that controls the volume of the sets $\mathcal{X}_v$. The value $\sigma = 1$ corresponds to the GCS in Figure 9-2. While for $\sigma \neq 1$ each set $\mathcal{X}_v$ is shrunk or enlarged via a uniform scaling, with scale factor σ, relative to a fixed Chebyshev center of the set (center of the inscribed circle of maximum area).

When the edge cost is the Euclidean distance, the top panel in Figure 9-3 shows that our relaxation is exact for all values of σ. This was expected for σ close to zero, since our relaxation is exact when the sets are singletons. Similarly, the problem is trivial for very large σ, when the regions are so big that, no matter the discrete

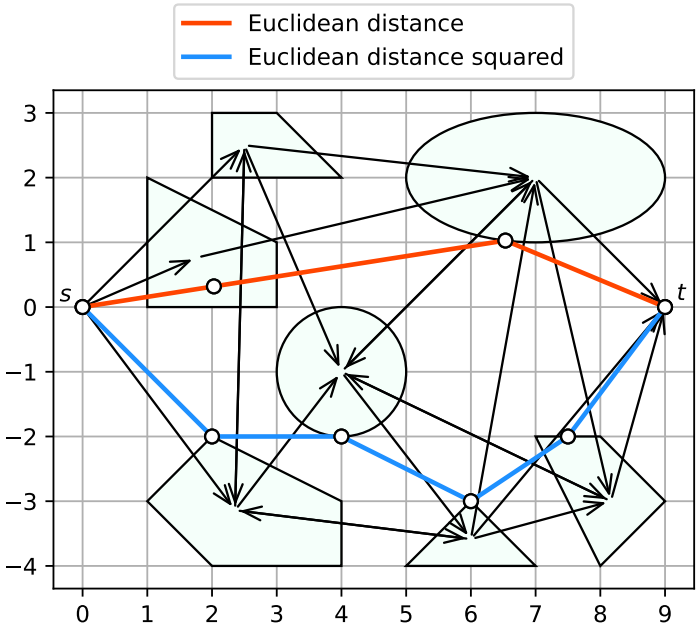

Figure 9-2: Two-dimensional SPP in GCS with the optimal solution depicted for two edge costs, the Euclidean distance and the Euclidean distance squared.

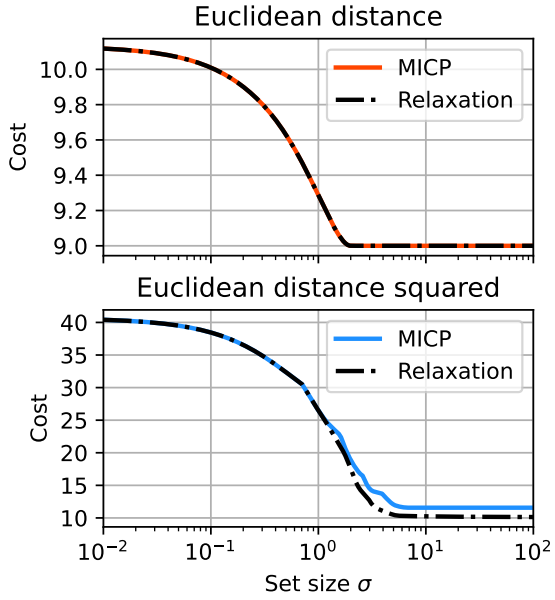

Figure 9-3: Optimal values of the SPP in GCS and its convex relaxation as functions of the edge cost and the size of the sets σ.

path we take, we can always reach the target via a straight line. However, that our relaxation is exact for all the intermediate values of σ is not an obvious result. With the Euclidean distance squared, our relaxation is still guaranteed to be tight as σ goes to zero. This is confirmed by the bottom panel of Figure 9-3. When σ is very large, our problem is equivalent to the HPP (see proof of Theorem 9.2), which is NP-complete. The optimal value of the problem is $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_s\|_2^2/l = 11.6$, where $l = 7$ is the number of edges in the longest s-t path in the graph in Figure 9-2. A close inspection of the bottom of Figure 9-3 reveals that, for large σ, our relaxation yields the lower bound $\|\boldsymbol{\theta}_t - \boldsymbol{\theta}_s\|_2^2/(|\mathcal{V}| - 1) = 10.1$, which corresponds to the simple inequality $l \leq |\mathcal{V}| - 1$. Using a duality argument, it can be verified that our relaxation always recovers this bound.

9.6.2 Large-scale random instances

We present a statistical analysis of the performance of our formulation. We generate a variety of random large-scale SPPs in GCS, and we analyze the relaxation tightness and the solution times of our MICP as functions of various problem parameters.

We stress that generating random graphs representative of the "typical" SPP in GCS we might encounter in practice is a difficult operation. Inevitably, the instances we describe below are not completely representative, and our algorithm might perform worse or better on other classes of random graphs. Our goal here is to show that our MICP is not limited to small-scale problems.

We construct an SPP in GCS as follows. We set $\mathcal{X}_s := \{\mathbf{0}\}$ and $\mathcal{X}_t := \{\mathbf{1}\}$. The rest of the sets $\mathcal{X}_v$ are axis-aligned hyper cubes with volume Λ and center drawn uniformly at random in $[0, 1]^n$. Given a number $|\mathcal{E}|$ of edges, we construct the edge set in two steps. First we generate multiple s-t paths such that every vertex $v \neq s, t$ is traversed exactly by one path. These are determined via a random partition of the set $\mathcal{V} - \{s, t\}$: the number of sets in the partition (number of paths) is drawn uniformly from the interval $[1, |\mathcal{V}| - 2]$, and also the number of vertices in each set (length of each path) is a uniform random variable. Then we extend the edge set by drawing edges uniformly at random from the set $\{(v, w) \in \mathcal{V}^2 : w \neq s, v \neq t, v \neq w\}$ until a desired cardinality $|\mathcal{E}|$ is reached. We do not enforce any edge constraint $\mathcal{X}_e$, and all the vertex costs f_v are zero. As edge costs f_e we consider the Euclidean distance (9.10) and the Euclidean distance squared (9.11), which both make our MICP a mixed-integer Second-Order-Cone Program(SOCP).

For each edge cost, we first solve 100 random instances with the following nominal parameters: volume $\Lambda = 0.01$, $n = 4$ dimensions, $|\mathcal{V}| = 50$ vertices, and $|\mathcal{E}| = 100$ edges. Then we solve four other batches of 100 problems where, in each batch, a

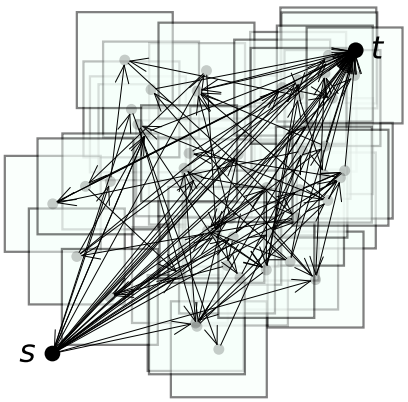

Figure 9-4: Projection onto two dimensions of a random instance of the SPP in GCS from Section 9.6.2. The problem parameters have nominal value.

different subset of these parameters is increased by a factor of 5. Specifically, these additional batches test our formulation in case of large sets $\mathcal{X}_v$ (Λ from 0.01 to 0.05), high dimensions (n from 4 to 20), dense graphs ($|\mathcal{E}|$ from 100 to 500), and large graphs ($|\mathcal{V}|$ and $|\mathcal{E}|$ from 50 and 100 to 250 and 500). To give an idea of what these problems look like, the projection onto two dimensions of a GCS generated using the nominal parameters is shown in Figure 9-4.

Figure 9-5 shows the relaxation gap (cost gap between the MICP and its relaxation, normalized by the MICP cost) versus the MICP solution time for all the instances described above. As observed in the previous example, the Euclidean edge cost (9.10) results in easier programs: our relaxation is tight in almost all the instances and the solution times are relatively low. The squared edge cost (9.11) leads to more challenging problems, even though the maximum relaxation gap and runtime are only 2.1% and 0.66s in the nominal case. When the volume of the cubes $\mathcal{X}_v$ is increased to $\Lambda = 0.05$ these values increase to 9.1% and 1.12s, and the performance of our MICP is minimally affected. Note that this is not in contrast with the previous example, where we analyzed the regime of extremely large sets $\mathcal{X}_v$. Note also that the volume of the sets does not affect the MICP size. The growth of the space dimension to $n = 20$ increases the size of our programs, and also loosens the relaxation. The largest relaxation gap is 28.9%, and our MICP takes 72s to be solved in the worst case. Similarly, when the number $|\mathcal{E}|$ of edges is increased to 500 the maximum relaxation gap and runtime become 32.9% and 174s. This is due to the combination of the quadratic edge cost and the large number of cycles that we have in a graph with high density of edges $|\mathcal{E}|/|\mathcal{V}|$. To show this, in the last batch of problems we keep $|\mathcal{E}| = 500$ and we increase the number of vertices to $|\mathcal{V}| = 250$. This increases the MICP size further but makes the graph sparser, reducing the maximum relaxation

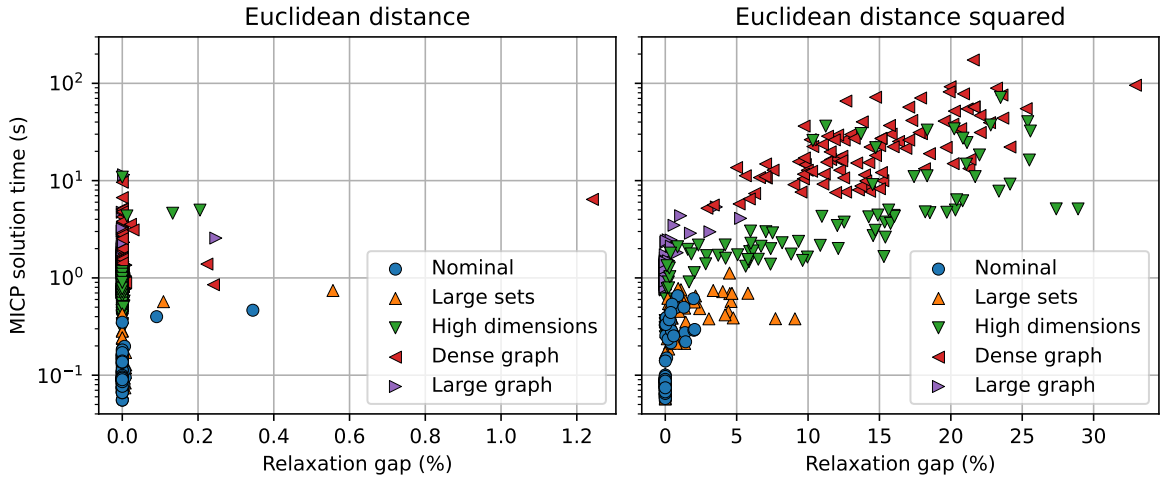

Figure 9-5: Relaxation gap versus MICP solution time for the 500 random instances described in Section 9.6.2. Two edge costs are analyzed: the Euclidean distance (9.10) and the Euclidean distance squared (9.11). For each edge cost, 100 nominal instances are generated with the nominal problem parameters, and four other batches of 100 instances each are obtained by increasing a different subset of the parameters. Our relaxation is almost always exact with the Euclidean cost. While, with the Euclidean cost squared, it is more sensitive to the dimension n of the space and the density of the graph G. (Note the different horizontal scales of the two plots.)

gap and runtime to 5.3% and 5.4s.

9.6.3 Evaluation of the rounding algorithm

We continue the previous example with an analysis of the performance of the rounding algorithm described in Section 9.5. Given that with the Euclidean edge cost our relaxation is almost always exact (see left panel in Figure 9-5), here we only consider the squared edge cost (9.11). For each batch of problem instances described in the previous subsection, we report in Figure 9-6 the histogram of:

- The optimality gap $\delta_{\text{opt}} := (f_{\text{round}} - f_{\text{opt}})/f_{\text{opt}}$, obtained by comparing the cost of the rounded solution to the optimal value of the MICP.
- The certified optimality gap $\delta_{\text{relax}} := (f_{\text{round}} - f_{\text{relax}})/f_{\text{relax}}$ which upper bounds optimality gap δ_{opt} and is obtained for free from our rounding algorithm.

The colors used in Figure 9-6 match the ones in Figure 9-5. For what concerns the numerical parameters in our rounding algorithm, we set to five the number of distinct paths and to one hundred the maximum number of random trials. Note also that, since these instances of the SPP in GCS do not have edge constraints, our

rounding strategy is guaranteed to always identify a feasible solution (see discussion in Section 9.5).

As shown in Figure 9-6, the optimality gaps δ_{opt} are almost always zero, i.e., our rounding strategy recovers an optimal solution most of the times. The main exception comes from the instances with dense graphs (red histograms). As noted above, these are the instances where our convex relaxation is not very tight; but even in this case the rounding yields an optimality gap smaller than 20% most of the times, and always smaller than 55%. A similar analysis can be done for the certified optimality gap δ_{relax}, which tightly upper bounds the actual optimality gap δ_{opt} for most of the problems.

Importantly, the rounded solutions in this analysis are computed in a fraction of the time that is needed to solve the corresponding MICPs. For the instances with large graphs G, the rounding algorithms takes between 0.8s and 4.6s (without any parallelization of the random trials). For all the other instances, it takes between 0.01s and 0.3s. Note that this is dramatically faster than the tens or hundreds of seconds necessary to solve some of the MICPs (see right panel in Figure 9-5).

9.6.4 Symmetric problems

We conclude by showing how symmetries in the GCS can deteriorate the convex relaxation of our MICP and, in principle, make it arbitrarily loose. We illustrate this through the following carefully designed problem.

We consider the SPP in GCS depicted in Figure 9-7. We have an acyclic graph with $|\mathcal{V}| = 5$ vertices and $|\mathcal{E}| = 5$ edges. All the sets $\mathcal{X}_v$ are singletons $\{\boldsymbol{\theta}_v\}$, except for $\mathcal{X}_3$ which is a full-dimensional rectangle. As an edge cost, we use the Euclidean distance (9.10). Solving this problem, we obtain the optimal path $p = (s, 1, 3, t)$ with cost 7.4 (the symmetric solution $p = (s, 2, 3, t)$ would also be optimal). The corresponding vertex positions are connected by an orange line in Figure 9-7.

Figure 9-8 illustrates the solution of the relaxation of the MICP (5.8). For each edge $e = (v, w)$, we connect the optimal points $\bar{\boldsymbol{z}}_v^e := \boldsymbol{z}_v^e / y_e$ and $\bar{\boldsymbol{z}}_w^e := \boldsymbol{z}_w^e / y_e$ with an orange line, labeled in blue with the corresponding flow y_e. Note that, for $y_e > 0$, we have $\tilde{f}_e(\boldsymbol{z}_v^e, \boldsymbol{z}_w^e, y_e) = f_e(\bar{\boldsymbol{z}}_v^e, \bar{\boldsymbol{z}}_w^e) y_e$, and the vectors $\bar{\boldsymbol{z}}_v^e$ and $\bar{\boldsymbol{z}}_w^e$ are the actual points where the cost of the edge e is evaluated. Note also that, we have $\bar{\boldsymbol{z}}_v^e \in \mathcal{X}_v$ and $\bar{\boldsymbol{z}}_w^e \in \mathcal{X}_w$. The relaxation splits the unit of flow injected in the source into two: half unit is shipped to the target via the top path, the other half via the bottom path. The optimal value of this convex program is 7.0.

The looseness of the relaxation can be explained as follows. If we denote with ρ the flow traversing edge $(1, 3)$, the flow conservation gives $y_{(2,3)} = 1 - \rho$, while the flow

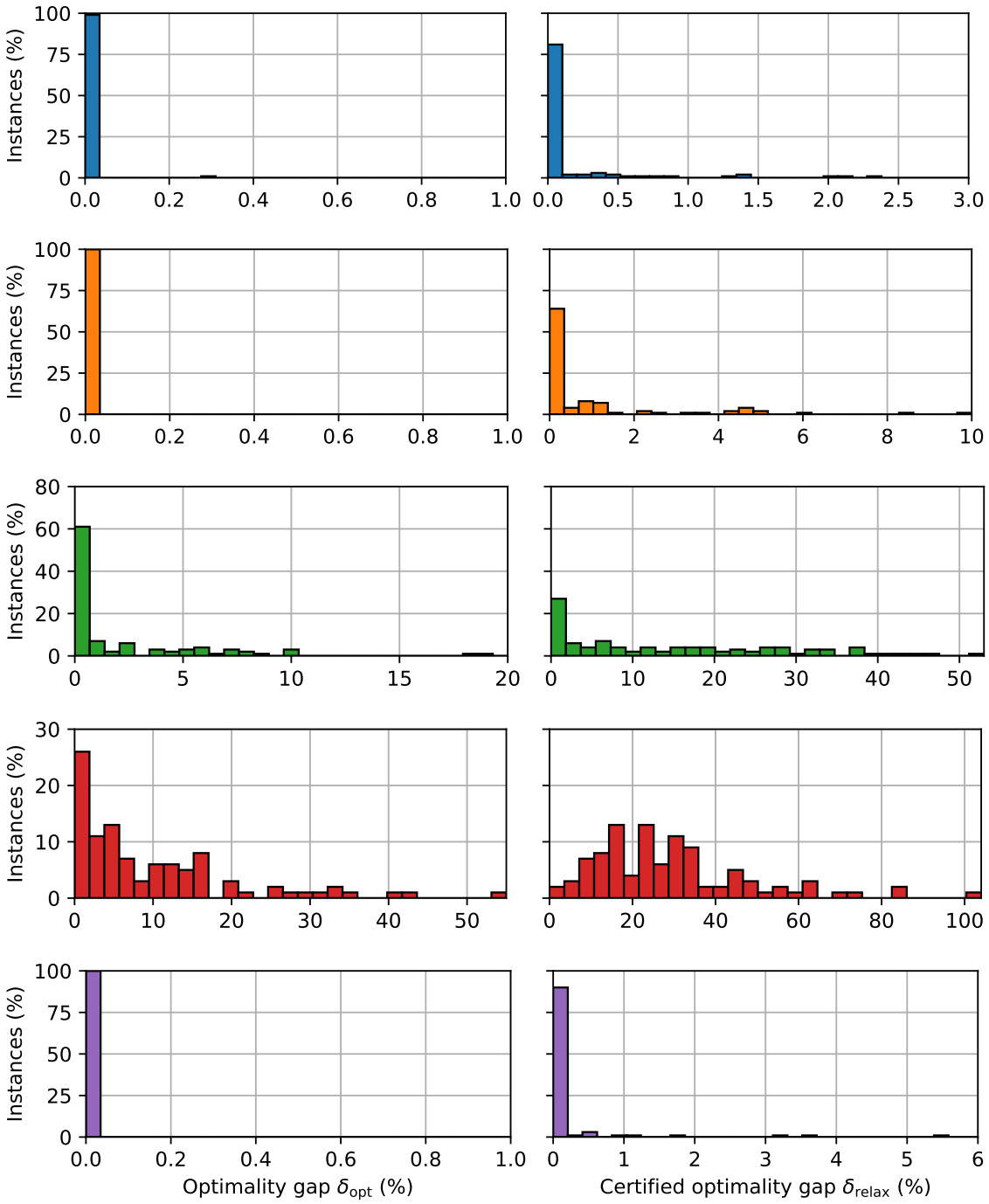

Figure 9-6: Histograms of the optimality gap and the certified optimality gap for the 500 instances of the SPP in GCS described in Section 9.6.2. From top to bottom, SPP in GCS with: nominal parameters, large sets $\mathcal{X}_v$, high-dimensional sets $\mathcal{X}_v$, dense graph G, and large graph G. Note the different scales both on the horizontal and the vertical axis.

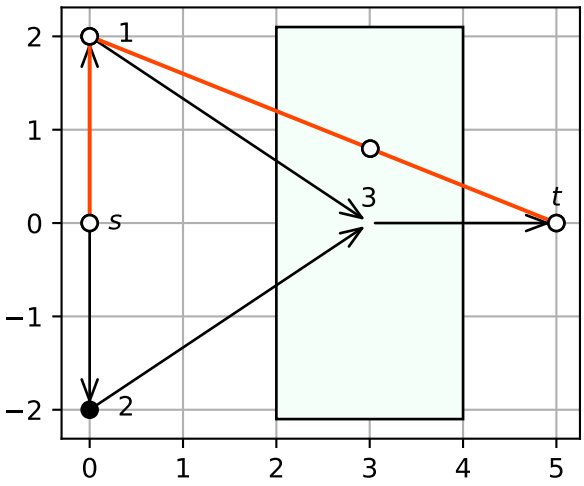

Figure 9-7: Instance of the SPP in GCS that shows how symmetries in the GCS can deteriorate the convex relaxation of our MICP. The optimal optimal vertex positions are connected by orange lines.

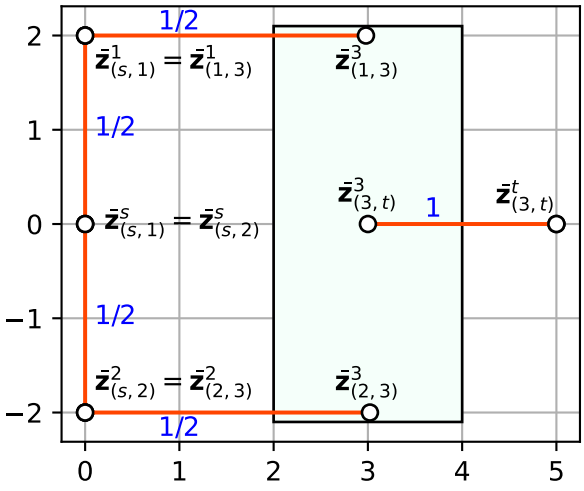

Figure 9-8: Optimal solution of the relaxation. The cost contribution of edge e is obtained by multiplying the flow y_e by the distance between $\bar{\boldsymbol{z}}^e_v$ and $\bar{\boldsymbol{z}}^e_w$. Since only the mean of $\bar{\boldsymbol{z}}^3_{(1,3)}$ and $\bar{\boldsymbol{z}}^3_{(2,3)}$ is required to match $\bar{\boldsymbol{z}}^3_{(3,t)}$, the cost is minimized by moving these two points closer to $\bar{\boldsymbol{z}}^1_{(1,3)}$ and $\bar{\boldsymbol{z}}^2_{(2,3)}$, respectively. For each edge $e = (v, w)$, the orange line connects the surrogates $\bar{\boldsymbol{z}}^e_v$ and $\bar{\boldsymbol{z}}^e_w$ of the vertex positions $\boldsymbol{x}_v$ and $\boldsymbol{x}_w$, and is labeled with the flow y_e.

through the edge $(3, t)$ is always one. Since the variables $\bar{\boldsymbol{z}}^1_{(1,3)}$, $\bar{\boldsymbol{z}}^2_{(2,3)}$, and $\bar{\boldsymbol{z}}^t_{(3,t)}$ are forced to match $\boldsymbol{\theta}_1$, $\boldsymbol{\theta}_2$, and $\boldsymbol{\theta}_t$, respectively, the cost terms in (9.5a) corresponding to the edges $(1, 3)$, $(2, 3)$, and $(3, t)$ read

$$\rho\|\bar{\boldsymbol{z}}^3_{(1,3)} - \boldsymbol{\theta}_1\|_2 + (1-\rho)\|\bar{\boldsymbol{z}}^3_{(2,3)} - \boldsymbol{\theta}_2\|_2 + \|\boldsymbol{\theta}_t - \bar{\boldsymbol{z}}^3_{(3,t)}\|_2. \tag{9.12}$$

The only constraint that links these variables is (9.5d) for $v = 3$, which gives $\rho\bar{\boldsymbol{z}}^3_{(1,3)} + (1-\rho)\bar{\boldsymbol{z}}^3_{(2,3)} = \bar{\boldsymbol{z}}^3_{(3,t)}$. When $\rho = 1/2$, this constraint asks the mean of $\bar{\boldsymbol{z}}^3_{(1,3)}$ and $\bar{\boldsymbol{z}}^3_{(2,3)}$ to match $\bar{\boldsymbol{z}}^3_{(3,t)}$, as opposed to forcing either one of the first two points to match the third, as it would be for $\rho \in \{0, 1\}$. Therefore, while keeping their mean equal to $\bar{\boldsymbol{z}}^3_{(3,t)}$, the points $\bar{\boldsymbol{z}}^3_{(1,3)}$ and $\bar{\boldsymbol{z}}^3_{(2,3)}$ can move vertically, and get closer to $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$. This reduces the first two terms in (9.12), and keeps the third term unchanged.

Although this example leads to a relaxation gap of only 5%, a simple variation of it shows that our relaxation can be arbitrarily loose. In particular, if we let $f_{(s,1)} := f_{(s,2)} := 0$ and we shift the centers of the sets $\mathcal{X}_3$ and $\mathcal{X}_t$ to the origin, then the cost of the MICP and its relaxation are reduced to 2 and 0, and the relaxation gap becomes 100%. Nevertheless, we emphasize that this is a contrived problem, and the instances we encounter in practice lead to these phenomena very rarely.

Chapter 10

Applications in optimal control

The goal of this chapter is showing how multiple classes of optimal-control problems can be reduced to a Shortest-Path Problem (SPP) in Graphs of Convex Sets (GCS). The techniques presented in this thesis can then be applied to automatically formulate these control problems as highly efficient Mixed-Integer Convex Programs (MICP). We will see that the mixed-integer formulations introduced in this chapter either generalize or significantly improve upon the ones that have been previously proposed in the literature.

The material in this chapter extends the one introduced in [131, Section 8]. It also generalizes many of the results presented in [129].

10.1 Minimum-time control of discrete-time linear systems

Consider the discrete-time linear dynamical system

$$\boldsymbol{s}_{k+1} = \boldsymbol{A}\boldsymbol{s}_k + \boldsymbol{B}\boldsymbol{a}_k, \tag{10.1}$$

where $\boldsymbol{s}_k \in \mathbb{R}^q$ and $\boldsymbol{a}_k \in \mathbb{R}^r$ are the system state and control action at time step $k \in \mathbb{Z}_{\geq 0}$. Given the initial conditions $\boldsymbol{s}_0 = \hat{\boldsymbol{s}}$, we look for a sequence of controls that drives the system state to a target compact convex set $\mathcal{T} \subset \mathbb{R}^q$ in the minimum number $K \in \mathbb{Z}_{\geq 0}$ of time steps. At each discrete time k, the state and control pair $(\boldsymbol{s}_k, \boldsymbol{a}_k)$ is constrained in a compact convex set $\mathcal{D} \subset \mathbb{R}^{q+r}$. The optimization problem

can be summarized as follows:

$$\begin{aligned}
\text{minimize} \quad & K && && (10.2a)\\
\text{subject to} \quad & \boldsymbol{s}_0 = \hat{\boldsymbol{s}}, && && (10.2b)\\
& (\boldsymbol{s}_k, \boldsymbol{a}_k) \in \mathcal{D}, && \forall k = 0, \dots, K-1, && (10.2c)\\
& \boldsymbol{s}_{k+1} = \boldsymbol{A}\boldsymbol{s}_k + \boldsymbol{B}\boldsymbol{a}_k, && \forall k = 0, \dots, K-1, && (10.2d)\\
& \boldsymbol{s}_K \in \mathcal{T}. && && (10.2e)
\end{aligned}$$

Remark 10.1. Of course, a simple method to solve problem (10.2) is to do bisection over the number time steps K, and solve a Linear Program (LP) with no objective function at every iteration.[1] On the other hand, bisection becomes quickly impractical as we consider more complex problems. For example, the technique presented in this section can be immediately combined with the ones in the next section to solve minimum-time problems for hybrid dynamical systems. For these problems our method yields a single and strong MICP, while the bisection approach would need to solve an MICP at every iteration.

To formulate problem (10.2) as an SPP in GCS we proceed as in Figure 10-1. The vertices in our graph are ordered in a sequence: $\mathcal{V} := \{0, 1, \dots, \bar{K}-1, t\}$ where $\bar{K}$ is a given upper bound on the optimal time horizon K. The source $s = 0$ is the first vertex and the target t is the last. Each vertex $v = 0, \dots, \bar{K}-1$ has two outgoing edges: $(v, v+1)$ that connects it to the next vertex in the sequence, and (v, t) that goes to the target. For each of these vertices, the continuous variable $\boldsymbol{x}_v := (\boldsymbol{s}_v, \boldsymbol{a}_v)$ represents the state and control pair at time $k = v$ (provided that we have not reached the target set yet). The variables $\boldsymbol{x}_t := \boldsymbol{s}_t$ paired with the target vertex represent the system state at the final time K. These variables are constrained by the following sets:

$$\begin{aligned}
&\mathcal{X}_s := \{(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D} : \boldsymbol{s} = \hat{\boldsymbol{s}}\},\\
&\mathcal{X}_v := \mathcal{D}, && \forall v = 1, \dots, \bar{K}-1,\\
&\mathcal{X}_t := \mathcal{T}.
\end{aligned}$$

Along every edge $e = (v, w) \in \mathcal{E}$ we require that the state and control at vertex v is coupled to the state at vertex w through the linear dynamics (10.1). This is achieved by pairing each edge $e = (v, w)$ with the convex set $\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w) : \boldsymbol{s}_w =$

[1] Formally, this approach assumes that the set $\mathcal{T}$ is invariant, i.e., for all $\boldsymbol{s} \in \mathcal{T}$ there exists $\boldsymbol{a} \in \mathbb{R}^r$ such that $(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}$ and $\boldsymbol{A}\boldsymbol{s} + \boldsymbol{B}\boldsymbol{a} \in \mathcal{T}$. A simple example of an invariant set is $\mathcal{T} := \{\boldsymbol{0} \in \mathbb{R}^q\}$, provided that $\mathcal{D}$ contains the origin.

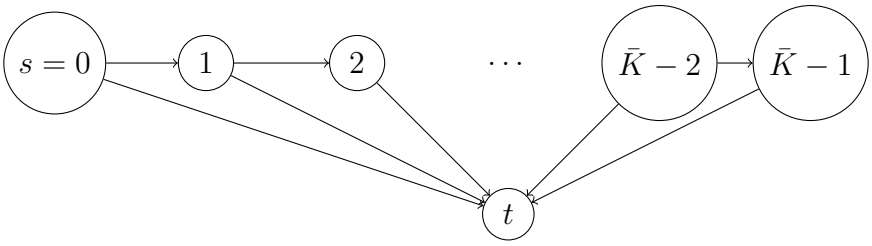

Figure 10-1: Formulation of the minimum-time control problem for discrete-time linear systems as an SPP in GCS.

$\boldsymbol{A}\boldsymbol{s}_v + \boldsymbol{B}\boldsymbol{a}_v\}$.

To minimize the number of edges in the optimal path (i.e., the time steps to reach the target set), we assign unit cost to every edge: $f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := 1$ for all $e = (v, w) \in \mathcal{E}$. The costs f_v of all the vertices are set to zero.

The solution of the SPP in GCS just constructed gives us a path in the graph in Figure 10-1. The optimal time horizon K is the length of this path. The optimal state and control sequence is given by the variables $\boldsymbol{s}_v$ and $\boldsymbol{a}_v$ paired with the vertices v along this path.

The graph in Figure 10-1 has no cycles and vertex costs. Therefore the corresponding MICP can be formulated as described in Section 9.3. Just for analysis purposes, let us report explicitly the MICP that results from the application our techniques (but note that in practice a user of our method would only have to assemble the GCS in Figure 10-1, and the MICP below would be constructed automatically). After multiple simplifications, our MICP reduces to the following optimization problem:

$$
\begin{aligned}
\text{minimize} \quad & 1 + \sum_{k=0}^{\bar{K}-2} y_k && && \text{(10.3a)}\\
\text{subject to} \quad & y_k, y'_k \in \{0,1\}, && \forall k = 0, \dots, \bar{K}-1, && \text{(10.3b)}\\
& y_0 + y'_0 = 1, && && \text{(10.3c)}\\
& y_{k+1} + y'_{k+1} = y_k, && \forall k = 0, \dots, \bar{K}-2, && \text{(10.3d)}\\
& y_{\bar{K}-1} = 0, && && \text{(10.3e)}\\
& \boldsymbol{\sigma}_0 = y_0 \hat{\boldsymbol{s}}, && && \text{(10.3f)}\\
& \boldsymbol{\sigma}'_0 = y'_0 \hat{\boldsymbol{s}}, && && \text{(10.3g)}\\
& (\boldsymbol{\sigma}_k, \boldsymbol{\alpha}_k, y_k) \in \tilde{\mathcal{D}}, && \forall k = 0, \dots, \bar{K}-1, && \text{(10.3h)}\\
& (\boldsymbol{\sigma}'_k, \boldsymbol{\alpha}'_k, y'_k) \in \tilde{\mathcal{D}}, && \forall k = 0, \dots, \bar{K}-1, && \text{(10.3i)}\\
& \boldsymbol{A}\boldsymbol{\sigma}_k + \boldsymbol{B}\boldsymbol{\alpha}_k = \boldsymbol{\sigma}_{k+1} + \boldsymbol{\sigma}'_{k+1}, && \forall k = 0, \dots, \bar{K}-2, && \text{(10.3j)}\\
& \boldsymbol{A}\boldsymbol{\sigma}'_k + \boldsymbol{B}\boldsymbol{\alpha}'_k \in \mathcal{T}, && \forall k = 0, \dots, \bar{K}-1. && \text{(10.3k)}
\end{aligned}
$$

The binary variable y_k takes a value of zero if at time step $k+1$ we reach the target set, and takes a value of one otherwise. The variable y'_k has the opposite role, it is one if we reach the target set and zero otherwise. The variables $\boldsymbol{\sigma}_k$ and $\boldsymbol{\alpha}_k$ represent the state and the control of the system at time k if $y_k = 1$, and are zero otherwise. Similarly, $\boldsymbol{\sigma}'_k$ and $\boldsymbol{\alpha}'_k$ represent the state and the control of the system at time k if $y'_k = 1$, and are zero otherwise. Therefore the actual system state and control action can be reconstructed as

$$(\boldsymbol{s}_k, \boldsymbol{a}_k) := (\boldsymbol{\sigma}_k, \boldsymbol{\alpha}_k) + (\boldsymbol{\sigma}'_k, \boldsymbol{\alpha}'_k),$$

for each time step $k = 0, \ldots, \bar{K} - 1$. The objective function minimizes the number of time steps that we spend before reaching the target set plus one. The second constraint states that at the first time step we either reach the target set or not. The third constraint that, if at time k we have not reached the target set, then we can try again at time $k+1$. The fourth constraint says that at time $\bar{K} - 1$ we must reach the target set if we have not done so already. The fifth to eighth conditions enforce the necessary constraints on the states and controls, depending on whether we have reached the target or not. The last two constraints enforce the linear dynamics, according to the values of the binary variables.

The costs and constraints in problem (10.3) are interpretable relatively easily. However, formulating this MICP directly would be a difficult task even for an expert in mixed-integer optimization. The advantage of our approach is that the MICP (10.3) is derived automatically: a user has only to specify the simple GCS in Figure 10-1, and our techniques from Chapter 9 take care of all the optimization details.

10.1.1 Comparison with existing formulations

Mixed-integer formulations for minimum-time control have been studied for more than thirty years (see, e.g., [33, 164, 161]). In the simple case in which the target set $\mathcal{T}$ contains only the origin (and also the set $\mathcal{D}$ contains the origin), the formulations

previously presented in the literature are variants of the following program:[2]

$$\begin{aligned}
\text{minimize} \quad & 1 + \sum_{k=0}^{\bar{K}-2} y_k && && \text{(10.4a)}\\
\text{subject to} \quad & y_k \in \{0,1\}, && \forall k = 0,\dots,\bar{K}-2, && \text{(10.4b)}\\
& \boldsymbol{s}_0 = \hat{\boldsymbol{s}}, && && \text{(10.4c)}\\
& (\boldsymbol{s}_0, \boldsymbol{a}_0) \in \mathcal{D}, && && \text{(10.4d)}\\
& (\boldsymbol{s}_k, \boldsymbol{a}_k, y_{k-1}) \in \tilde{\mathcal{D}}, && \forall k = 1,\dots,\bar{K}-2, && \text{(10.4e)}\\
& \boldsymbol{s}_{k+1} = \boldsymbol{A}\boldsymbol{s}_k + \boldsymbol{B}\boldsymbol{a}_k, && \forall k = 0,\dots,\bar{K}-1, && \text{(10.4f)}\\
& \boldsymbol{s}_{\bar{K}} = \boldsymbol{0}. && && \text{(10.4g)}
\end{aligned}$$

Here the role of the binaries y_k is the same as above ($y_k = 0$ if at time $k+1$ we reach the origin, and $y_k = 1$ otherwise), while the vectors $\boldsymbol{s}_k$ and $\boldsymbol{a}_k$ represent the actual system state and control at discrete time k.

Our formulation in (10.3) is easily verified to be stronger than the one in (10.4): the constraints in problem (10.3) imply the ones in (10.4), but not vice-versa.[3] This means that the convex relaxation of our SPP in GCS is guaranteed to be a tighter approximation of the original nonconvex problem than the convex relaxation of (10.4).

10.1.2 Numerical example: double integrator

To illustrate the relations between our formulation (10.3) and the one in (10.4), we consider the simple minimum-time problem depicted in Figure 10-2. We have a double-integrator system with

$$\boldsymbol{A} := \begin{bmatrix} 1 & h \\ 0 & 1 \end{bmatrix}, \qquad \boldsymbol{B} := \begin{bmatrix} 0 \\ h \end{bmatrix},$$

where $h := 0.01$ is the time discretization step. The constraint set $\mathcal{D}$ enforces a limit of one on the absolute value of each state and the control action: $\mathcal{D} := [-1, 1]^3$. The maximum number of time steps $\bar{K}$ is set to 250.

The top panel in Figure 10-2 illustrates the optimal state-space trajectories for a

[2] Actually, to the best of our knowledge, previous formulations of this problem relied on the big-M procedure and did not use the homogenization as in (10.4e). Therefore, the MICP (10.4) is already a more advanced (and more efficient version) of the formulations that have previously appeared in the literature.

[3] Take any feasible solution of the convex relaxation of (10.3). Define $(\boldsymbol{s}_k, \boldsymbol{a}_k) := (\boldsymbol{\sigma}_k, \boldsymbol{\alpha}_k) + (\boldsymbol{\sigma}'_k, \boldsymbol{\alpha}'_k)$ for $k = 0,\dots,\bar{K}-1$ and $\boldsymbol{s}_{\bar{K}} = \boldsymbol{0}$. Then all the constraints of the convex relaxation of (10.4) can be verified to hold.

variety of initial conditions $\hat{\boldsymbol{s}}$. The first component of the state is labelled as "position" and the second as "velocity." Close to each initial state we report the optimal value of the corresponding problem (i.e., the minimum number of time steps to reach the origin).

The central panel in Figure 10-2 reports the optimal solution of the convex relaxation of the MICP (10.4). As it can be seen, the trajectories generated by this convex relaxation do not approximate the optimal trajectories well. Also the optimal values of this relaxation are loose lower bounds on the optimal values of the MICP: the minimum and maximum relaxation gap are 28.9% and 57.7%. The bottom panel in Figure 10-2 illustrates the same results for the convex relaxation of our MICP (10.3). In this case the trajectories look more alike the optimal ones, and the lower bounds are tighter (minimum relaxation gap is 6.8% and maximum is 39.8%).

While our MICP is provably stronger, it has more variables and constraints than the simple MICP (10.4). Therefore its solution times are not necessarily lower. In the problems above the two formulations are roughly equivalent in terms of actual solution times of the overall MICP. However, we emphasize again that the main advantage of our approach is that it transfers without modifications to more complex control problems, while the MICP (10.4) is handcrafted for this specific class of problems.

10.2 Regulation of discrete-time piecewise-affine systems

PieceWise-Affine (PWA) systems are a very flexible framework for modeling a great variety of phenomena. Originally they were introduced as a generalization of linear systems and an approximation of more general classes of nonlinear systems [173]; later they became very popular as a more intuitive equivalent of several other families of hybrid systems [92], such as Mixed Logical Dynamical (MLD) systems. Their structure makes them very suitable for different types of analysis (e.g., stability [99], observability and controllability [10]) as well as optimal control [159, 12]. PWA systems have seen many applications [30], to mention a few: automotive control [22], power electronics [77], temporal logic control [194], and they also have recently spread in robot locomotion [124, 90] and manipulation [91]. Nowadays, together with MLD, they are the most popular modeling framework for hybrid systems in Model Predictive Control (MPC) [30, 23]. PWA dynamics are also efficiently fitted to data [63, 144, 119] and compressed into low-complexity models [78, 119].

In this section we show how optimal-control problems for PWA systems can be

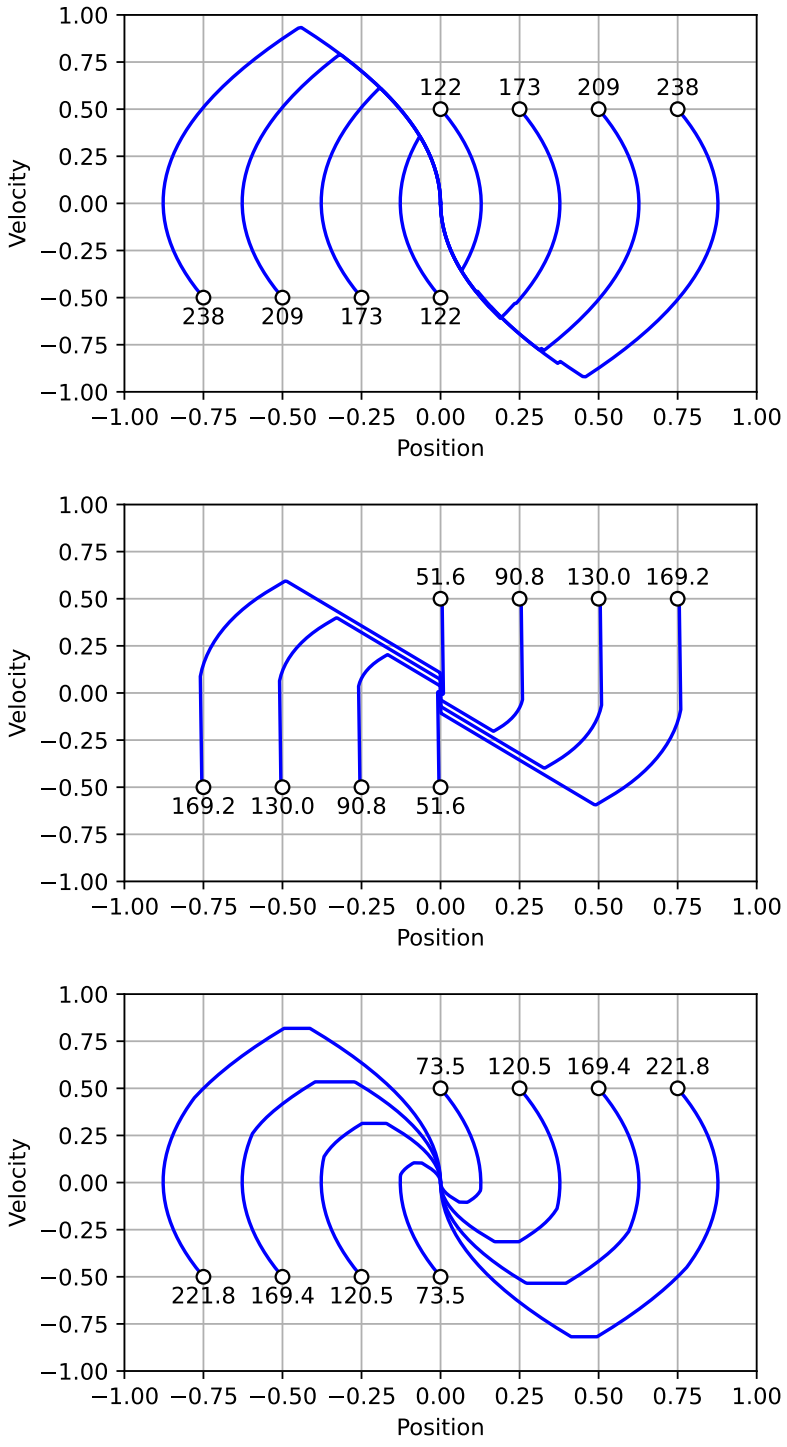

Figure 10-2: Minimum-time control of the discrete-time double integrator. *Top:* optimal solution the control problem. *Center:* optimal solution of the convex relaxation of the baseline MICP (10.4). *Bottom:* optimal solution of the convex relaxation of our MICP (10.3). In all the panels the initial states are labelled with the optimal cost of the corresponding optimization problem. The relaxation of our formulation yields a significantly tighter approximation of the optimal solution.

formulated as SPPs in GCS, and then automatically transcribed as efficient MICPs. Efficient mixed-integer formulations for these problems have been previously studied in [137, 129]. The approach that we present in this thesis generalizes the results in both these papers. We also mention that tight convex relaxations of a similar family of problems have been analyzed in [150, 7] (specifically, semidefinite relaxations for linear systems with discrete control inputs).

A strong mixed-integer formulation is just the first step to an efficient solution of these control problems: several customized algorithms have been presented in the literature for the solution of MICPs arising in hybrid MPC [11, 6, 70, 179, 71]. Also other approaches, not based on mixed-integer optimization, have been proposed for the optimal control of PWA systems, for example explicit MPC [21] or nonlinear-programming techniques [93]. The applicability of the first, however, is limited to small problems, while the second requires continuity properties that not all the PWA systems of interest might enjoy, and does not provide convergence and optimality guarantees.

10.2.1 Problem statement

Given a finite collection $\{\mathcal{D}_1, \dots, \mathcal{D}_I\}$ of compact convex subsets of the state and control space, a PWA system has dynamics

$$\boldsymbol{s}_{k+1} = \boldsymbol{A}_{i_k}\boldsymbol{s}_k + \boldsymbol{B}_{i_k}\boldsymbol{a}_k + \boldsymbol{c}_{i_k}, \qquad (\boldsymbol{s}_k, \boldsymbol{a}_k) \in \mathcal{D}_{i_k}.$$

The index $i_k \in \{1, \dots, I\}$ represents the system discrete mode at time k, which is itself a decision variable. We consider the problem of driving a PWA system from a given initial state $\boldsymbol{s}_0 = \hat{\boldsymbol{s}}$ to a target set $\mathcal{T}$, in a fixed number K of time steps.[4] The objective is to minimize the sum of a stage costs $\gamma(\boldsymbol{s}_k, \boldsymbol{a}_k)$ for $k = 0, \dots, K-1$, and a terminal cost $\varphi(\boldsymbol{s}_K)$. The functions $\gamma : \mathbb{R}^{q+r} \to \mathbb{R}$ and $\varphi : \mathbb{R}^q \to \mathbb{R}$ are convex. The

[4]The variant of this problem where K is a decision variable can be easily tackled by combining the techniques presented in this section with the ones from the previous section.

overall optimization can be stated as

$$\begin{aligned}
\text{minimize} \quad & \sum_{k=0}^{K-1} \gamma(\boldsymbol{s}_k, \boldsymbol{a}_k) + \varphi(\boldsymbol{s}_K) && && \text{(10.5a)}\\
\text{subject to} \quad & \boldsymbol{s}_0 = \hat{\boldsymbol{s}}, && && \text{(10.5b)}\\
& (\boldsymbol{s}_k, \boldsymbol{a}_k) \in \mathcal{D}_{i_k}, && \forall k = 0, \dots, K-1, && \text{(10.5c)}\\
& \boldsymbol{s}_{k+1} = \boldsymbol{A}_{i_k}\boldsymbol{s}_k + \boldsymbol{B}_{i_k}\boldsymbol{a}_k + \boldsymbol{c}_{i_k}, && \forall k = 0, \dots, K-1, && \text{(10.5d)}\\
& i_k \in \mathcal{I}, && \forall k = 0, \dots, K-1, && \text{(10.5e)}\\
& \boldsymbol{s}_K \in \mathcal{T}, && && \text{(10.5f)}
\end{aligned}$$

where we defined $\mathcal{I} := \{1, \dots, I\}$. Here both the states and controls $(\boldsymbol{s}_k, \boldsymbol{a}_k)$ and the system mode i_k are decision variables.

Problem (10.5) can be formulated as an SPP in GCS in multiple ways. In the next subsections we describe two pairs of formulations, that yield a different compromise between strength and size. These will be tested numerically in Sections 10.2.5 and 10.2.6.

10.2.2 Small but weak formulations

The first formulation of problem (10.5) as an SPP in GCS that we analyze is illustrated in Figure 10-3. The MICP resulting from this construction will be lightweight, but its convex relaxation will not be very tight.

The source $s = 0$ is the leftmost vertex and the target $t = K$ is the rightmost. In between, we have K layers of vertices labelled by two numbers: the first number is the time step k, the second is the system mode i. These layers are interleaved by auxiliary vertices that are labelled only with the time step k. The source is connected via an edge to each vertex in the first layer. Then each vertex in the first layer is connected to the first auxiliary vertex. This pattern is repeated until the final layer, whose vertices are connected to the target.

The continuous variables $\boldsymbol{x}_v$ paired with the vertices $v = 0, 1, \dots, K-1$ represent the system state $\boldsymbol{s}_k$ and control $\boldsymbol{a}_k$ at time $k = v$. The variable $\boldsymbol{x}_t$ paired with the target vertex t represents the final state of the system $\boldsymbol{s}_K$. The variables $\boldsymbol{x}_v$ paired with the remaining vertices (i.e., the vertices with labels of the form (k, i)) represent the state and control pair $(\boldsymbol{s}_k, \boldsymbol{a}_k)$ if at time k we are in mode i (i.e., if $i_k = i$). The

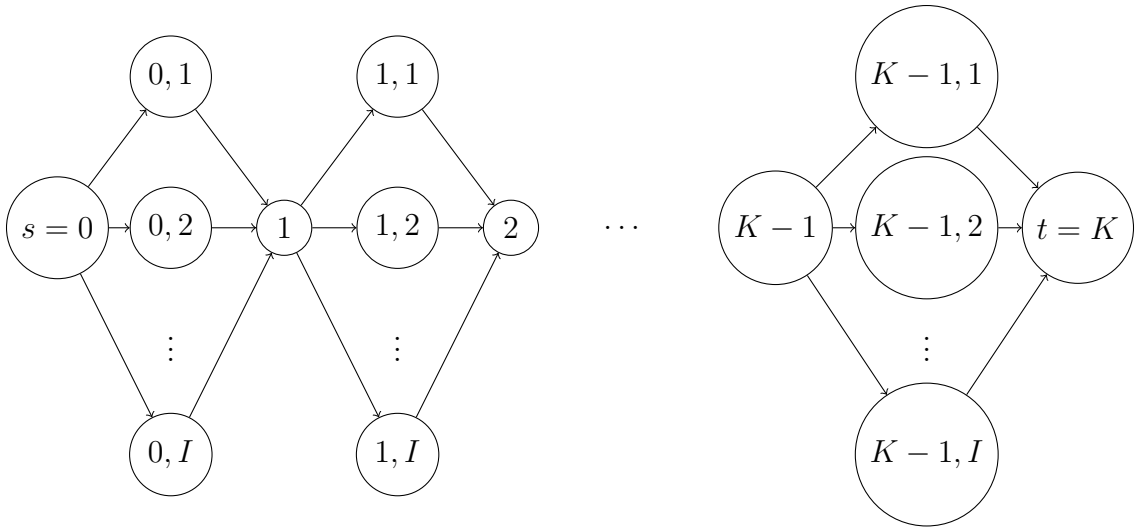

Figure 10-3: First formulation of the regulation problem for discrete-time PWA systems as an SPP in GCS. Constructing the GCS in this way leads to MICPs that are small but also weak.

variables just defined are constrained by the following convex sets:

$$\begin{aligned}
&\mathcal{X}_s := \{(\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D} : \boldsymbol{s} = \hat{\boldsymbol{s}}\}, \\
&\mathcal{X}_v := \mathcal{D}, && \forall v = 1, \dots, K-1, \\
&\mathcal{X}_v := \mathcal{D}_i, && \forall v = (k, i) \in \{0, \dots, K-1\} \times \mathcal{I}, \\
&\mathcal{X}_t := \mathcal{T},
\end{aligned}$$

where $\mathcal{D} \subset \mathbb{R}^{q+r}$ is a compact convex set that is large enough to contain all the sets $\mathcal{D}_i$ for $i \in \mathcal{I}$. (The choice of this set will actually be irrelevant for the final MICP.)

Along every edge $e = (v, w)$ that connects a vertex $v = k$ to a vertex $w = (k, i)$ we enforce the continuity of the state and the control through the edge constraint

$$\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w) : (\boldsymbol{s}_v, \boldsymbol{a}_v) = (\boldsymbol{s}_w, \boldsymbol{a}_w)\}.$$

For every edge of the $e = (v, w)$ that connects a vertex $v = (k, i)$ to a vertex $w = k+1$ we enforce the system dynamics via the edge constraint

$$\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w) : \boldsymbol{s}_w = \boldsymbol{A}_i \boldsymbol{s}_v + \boldsymbol{B}_i \boldsymbol{a}_v + \boldsymbol{c}_i\}.$$

The cost of each vertex $v = 0, \dots, K-1$ is equal to $f_v(\boldsymbol{x}_v) := \gamma(\boldsymbol{s}_v, \boldsymbol{a}_v)$. The cost of the target vertex $t = K$ is $f_t(\boldsymbol{x}_t) := \varphi(\boldsymbol{s}_t)$. All the other vertices (i.e., the ones

labelled by (k, i)) have zero cost. Also all the edges in the GCS have zero cost.

From the solution of this SPP in GCS we recover the optimal state and control sequences by looking at the optimal values of the variables $\boldsymbol{x}_v$ paired with the vertices $v = 0, \ldots, K$.

The graph in Figure 10-3 is acyclic, and the corresponding MICP is formulated as in Section 9.3. Only for analysis purposes, we report here the MICP that results from the application of our techniques. After multiple simplifications, we obtain

$$\begin{aligned}
\text{minimize} \quad & \sum_{k=0}^{K-1} \gamma(\boldsymbol{s}_k, \boldsymbol{a}_k) + \varphi(\boldsymbol{s}_K) && && \text{(10.6a)}\\
\text{subject to} \quad & y_{k,i} \in \{0, 1\}, && \forall k = 0, \ldots, K-1,\ i \in \mathcal{I}, && \text{(10.6b)}\\
& \boldsymbol{s}_0 = \hat{\boldsymbol{s}}, && && \text{(10.6c)}\\
& (\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1}, 1) = \sum_{i \in \mathcal{I}} (\boldsymbol{\sigma}_{k,i}, \boldsymbol{\alpha}_{k,i}, \boldsymbol{\sigma}^+_{k,i}, y_{k,i}), && \forall k = 0, \ldots, K-1, && \text{(10.6d)}\\
& \boldsymbol{\sigma}^+_{k,i} = \boldsymbol{A}_i \boldsymbol{\sigma}_{k,i} + \boldsymbol{B}_i \boldsymbol{\alpha}_{k,i} + \boldsymbol{c}_i y_{k,i}, && \forall k = 0, \ldots, K-1,\ i \in \mathcal{I}, && \text{(10.6e)}\\
& (\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i}, y_{k,i}) \in \tilde{\mathcal{D}}_i, && \forall k = 0, \ldots, K-1,\ i \in \mathcal{I}, && \text{(10.6f)}\\
& \boldsymbol{s}_K \in \mathcal{T}. && && \text{(10.6g)}
\end{aligned}$$

Here the binary variables $y_{k,i}$ are one if the system is in mode i at time k. The variables $\boldsymbol{s}_k$ and $\boldsymbol{a}_k$ are the system state and control at time k. The variables $(\boldsymbol{\sigma}_{k,i}, \boldsymbol{\alpha}_{k,i}, \boldsymbol{\sigma}^+_{k,i})$ are auxiliary variables that are equal to $(\boldsymbol{s}_k, \boldsymbol{a}_k, \boldsymbol{s}_{k+1})$ if $y_{k,i} = 1$, and are zero otherwise. This formulation is computationally light, but its convex relaxation can be quite loose. This MICP is identical to the one proposed in [129, Section 3.2], which, however, was hand-crafted and not generated algorithmically.

The convex relaxation of the MICP (10.6) can be easily tightened. Instead of enforcing the costs on the vertices $v = 0, \ldots, K$, we can enforce them on the edges that connect the vertices (k, i) and $k+1$ for all $k = 0, 1, \ldots, K-1$ and $i \in \mathcal{I}$. Specifically, for every edge $e = (v, w)$ that connects vertex $v = (k, i)$ to vertex $w = k + 1$, with $k = 0, \ldots, K-2$ and $i \in \mathcal{I}$, we define

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \gamma(\boldsymbol{s}_v, \boldsymbol{a}_v).$$

For the edges $e = (v, t)$ that connect vertex $v = (K-1, i)$ to the target $t = K$, with

$i \in \mathcal{I}$, we let

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_t) := \gamma(\boldsymbol{s}_v, \boldsymbol{a}_v) + \varphi(\boldsymbol{s}_t).$$

While for all the vertices $v \in \mathcal{V}$ we let $f_v(\boldsymbol{x}_v) = 0$.

This modification to the GCS has the effect of replacing the objective function (10.6a) with

$$\sum_{i \in \mathcal{I}} \left(\sum_{k=0}^{K-1} \tilde{\gamma}(\boldsymbol{\sigma}_{k,i}, \boldsymbol{\alpha}_{k,i}, y_{k,i}) + \tilde{\varphi}(\boldsymbol{\sigma}^+_{K-1,i}, y_{K-1,i}) \right). \tag{10.7}$$

For any feasible solution of the MICP (10.6), this alternative objective function is easily seen to yield the same value as the previous objective (10.6a).[5] On the other hand, when the binary variables $y_{k,i}$ are allowed to be fractional, the new objective can be greater than the original one, and therefore yields a tighter relaxation. To see this, note that, for any fixed time step k, the feasible solutions of the convex relaxation of (10.6) satisfy

$$\begin{aligned}
\sum_{i \in \mathcal{I}} \tilde{\gamma}(\boldsymbol{\sigma}_{k,i}, \boldsymbol{\alpha}_{k,i}, y_{k,i}) &= \sum_{i \in \mathcal{I}_{>0}} y_{k,i} \gamma(\boldsymbol{\sigma}_{k,i}/y_{k,i}, \boldsymbol{\alpha}_{k,i}/y_{k,i}) \\
&\geq \gamma \left(\sum_{i \in \mathcal{I}_{>0}} y_{k,i} (\boldsymbol{\sigma}_{k,i}/y_{k,i}, \boldsymbol{\alpha}_{k,i}/y_{k,i}) \right) \\
&= \gamma \left(\sum_{i \in \mathcal{I}_{>0}} (\boldsymbol{\sigma}_{k,i}, \boldsymbol{\alpha}_{k,i}) \right) \\
&= \gamma \left(\sum_{i \in \mathcal{I}} (\boldsymbol{\sigma}_{k,i}, \boldsymbol{\alpha}_{k,i}) \right) \\
&= \gamma(\boldsymbol{s}_k, \boldsymbol{a}_k),
\end{aligned}$$

where we defined $\mathcal{I}_{>0} := \{i \in \mathcal{I} : y_{k,i} > 0\}$, and where the inequality follows from the convexity of γ as in (2.10). Analogous steps show that the same inequality holds also for the terminal cost:

$$\sum_{i \in \mathcal{I}} \tilde{\varphi}(\boldsymbol{\sigma}^+_{K-1,i}, y_{K-1,i}) \geq \varphi(\boldsymbol{s}_K).$$

[5] If at time k we decide to be in mode $j \in \mathcal{I}$, then we have $y_{k,j} = 1$ and $y_{k,i} = 0$ for all $i \neq j$. This gives us $(\boldsymbol{\sigma}_{k,j}, \boldsymbol{\alpha}_{k,j}) = (\boldsymbol{s}_k, \boldsymbol{a}_k)$ and $(\boldsymbol{\sigma}_{k,i}, \boldsymbol{\alpha}_{k,i}) = (\boldsymbol{0}, \boldsymbol{0})$ for al $i \neq j$. In turn, this implies

$$\sum_{i \in \mathcal{I}} \tilde{\gamma}(\boldsymbol{s}_{k,i}, \boldsymbol{a}_{k,i}, y_{k,i}) = \tilde{\gamma}(\boldsymbol{s}_k, \boldsymbol{a}_k, 1) + \sum_{i \in \mathcal{I} \setminus \{j\}} \tilde{\gamma}(\boldsymbol{0}, \boldsymbol{0}, 0) = \gamma(\boldsymbol{s}_k, \boldsymbol{a}_k),$$

which shows the exactness of the alternative formulation of the stage cost in (10.7). The exactness of the alternative formulation of the terminal cost is shown in a similar way.

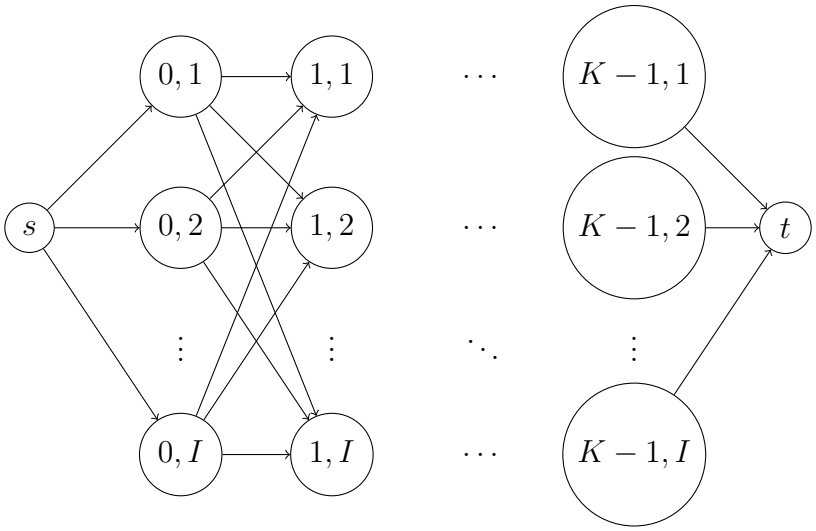

Figure 10-4: Strong and large formulation of the regulation problem for discrete-time PWA systems as an SPP in GCS.

The formulation with the modified objective (10.7) is essentially the same as the one proposed in [137].[6] Although it is provably stronger, it can often lead to a harder class of optimization problems. For example, if the stage and terminal costs are quadratic (and all the constraints polyhedral), then the formulation in (10.6) is a Mixed-Integer Quadratic Program (MIQP), while the one with the modified objective is a Mixed-Integer Second-Order-Cone Program (MISOCP). This can be a disadvantage since the solvers for MIQPs are typically significantly more effective than the ones for MISOCPs. The two formulations above have been benchmarked in [129]; we will expand on this benchmark in Section 10.2.6 below.

10.2.3 Strong but large formulations

We now illustrate a second class of mixed-integer formulations of problem (10.5). These will be larger than the previous ones (i.e., more variables and constraints) but they will also be stronger (i.e., they will have tighter convex relaxation).

We construct the GCS illustrated in Figure 10-4. The source s is the leftmost vertex and the target t is the rightmost. In between, we have K layers with I vertices each. The source is connected via an edge to each vertex in the first layer, and all the vertices in the last layer are connected to the target. Each pair of consecutive layers is fully connected.

As in the previous formulations, the continuous variable $\boldsymbol{x}_v$ paired with the vertex

[6] More precisely, in that work only the stage cost is reformulated as in (10.7), while the terminal cost is still enforced as in (10.6a).

$v = (k, i)$, for $k = 0, \dots, K-1$ and $i \in \mathcal{I}$, represents the state and control pair $(\boldsymbol{s}_k, \boldsymbol{a}_k)$ if at time k we are in mode i. The vertices are paired with the following sets:

$$\begin{aligned} &\mathcal{X}_s := \{\hat{\boldsymbol{s}}\}, \\ &\mathcal{X}_v := \mathcal{D}_i, && \forall v = (k, i) \in \{0, \dots, K-1\} \times \mathcal{I}, \\ &\mathcal{X}_t := \mathcal{T}. \end{aligned}$$

All the edges $e = (s, v)$ outgoing the source are paired with the convex set

$$\mathcal{X}_e := \{(\boldsymbol{x}_s, \boldsymbol{x}_v) : \boldsymbol{s}_s = \boldsymbol{s}_v\}.$$

These constraints enforce the initial conditions. The edges $e = (v, w)$ connecting vertex $v = (k, i)$ to either $w = (k+1, j)$ or $w = t$ are paired with the convex sets

$$\mathcal{X}_e := \{(\boldsymbol{x}_v, \boldsymbol{x}_w) : \boldsymbol{s}_w = \boldsymbol{A}_i \boldsymbol{s}_v + \boldsymbol{B}_i \boldsymbol{a}_v + \boldsymbol{c}_i\},$$

which enforce the system dynamics.

Similarly to the previous formulation, we have now the choice of enforcing the stage and terminal costs either on the vertices or on the edges. Also in this case enforcing them on the edges leads to a stronger formulation. In addition, this time this choice has not impact on the problem class. Therefore below we only consider one scenario, where the stage and terminal costs are distributed on the edges of the GCS. We set the cost of all the vertices to zero. For all the edges $e = (v, w)$ that connect a vertex $v = (k, i)$ to a vertex $w = (k+1, j)$, we let

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_w) := \gamma(\boldsymbol{s}_v, \boldsymbol{a}_v).$$

The cost of all the edges that connect a vertex $v = (K-1, i)$ to the target t is

$$f_e(\boldsymbol{x}_v, \boldsymbol{x}_t) := \gamma(\boldsymbol{s}_v, \boldsymbol{a}_v) + \varphi(\boldsymbol{s}_t).$$

The graph in Figure 10-4 is acyclic and has zero vertex costs. The corresponding MICP is formulated as in Section 9.3. For analysis purposes, the MICP that we

obtain in this case reads as follows. The objective function is

$$\sum_{k=0}^{K-2}\sum_{i\in\mathcal{I}}\sum_{j\in\mathcal{I}}\tilde{\gamma}(\boldsymbol{\sigma}_{k,i,j},\boldsymbol{\alpha}_{k,i,j},y_{k,i,j})+\sum_{i\in\mathcal{I}}\tilde{\gamma}(\boldsymbol{\sigma}_{K-1,i},\boldsymbol{\alpha}_{K-1,i},y_{K-1,i}) + \sum_{i\in\mathcal{I}}\tilde{\varphi}(\boldsymbol{A}_i\boldsymbol{\sigma}_{K-1,i}+\boldsymbol{B}_i\boldsymbol{\alpha}_{K-1,i}+\boldsymbol{c}_i y_{K-1,i},y_{K-1,i}). \quad (10.8)$$

The constraints are

$$\begin{aligned}
&y_{k,i}\in\{0,1\}, && \forall k=0,\dots,K-1,\ i\in\mathcal{I},\\
&y_{k,i,j}\in\{0,1\}, && \forall k=0,\dots,K-2,\ i,j\in\mathcal{I},\\
&\sum_{i\in\mathcal{I}} y_{0,i}=1 &&\\
&\boldsymbol{\sigma}_{0,i}=y_{0,i}\hat{\boldsymbol{s}}, && \forall i\in\mathcal{I},\\
&\sum_{j\in\mathcal{I}}(\boldsymbol{\sigma}_{k,i,j},\boldsymbol{\alpha}_{k,i,j},y_{k,i,j})=(\boldsymbol{\sigma}_{k,i},\boldsymbol{\alpha}_{k,i},y_{k,i}), && \forall k=0,\dots,K-2,\ i\in\mathcal{I},\\
&\sum_{i\in\mathcal{I}}(\boldsymbol{\sigma}^+_{k,i,j},\boldsymbol{\alpha}^+_{k,i,j},y_{k,i,j})=(\boldsymbol{\sigma}_{k+1,j},\boldsymbol{\alpha}_{k+1,j},y_{k+1,j}), && \forall k=0,\dots,K-2,\ j\in\mathcal{I},\\
&(\boldsymbol{\sigma}_{k,i,j},\boldsymbol{\alpha}_{k,i,j},y_{k,i,j})\in\tilde{\mathcal{D}}_i, && \forall k=0,\dots,K-2,\ i,j\in\mathcal{I},\\
&(\boldsymbol{\sigma}^+_{k,i,j},\boldsymbol{\alpha}^+_{k,i,j},y_{k,i,j})\in\tilde{\mathcal{D}}_j, && \forall k=0,\dots,K-2,\ i,j\in\mathcal{I},\\
&\boldsymbol{\sigma}^+_{k,i,j}=\boldsymbol{A}_i\boldsymbol{\sigma}_{k,i,j}+\boldsymbol{B}_i\boldsymbol{\alpha}_{k,i,j}+\boldsymbol{c}_i y_{k,i,j}, && \forall k=0,\dots,K-2,\ i,j\in\mathcal{I},\\
&(\boldsymbol{A}_i\boldsymbol{\sigma}_{K-1,i}+\boldsymbol{B}_i\boldsymbol{\alpha}_{K-1,i}+\boldsymbol{c}_i y_{K-1,i},y_{K-1,i})\in\tilde{\mathcal{T}}, && \forall i\in\mathcal{I}.
\end{aligned}$$

Here the binary variable $y_{k,i}$ is one if the system is in mode i at time k. While $y_{k,i,j}$ is one if a time k we transition from mode i to mode j. The vectors labelled with the same indices represent the system state and control if the corresponding binary variable is one.

10.2.4 Big-M formulation

The first mixed-integer formulation for optimal control of discrete-time PWA systems appeared in [12, Section 3.1]. That formulation was based on the so-called "big-M" method (and was intended primarily as a proof of concept rather than an efficient formulation). We report it here for completeness, since it is still frequently used and we will include it in the upcoming numerical experiments.

The big-M formulation assumes that the sets $\mathcal{D}_i$ are polytopes:

$$\mathcal{D}_i:=\{(\boldsymbol{s},\boldsymbol{a}):\boldsymbol{F}_i\boldsymbol{s}+\boldsymbol{G}_i\boldsymbol{a}\leq\boldsymbol{h}_i\},\qquad\forall i\in\mathcal{I}.$$

For all $i \in \mathcal{I}$, we define the big-M parameters

$$\begin{aligned}
\boldsymbol{l}_i &:= \max\{\boldsymbol{F}_i\boldsymbol{s} + \boldsymbol{G}_i\boldsymbol{a} - \boldsymbol{h}_i : (\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}_j,\ j \in \mathcal{I}\},\\
\boldsymbol{d}_i &:= \min\{\boldsymbol{A}_i\boldsymbol{s} + \boldsymbol{B}_i\boldsymbol{a} + \boldsymbol{c}_i : (\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}_j,\ j \in \mathcal{I}\},\\
\boldsymbol{e}_i &:= \max\{\boldsymbol{A}_i\boldsymbol{s} + \boldsymbol{B}_i\boldsymbol{a} + \boldsymbol{c}_i : (\boldsymbol{s}, \boldsymbol{a}) \in \mathcal{D}_j,\ j \in \mathcal{I}\},
\end{aligned}$$

where the minimum and the maximum are elementwise. These can be computed by solving a collection of linear programs. The constraints of the big-M MICP are

$$\begin{aligned}
&y_{k,i} \in \{0, 1\}, && \forall k = 0, \dots, K-1,\ i \in \mathcal{I}, && \text{(10.9a)}\\
&\boldsymbol{s}_0 = \hat{\boldsymbol{s}}, && && \text{(10.9b)}\\
&\boldsymbol{F}_i\boldsymbol{s}_k + \boldsymbol{G}_i\boldsymbol{a}_k \le \boldsymbol{h}_i + \boldsymbol{l}_i(1 - y_{k,i}), && \forall k = 0, \dots, K-1,\ i \in \mathcal{I}, && \text{(10.9c)}\\
&\sum_{i \in \mathcal{I}} y_{k,i} = 1, && \forall k = 0, \dots, K-1, && \text{(10.9d)}\\
&\boldsymbol{s}_{k+1} = \sum_{i \in \mathcal{I}} \boldsymbol{\sigma}^+_{k,i}, && \forall k = 0, \dots, K-1, && \text{(10.9e)}\\
&\boldsymbol{\sigma}^+_{k,i} \le \boldsymbol{A}_i\boldsymbol{s}_k + \boldsymbol{B}_i\boldsymbol{a}_k + \boldsymbol{c}_i - \boldsymbol{d}_i(1 - y_{k,i}), && \forall k = 0, \dots, K-1,\ i \in \mathcal{I}, && \text{(10.9f)}\\
&\boldsymbol{\sigma}^+_{k,i} \ge \boldsymbol{A}_i\boldsymbol{s}_k + \boldsymbol{B}_i\boldsymbol{a}_k + \boldsymbol{c}_i - \boldsymbol{e}_i(1 - y_{k,i}), && \forall k = 0, \dots, K-1,\ i \in \mathcal{I}, && \text{(10.9g)}\\
&\boldsymbol{\sigma}^+_{k,i} \le \boldsymbol{e}_i y_{k,i}, && \forall k = 0, \dots, K-1,\ i \in \mathcal{I}, && \text{(10.9h)}\\
&\boldsymbol{\sigma}^+_{k,i} \ge \boldsymbol{d}_i y_{k,i}, && \forall k = 0, \dots, K-1,\ i \in \mathcal{I}, && \text{(10.9i)}\\
&\boldsymbol{s}_K \in \mathcal{T}. && && \text{(10.9j)}
\end{aligned}$$

The objective function is equal to (10.5a).

This formulation can be formally shown to have weaker convex relaxation than the ones above. On the other hand, it has small number of variables and constraints, therefore in some cases its runtime can be competitive with ours. Below we compare all the formulations discussed in this chapter on two challenging numerical examples.

10.2.5 Numerical example: footstep planning

We consider the problem shown in Figure 10-5, where we optimize the sequence of footsteps necessary for a walking robot to traverse a set of stepping stones. The dynamics of the robot is approximated as a double integrator with position $\boldsymbol{q} \in \mathbb{R}^2$, velocity $\boldsymbol{v} \in \mathbb{R}^2$, and force $\boldsymbol{a} \in \mathbb{R}^2$. The equations governing the motion of the robot are then $\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \boldsymbol{v}_k$ and $\boldsymbol{v}_{k+1} = \boldsymbol{v}_k + \eta\boldsymbol{a}_k$, where η is a scalar parameter that regulates the system controllability. The system state at time k is $\boldsymbol{s}_k := (\boldsymbol{q}_k, \boldsymbol{v}_k)$. The initial position is $\hat{\boldsymbol{q}} := (-3.5, 0.5)$ (green plus in Figure 10-5), the initial velocity

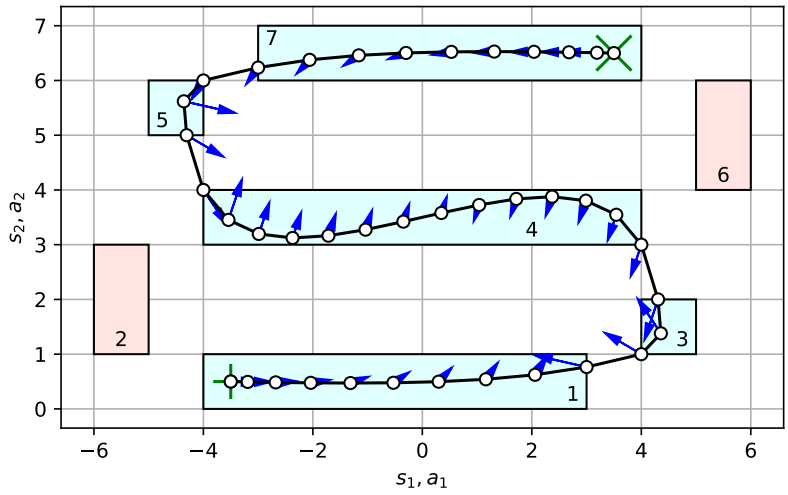

Figure 10-5: Footstep-planning problem. A simplified robot has to navigate from the start (green plus) to the goal (green cross). At each time step, the robot configuration needs to lie in one of the stepping stones. The light-blue and red stepping stones have high and low controllability, respectively. The optimal trajectory is depicted with white circles and the optimal controls as blue arrows.

is $\hat{\boldsymbol{v}} := \mathbf{0}$. The initial state is then $\hat{\boldsymbol{s}} := (-3.5, 0.5, 0, 0)$. At each time step $k = 1, \ldots, K-1$, the position $\boldsymbol{q}_k$ must belong to one of the seven stepping stones in Figure 10-5, while the velocity and the controls are limited by the constraints $\|\boldsymbol{v}_k\|_\infty \leq 1$ and $\|\boldsymbol{a}_k\|_\infty \leq 1$. The goal is to reach the point $\boldsymbol{q}_K := (3.5, 6.5)$ (green cross in Figure 10-5) with zero velocity $\boldsymbol{v}_K$ in $K := 40$ time steps. Therefore we let $\mathcal{T} := \{(3.5, 6.5, 0, 0)\}$. The stage cost is $\gamma(\boldsymbol{s}_k, \boldsymbol{a}_k) := \|\boldsymbol{v}_k\|_2^2/5 + \|\boldsymbol{a}_k\|_2^2$, and the terminal cost is zero.

We let the parameter η vary between the seven stepping stones. The five stones in the range $-5 \leq q_1 \leq 5$ (light blue in Figure 10-5) have $\eta = 1$. While in the other two stones (red in Figure 10-5) we make the system more expensive to control by setting $\eta = 0.1$. These can be thought as slippery stones where it is undesirable to place the foot of the robot. Since the parameter η varies with the state, the system dynamics is PWA and the control problem falls into the class considered above. The top panel in Figure 10-5 shows the optimal trajectory $\boldsymbol{q}_0, \ldots, \boldsymbol{q}_K$ as white circles, and the corresponding optimal controls $\boldsymbol{a}_0, \ldots, \boldsymbol{a}_{K-1}$ as blue arrows.

We compare the performance of the four mixed-integer formulations discussed in this chapter. In order of increasing strength:

- The big-M formulation (10.9), which first appeared in [12, Section 3.1].
- The formulation (10.6), proposed in [129], and recoverable as a special case of

	Big-M (10.9)	Form. (10.6)	Form. (10.7)	Form. (10.8)
Problem class	**MIQP**	**MIQP**	MISOCP	MISOCP
Relaxation cost	0	0.44	0.48	**5.49**
MICP cost	7.19	7.19	7.19	7.19
Relaxation gap	100%	94%	93%	**24%**
Binary var.	**280**	**280**	**280**	2191
Continuous var.	**1364**	3044	3324	26774
Linear constr.	8048	**4928**	**4928**	57631
Conic constr.	**0**	**0**	280	1918

Table 10.1: Problem class, cost statistics, and program size for the footstep-planning problem using the various mixed-integer formulations described in this chapter.

the techniques introduced in this thesis.

- The formulation (10.7), proposed in [137], which is also a special case of the techniques introduced in this thesis.

- The formulation (10.8), which is a new formulation obtained algorithmically through the application of our techniques.

The computer used in this (as well as the next) comparison is a laptop with processor 2.4 GHz 8-Core Intel Core i9 and memory 64 GB 2667 MHz DDR4.

We start in Table 10.1 where, for each formulation, we report: the class of mixed-integer program, the cost of the relaxation compared to the cost of the MICP, and the size of the optimization problems. (Recall that the relaxation gap is defined as the difference between the optimal value of the MICP and the one of its relaxation, normalized by the MICP optimal value.) The simple big-M formulation (10.9) has the weakest relaxation, which gives a vacuous lower bound of zero on the MICP cost (100% relaxation gap). The two relaxations from Section 10.2.2 have similar cost, both very loose (94% and 93% relaxation gap). The formulation (10.8) is much larger than the others (one order of magnitude more variables and constraints) but yields a significantly lower relaxation gap, only 24%.

Table 10.2 shows how the size and the strength of the various formulations are reflected on the solver time. We consider the solver `Gurobi 10.0` with default options. The fastest formulation is the strongest and largest one (10.8), and takes 13.5s. The big-M formulation (10.9) is surprisingly fast, given its weakness and takes 19.1s. The formulations (10.6) and (10.7) have very similar size and strength, but the second takes a dramatically larger time to solve. This is mostly due to the fact that the second problem is an MISOCP, while the first is an MIQP.

	Big-M (10.9)	Form. (10.6)	Form. (10.7)	Form. (10.8)
Relaxation time (s)	0.039	**0.036**	0.080	1.49
MICP time (s)	19.1	26.5	547	**13.5**

Table 10.2: Runtime statistics for the solution of the footstep-planning problem using `Gurobi 10.0`.

	Big-M (10.9)	Form. (10.6)	Form. (10.7)	Form. (10.8)
Relaxation time (s)	**0.048**	0.075	0.21	2.58
MICP time (s)	3600 (TL)	3600 (TL)	3600 (TL)	**15.7**
Cost upper bound	∞	7.25	7.53	**7.19**
Cost lower bound	0.00	1.44	2.37	**7.19**
Gap at TL	100%	80.2%	68.5%	**0%**

Table 10.3: Runtime and cost statistics for the solution of the footstep-planning problem using `MOSEK 10.0`. The acronym TL stands for time limit (set to one hour). Cost upper bound is the cost of the best feasible solution found during BB. Cost lower bound is the lower bound on the optimal value provided by the BB algorithm. Gap at TL is the percentage difference of the these two costs.

Mixed-integer solvers use many heuristics, and it is hard to draw conclusions only by comparing the performance of one solver. For completeness, in Table 10.3 we report the runtimes of the solver `MOSEK 10.0` (with default options). In this case the comparison is much more in favor of the strongest formulation (10.8). This yields a runtime of only 15.7s, while none of the other formulations can solve the problem within a time limit of one hour. For the formulations that could not solve the problem, Table 10.3 shows the cost of the best feasible solution ("cost upper bound") and the lower bound provided by the Branch and Bound (BB) solver after one hour of computations. The big-M formulation (10.9) does not yield a feasible solution within one our of BB, and the lower bound on the optimal value is still zero at the termination. The other two formulations perform slightly better, but still have a gap of 80.2% and 68.5% at the time limit. Although using the solver `Gurobi` the four formulations performed comparably, we observe that with `MOSEK` the strength of the formulation has a much crucial impact on the solve times.

We continue this comparison with a pictorial illustration of the tightness of the convex relaxations of the various formulations. In the top panel of Figure 10-6 we show the optimal sequence of stepping stones for the footstep-planning problem. For each time step $k = 0, \dots, K-1$ we depict in blue the optimal value of i_k (i.e., the stepping stone in which the system configuration $\boldsymbol{q}_k$ lies). The stones are numbered as they are labelled in Figure 10-5. The second to last panels show the values of the binary variables $y_{k,i}$ for $k = 0, \dots, K-1$ and $i \in \mathcal{I}$, obtained by solving the

convex relaxations of the various MICPs. In the figure, the fractional value of $y_{k,i}$ is represented by the opacity of the point corresponding to time step k and stepping stone i. The point is fully blue if $y_{k,i} = 1$ and is white if $y_{k,i} = 0$. The formulations in Figure 10-6 are ordered as follows: the second panel is the big-M formulation (10.9), the third is the formulation (10.6), the fourth is the formulation (10.7), and the last is the formulation (10.8). For the last formulation we also illustrate the optimal values of the variables $y_{k,i,j}$, represented by the opacity of the lines connecting the points in the grid.

As it can be seen the largest formulation (10.8) is the one that approximates most closely the optimal value of the binary variables $y_{k,i}$. It guesses correctly the optimal sequence of stepping stones, but it is not sure about the exact times at which it should transition from one stepping stone to the next. The other relaxations contain almost no information about the optimal solution.

Remark 10.2. In this and the following comparison we do not report the performance of the rounding strategy from Section 9.5 since rounding is quite ineffective for the problems analyzed in this chapter. Specifically, for optimal-control problems like the one considered here, only a tiny fraction of the paths through the GCS yields a feasible convex restriction 9.9. Therefore our rounding strategy is often unable to find a feasible solution of the SPP in GCS.

10.2.6 Numerical example: ball and paddle

As a second benchmark we consider the problem illustrated in Figure 10-7, of flipping a ball through the use of a paddle. The control input $\boldsymbol{a} \in \mathbb{R}^2$ is the translational acceleration of the paddle (which cannot rotate). The state $\boldsymbol{s} \in \mathbb{R}^{10}$ of the system is composed by: s_1 and s_2 position of the ball, s_3 angle of the ball, s_4 and s_5 position of the paddle, and their time derivatives. The task is to rotate the ball from the initial state $\hat{s}_3 = \pi$, $\hat{s}_k = 0$ for $k \neq 3$ to the origin. The motion of the ball is governed by the gravity and the contact forces that arise from the interactions with the paddle and the ceiling. The dynamics is discretized using the semi-implicit Euler method,

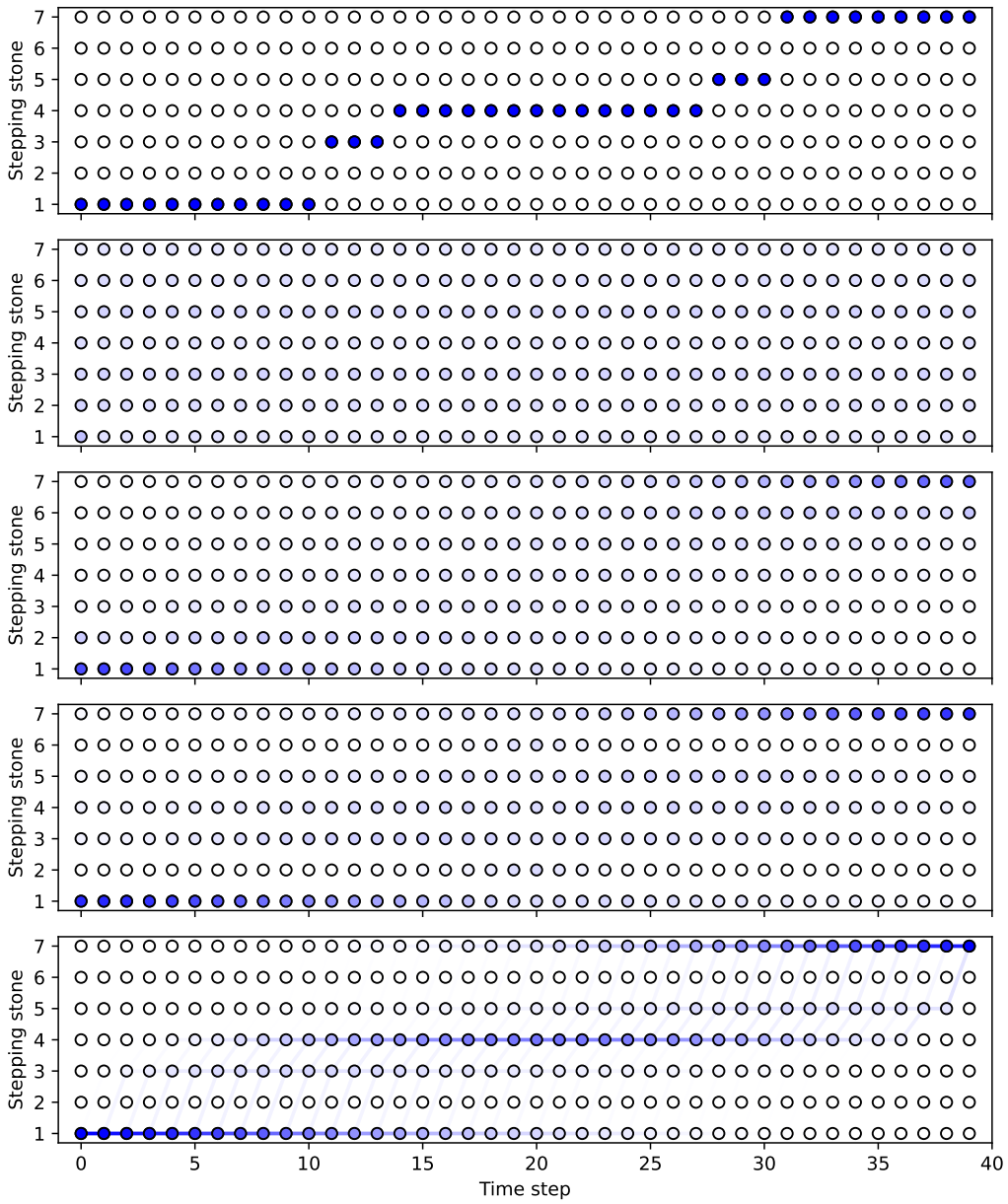

Figure 10-6: *Top:* optimal sequence of stepping stones for the footstep-planning problem. *Other panels:* optimal value of the relaxed binary variables for the big-M formulation (10.9), formulation (10.6), formulation (10.7), and formulation (10.8).

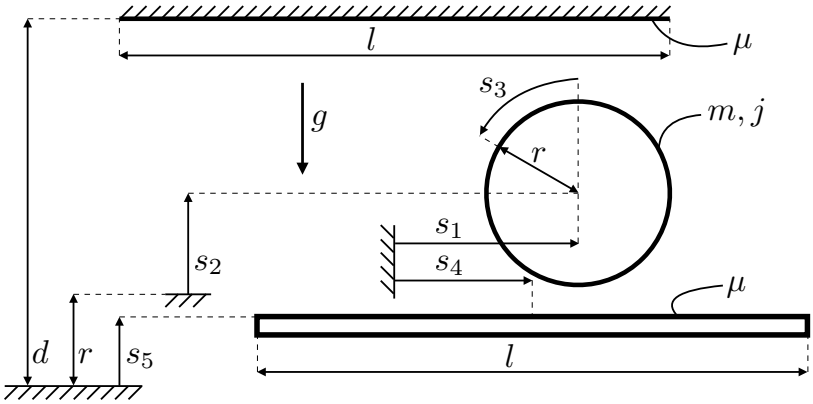

Figure 10-7: Benchmark problem of rotating a two-dimensional ball by 180 degrees. The ball is subject to the gravity force and can collide with both the paddle (bottom) and the ceiling (top), the control input is the translational acceleration of the paddle.

resulting in

$$s_\nu^+ = s_\nu + h s_{\nu+5}^+, \qquad \forall \nu = 1, \ldots, 5, \tag{10.10a}$$
$$s_6^+ = s_6 + h(\lambda_{\mathrm{pt}} - \lambda_{\mathrm{ct}})/m, \tag{10.10b}$$
$$s_7^+ = s_7 + h(\lambda_{\mathrm{pn}} - \lambda_{\mathrm{cn}} - mg)/m, \tag{10.10c}$$
$$s_8^+ = s_8 + hr(\lambda_{\mathrm{pt}} + \lambda_{\mathrm{ct}})/j, \tag{10.10d}$$
$$s_9^+ = s_9 + h a_1, \tag{10.10e}$$
$$s_{10}^+ = s_{10} + h a_2. \tag{10.10f}$$

Here the plus superscript denotes the state value at the next time step. The parameters $r = 0.1$, $m = 1$, $j = \frac{2}{5}mr^2$ are the radius, the mass, and the moment of inertia of the ball, $g = 10$ is the gravity acceleration, $h = 0.05$ is the discretization step, $d = 0.4$ and $l = 0.6$ are geometric parameters (see Figure 10-7). The letter λ represents a contact force, with the subscripts p and c denoting the paddle and the ceiling, and t and n denoting the tangential and the normal components.

Contact phenomena are modeled with the inelastic time-stepping scheme from [175], where the complementarity conditions are solved ahead of time to express the contact forces as explicit PWA functions of the state and the inputs. Friction coefficients are set to $\mu = 0.2$. Overall we get a PWA system with $I = 7$ modes: no contact, ball in contact with the paddle (rolling, sliding left or right), and ball in contact with the ceiling (rolling, sliding left or right). Together with the state limits defined by the

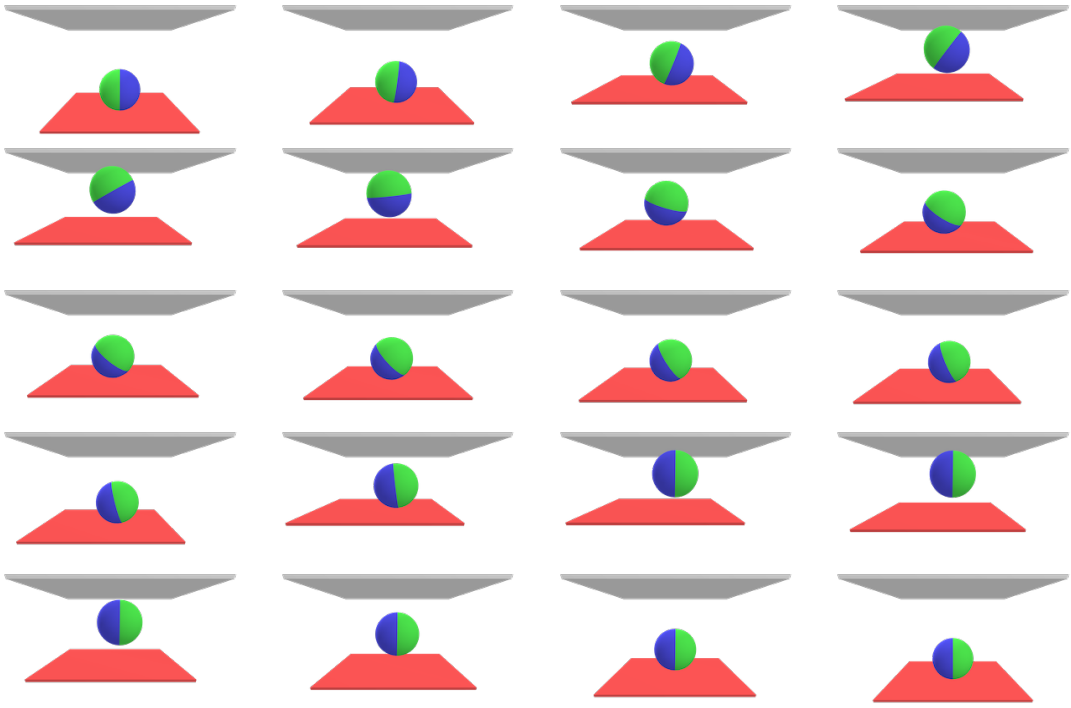

Figure 10-8: Snapshots of the optimal solution of the ball-and-paddle problem. The paddle moves the ball, which makes and breaks contact with the ceiling twice in order to rotated by 180 degrees.

contact constraints, we enforce the constraints

$$\begin{aligned} -\bar{\boldsymbol{s}} \leq \boldsymbol{s} \leq \bar{\boldsymbol{s}}, \qquad & \bar{\boldsymbol{s}} := (0.3, 0.2, 1.2\pi, 0.3, 0.15, 2, 2, 10, 2, 2), \\ -\bar{\boldsymbol{a}} \leq \boldsymbol{a} \leq \bar{\boldsymbol{a}}, \qquad & \bar{\boldsymbol{a}} := (30, 30). \end{aligned}$$

In addition, since the paddle and the ceiling have limited width, in case at the next time step the ball is going to hit, e.g., the paddle, we require $|s_1^+ - s_4^+| \leq l/2$. The terminal set $\mathcal{T}$ forces the system to be in the origin after $K = 20$ time steps. We set a quadratic stage cost $\gamma(\boldsymbol{s}_k, \boldsymbol{a}_k) := \boldsymbol{s}_k^\top \boldsymbol{Q} \boldsymbol{s}_k + \boldsymbol{a}_k^\top \boldsymbol{R} \boldsymbol{a}_k$ with

$$\begin{aligned} \boldsymbol{Q} &:= \operatorname{diag}(1, 1, 0.01, 1, 1, 1, 1, 0.01, 1, 1), \\ \boldsymbol{R} &:= \operatorname{diag}(0.01, 0.001), \end{aligned}$$

where the function diag stacks its argument in a diagonal matrix. The terminal cost γ is zero. Figure 10-8 shows a sequence of snapshots of the optimal solution of the problem.

Despite the simplicity of the dynamical system, this control problem is quite challenging. Analyzing the dynamics (10.10) it can be noticed that the presence of the

	Big-M (10.9)	Form. (10.6)	Form. (10.7)	Form. (10.8)
Problem class	**MIQP**	**MIQP**	MISOCP	MISOCP
Relaxation cost	0.099	2.72	4.79	**9.94**
MICP cost	37.2	37.2	37.2	37.2
Relaxation gap	99.7%	92.7%	87.1%	**73.3%**
Binary var.	**140**	**140**	**140**	1071
Continuous var.	**1650**	3330	3470	25212
Linear constr.	9880	**5920**	**5920**	66891
Conic constr.	**0**	**0**	140	938

Table 10.4: Problem class, cost statistics, and program size for the ball-and-paddle problem.

ceiling is essential to accomplish the rotation task. Indeed, as long as λ_{ct} is zero, the horizontal and the angular dynamics of the ball cannot be controlled independently, even assuming to have direct control on λ_{pt} (see (10.10b) and (10.10d)). Therefore, we cannot both rotate the ball and place it back at the center of the paddle. This lack of controllability makes it very complex to detect an initial feasible trajectory, and prevents many branches from being cut in the early stages of the BB algorithm. In this sense, we expect that the convex relaxation plays here a fundamental role in guiding the growth of the BB tree.

As in the previous comparison, we start by analyzing the relaxation gaps and the size of the multiple formulations. The strongest formulation (10.8) yields the smallest, but quite large, relaxation gap: 73.3%. The other relaxations are significantly smaller but even weaker. The big-M relaxation (10.6) provides also in this case a vacuous lower bound of almost zero.

In Table 10.5 we report the solution statistics for the solver `Gurobi 10.0`. In this case the formulation (10.6) is significantly more effective than all the others. It is the only one that allows us to solve the problem within a time limit of one hour. After the time limit, the big-M formulation has still a gap of 98.3% and the formulation (10.7) has 62.7%. The strongest formulation (10.8) leads to algorithmic issues with `Gurobi`.[7]

In Table 10.6 we report the statistics `MOSEK 10.0`. In this case the BB does not converge with any of the formulation. After the time limit of one hour, the formulation that gives us the best feasible solution is (10.8), which led to a failure with `Gurobi`. This solution is not far from the global minimum (the suboptimality

[7]For this instance, `Gurobi` is not able to solve the root-node problem, and does not make any progress in the BB phase. This problem persists even if we select different methods to solve the root-node problem (e.g., primal simplex, dual simplex, or barrier). Strangely, however, `Gurobi` is able to solve the convex relaxation of the MICP in a couple of seconds, when asked to do so outside the BB algorithm.

	Big-M (10.9)	Form. (10.6)	Form. (10.7)	Form. (10.8)
Relaxation time (s)	0.070	**0.064**	0.099	2.40
MICP time (s)	3600 (TL)	**1351**	3600 (TL)	Fail
Cost upper bound	43.5	**37.2**	38.6	Fail
Cost lower bound	0.76	**37.2**	14.4	Fail
Gap at TL	98.3%	**0%**	62.7%	Fail

Table 10.5: Runtime and cost statistics for the ball-and-paddle problem using `Gurobi 10.0`.

	Big-M (10.9)	Form. (10.6)	Form. (10.7)	Form. (10.8)
Relaxation time (s)	**0.093**	0.118	0.532	2.84
MICP time (s)	3600 (TL)	3600 (TL)	3600 (TL)	3600 (TL)
Cost upper bound	∞	45.2	49.1	**41.2**
Cost lower bound	0.93	8.11	11.1	**12.6**
Gap at TL	100%	82.1%	77.4%	**69.4%**

Table 10.6: Runtime and cost statistics for the ball-and-paddle problem using `MOSEK 10.0`.

is approximately 10%) but `MOSEK` can only guarantee that the suboptimality gap is not larger than 69.4%. The big-M formulation does not identify a feasible solution within the time limit, and the other formulations perform comparable to (10.8).

Also in this experiments we then observe that the performance of the various formulations is very dependent on the choice of the solver. As for the footstep-planning problem, `MOSEK` performs best with a large but strong formulation. Conversely, with `Guorbi` the formulation strength does not seem to play and equally important role.

We conclude this comparison by illustrating in Figure 10-9 the optimal mode sequences for the MICPs and the relaxations. Also in this case, and as expected, the strongest formulation (10.8) provides us with the most accurate guess on the optimal value of the binary variables $y_{k,i}$. However, it does not come as close as in the stepping-stone problem above.

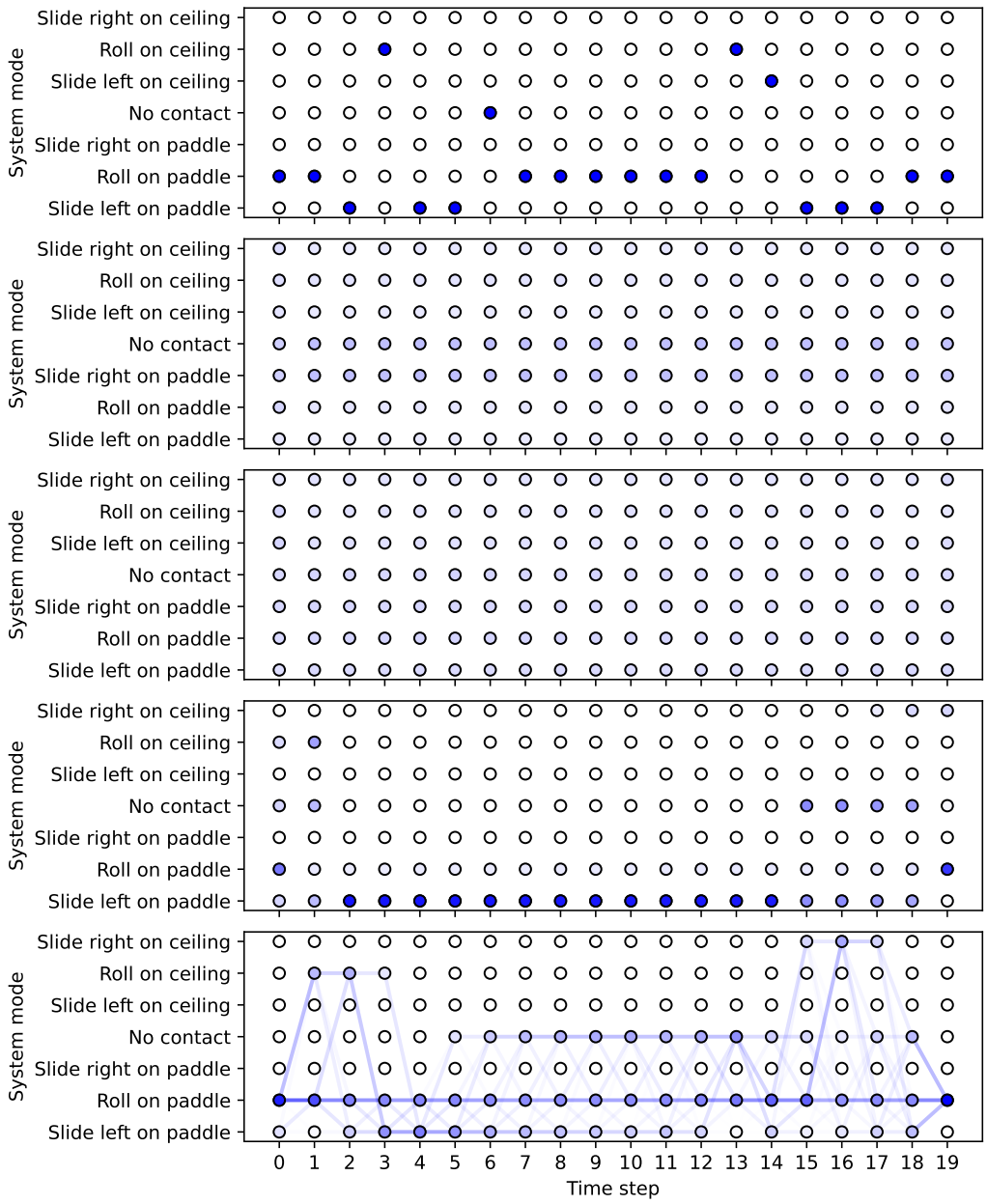

Figure 10-9: *Top:* optimal contact sequence for the ball-and-paddle problem. *Other panels:* optimal value of the relaxed binary variables for the big-M formulation (10.9), formulation (10.6), formulation (10.7), and formulation (10.8).

Chapter 11

Applications in motion planning

We apply the methods introduced in this thesis to solve motion-planning problems for autonomous robots that move thorough complex environments. Specifically, we consider the problem of designing a trajectory through a collection of safe convex sets, i.e., convex regions of the robot environment that do not intersect with obstacles. This problem is modelled as a Shortest-Path Problem (SPP) in Graphs of Convex Sets (GCS), and solved using the optimization techniques introduced in the previous chapters. We will focus on motion-planning problems where a basic preprocessing of the convex sets (e.g., finding their pairwise intersections) can be done offline. At runtime we quickly generate a trajectory that meets desired boundary conditions.

The material in this chapter is based on [128], and part of the presentation follows [127]. In Section 11.1 we will state the motion-planning problem in full generality. This problem will not lend itself to a direct transcription as an SPP in GCS, but will contain all the components that we might want our motion-planning method to handle. In the subsequent sections we will discuss multiple ways in which this problem can be simplified to make it efficiently solvable as an SPP in GCS.

11.1 Problem statement

We consider the design of a smooth trajectory in $\mathbb{R}^d$. A trajectory is represented as the function $\boldsymbol{q} : [0, T] \to \mathbb{R}^d$, where T is the time taken to traverse the trajectory. We considers trajectories with K continuous derivatives, and denote with $\boldsymbol{q}^{(k)}$ the kth derivative of $\boldsymbol{q}$ for $k = 0, \ldots, K$. The zeroth derivative is the trajectory itself, $\boldsymbol{q}^{(0)} := \boldsymbol{q}$.

We seek a trajectory that meets given boundary conditions

$$\begin{aligned} \boldsymbol{q}^{(k)}(0) &= \boldsymbol{q}_0^k, && \forall k = 0, \dots, K, \\ \boldsymbol{q}^{(k)}(T) &= \boldsymbol{q}_T^k, && \forall k = 0, \dots, K, \end{aligned}$$

where $\boldsymbol{q}_0^k, \boldsymbol{q}_T^k \in \mathbb{R}^d$ are the initial and final values of the kth derivative. At each time we require that the trajectory stay in a given set $\mathcal{S} \subset \mathbb{R}^d$ of safe points:

$$\boldsymbol{q}(\tau) \in \mathcal{S}, \qquad \forall \tau \in [0, T].$$

We assume that the safe set $\mathcal{S}$ is the union of a finite family of compact convex sets,

$$\mathcal{S} := \bigcup_{i \in \mathcal{I}} \mathcal{C}_i.$$

(Note that in the literature we can find a variety of practical methods to decompose complex spaces into convex sets, possibly approximately [118, 8, 80, 46, 192]. There also exist decomposition algorithms tailored to the configuration spaces of kinematic trees and articulated robots [3, 188, 44, 152].)

For $k = 1, \dots, K$, the kth derivative of our trajectory is constrained in a closed convex set $\mathcal{C}^k$ at all times:

$$\boldsymbol{q}^{(k)}(\tau) \in \mathcal{C}^k, \qquad \forall \tau \in [0, T].$$

We allow the final time T to be optimized, and we set the following bounds on its value:

$$T_{\min} \leq T \leq T_{\max}$$

where $T_{\max} \geq T_{\min} \geq 0$.

Our objective function features two components: a cost on the trajectory duration and a penalty on the magnitude of the trajectory derivatives. In formulas,

$$\psi(\boldsymbol{q}, T) := \varphi_0(T) + \sum_{k=1}^{K} \int_0^T \varphi_k(\boldsymbol{q}^{(k)}(\tau)) d\tau, \tag{11.1}$$

where the cost functions φ_k, $k = 0, \dots, K$, are convex.

Overall, our motion-planning problem is stated as

$$\begin{aligned}
\text{minimize} \quad & \psi(\boldsymbol{q}, T) && && \text{(11.2a)}\\
\text{subject to} \quad & \boldsymbol{q}^{(k)}(0) = \boldsymbol{q}_0^k, && \forall k = 0, \dots, K, && \text{(11.2b)}\\
& \boldsymbol{q}^{(k)}(T) = \boldsymbol{q}_T^k, && \forall k = 0, \dots, K, && \text{(11.2c)}\\
& \boldsymbol{q}(\tau) \in \mathcal{S}, && \forall \tau \in [0, T], && \text{(11.2d)}\\
& \boldsymbol{q}^{(k)}(\tau) \in \mathcal{C}^k, && \forall \tau \in [0, T],\ k = 1, \dots, K, && \text{(11.2e)}\\
& T_{\min} \le T \le T_{\max}. && && \text{(11.2f)}
\end{aligned}$$

The optimization variables are the function $\boldsymbol{q}$ and the final time T. The former is infinite dimensional, but we will restrict candidate trajectories to curves parameterized by a finite set of points (e.g., composite Bézier curves).

Below we describe how the motion planning problem (11.2) can be tackled using the techniques introduced in this thesis. As anticipated, we will not be able to reduce problem (11.2) to an SPP in GCS exactly, since the simultaneous optimization of the trajectory shape and timing leads to nonconvex constraints that do not immediately fit in out framework. In the next section, we consider a simplified problem that is easily reformulated as an SPP in GCS exactly. This will illustrate the basic idea behind our approach and will set the basis for the techniques in the upcoming sections, which will attempt to solve problem (11.2) in its completeness.

11.2 Minimum-length trajectories

We start by considering a simplified version of problem (11.2), where we seek a trajectory $\boldsymbol{q}$ of minimum Euclidean length:

$$\ell(\boldsymbol{q}) := \int_0^T \|\boldsymbol{q}^{(1)}(\tau)\|_2 d\tau.$$

Note that this objective is a special case of (11.1), where φ_1 is the $\mathcal{L}_2$ norm and φ_k is zero for all $k \neq 1$.

In the constraints of problem (11.2), we let the final time be free, i.e., $T_{\min} := 0$ and $T_{\max} := \infty$. We drop all the derivative constraints, i.e., $\mathcal{C}^k := \mathbb{R}^d$ for all $k = 1, \dots, K$. We also set all the boundary values of the trajectory derivatives to zero, so that the corresponding constraints become redundant.

Overall, our motion-planning problem is reduced to:

$$\begin{aligned} \text{minimize} \quad & \ell(\boldsymbol{q}) && && \text{(11.3a)} \\ \text{subject to} \quad & \boldsymbol{q}(0) = \boldsymbol{q}_0^0, && && \text{(11.3b)} \\ & \boldsymbol{q}(T) = \boldsymbol{q}_T^0, && && \text{(11.3c)} \\ & \boldsymbol{q}(\tau) \in \mathcal{S}, && \forall \tau \in [0, T]. && \text{(11.3d)} \end{aligned}$$

We observe that the optimal solution of this problem is always a polygonal (piecewise linear) trajectory.

Problem (11.3) can be reformulated as an SPP in GCS exactly. To do so, we need to define a graph $G = (\mathcal{V}, \mathcal{E})$, and assign convex constraint sets $\mathcal{X}_v$ and $\mathcal{X}_e$ and convex cost functions f_v and f_e to each vertex v and edge e. Below we describe each of these problem components; a visual illustration is given in Figure 11-1. At a high level, the plan is to construct a graph G whose paths represent different sequences of safe regions $\mathcal{C}_i$ that the robot can cross to reach the goal. Each safe region is then paired with a trajectory segment $\boldsymbol{q}_i$, and the GCS framework is used to couple the selection of a path with adequate costs and continuity constraints on the joint shape of the segments $\boldsymbol{q}_i$.

11.2.1 The graph

We let the graph $G = (\mathcal{V}, \mathcal{E})$ be the **intersection graph** of the convex regions $\mathcal{C}_i$ that compose the free space $\mathcal{S}$. This is illustrated in the top-left, top-right, and center-left panels of Figure 11-1. Each safe region $\mathcal{C}_i$ is paired with a vertex $i \in \mathcal{V}$ and connected by an edge (i, j) to all the regions $\mathcal{C}_j$ with $j \in \mathcal{I}$ that intersect with it. The initial configuration $\boldsymbol{q}_0^0$ is represented by the source vertex s, and is connected by an edge (s, i) to each region $\mathcal{C}_i$ that contains it. Similarly, the final configuration $\boldsymbol{q}_T^0$ is paired with the target vertex t and connected to each region that contains it via an edge (i, t). Overall, our graph has vertices $\mathcal{V} := \mathcal{I} \cup \{s, t\}$ and edges

$$\begin{aligned} \mathcal{E} := \{(i,j) \in \mathcal{I}^2 : \mathcal{C}_i \cap \mathcal{C}_j \neq \emptyset,\ i \neq j\} \cup \{(s,i) : \boldsymbol{q}_0^0 \in \mathcal{C}_i,\ i \in \mathcal{I}\} \\ \cup \{(i,t) : \boldsymbol{q}_T^0 \in \mathcal{C}_i,\ i \in \mathcal{I}\}. \end{aligned}$$

Note that an s-t path in this graph identifies a sequence of safe regions that connect the start and goal configurations.

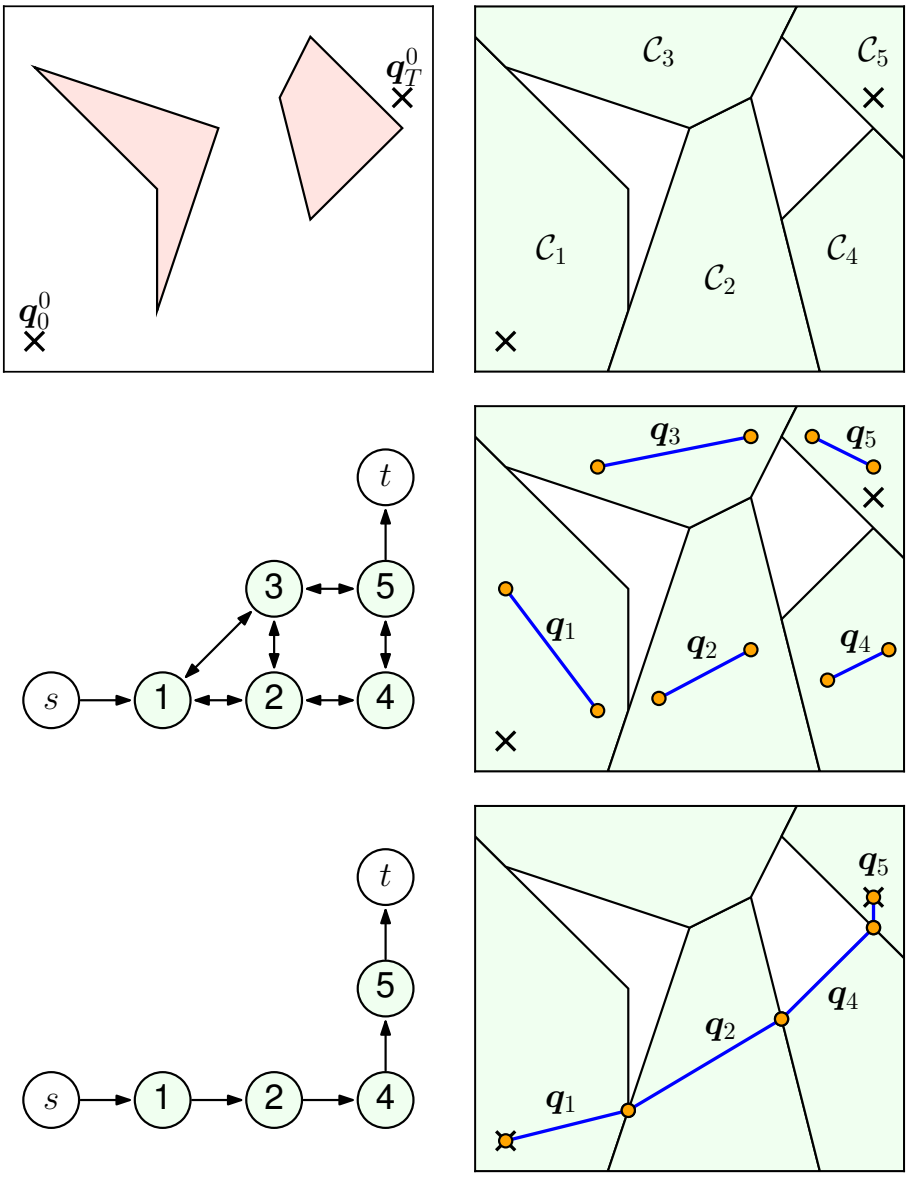

Figure 11-1: Optimization of a minimum-length trajectory as an SPP in GCS. *Top left:* environment with two obstacles in red, and with initial and final configurations. *Top right:* decomposition of the safe set into convex regions. *Center left:* intersection graph of the decomposition, with the vertices s and t representing the initial and final configurations. *Center right:* a trajectory segment is assigned to each convex region. *Bottom:* traversing a path in the intersection graph activates adequate costs and continuity constraints on the joint shape of the corresponding trajectory segments.

11.2.2 The convex constraint sets

As observed, the optimal solution of problem (11.3) is a polygonal trajectory, and there is no loss in assuming that our robot moves in a straight line $\boldsymbol{q}_i$ when traversing a region $\mathcal{C}_i$. We then pair each vertex $i \in \mathcal{I}$ with the convex constraint set $\mathcal{X}_i := \mathcal{C}_i \times \mathcal{C}_i$. This set contains the point $\boldsymbol{x}_i := (\boldsymbol{q}_{i,0}, \boldsymbol{q}_{i,1})$. The variables $\boldsymbol{q}_{i,0} \in \mathbb{R}^d$ and $\boldsymbol{q}_{i,1} \in \mathbb{R}^d$ represent the initial and final points of the line segment $\boldsymbol{q}_i$, respectively. (See center-right panel in Figure 11-1). Because of the convexity of $\mathcal{C}_i$, if both $\boldsymbol{q}_{i,0}$ and $\boldsymbol{q}_{i,1}$ lie in $\mathcal{C}_i$ then so does the whole segment $\boldsymbol{q}_i$, and our trajectory is collision free.

For all the edges $e = (i, j) \in \mathcal{E} \cap \mathcal{I}^2$, we let the constraint set $\mathcal{X}_e$ enforce the continuity of our trajectory through the equality $\boldsymbol{q}_{i,1} = \boldsymbol{q}_{j,0}$. This equality forces the final point of $\boldsymbol{q}_i$ to coincide with the initial point of $\boldsymbol{q}_j$, and is activated only when the optimal path in the GCS traverses the edge e.

The source s and the target t are auxiliary vertices that enforce the boundary conditions. They are paired with the singletons $\mathcal{X}_s := \{\boldsymbol{q}_0^0\}$ and $\mathcal{X}_t = \{\boldsymbol{q}_T^0\}$. For all the edges $e = (s, i)$ outgoing the source, the set $\mathcal{X}_e$ enforces the initial conditions $\boldsymbol{q}_{i,0} = \boldsymbol{q}_0^0$. This ensures that our trajectory starts at $\boldsymbol{q}_0^0$ as desired. Similarly, for the edges $e = (i, t)$ incoming the target, we define $\mathcal{X}_e$ so that the terminal conditions $\boldsymbol{q}_{i,1} = \boldsymbol{q}_T^0$ is satisfied.

11.2.3 The convex cost functions

We let the cost of each vertex $i \in \mathcal{I}$ be the Euclidean length of the line segment $\boldsymbol{q}_i$, i.e., $f_i(\boldsymbol{x}_i) := \|\boldsymbol{q}_{i,1} - \boldsymbol{q}_{i,0}\|_2$. Therefore, every time that we move in a straight line through a safe set $\mathcal{C}_i$ we pay a price equal to the distance travelled. The source s and the target t have zero cost, as well as all the edges $e \in \mathcal{E}$.

11.2.4 Solution methods

The reformulation of the planning problem (11.3) as an SPP in GCS is exact, up to the potential conservatism of the convex decomposition of the environment. The optimal path in the GCS determines the safe regions $\mathcal{C}_i$ that our robot must traverse. The overall trajectory $\boldsymbol{q}$ is obtained by sequencing the line segments $\boldsymbol{q}_i$ associated with these regions, as shown in the bottom row of Figure 11-1.

Using the techniques introduced in this thesis, the SPP in GCS is automatically formulated as a strong Mixed-Integer Convex Program (MICP). This problem can be solved to global optimality using the Branch and Bound (BB) algorithm described in Section 3.3.2, or approximately using the rounding strategy from Section 9.5. It is interesting to observe that the rounding approach results in a complete motion

planner. In fact, any s-t path in the intersection graph identifies a sequence of safe regions $\mathcal{C}_{i_0}, \ldots, \mathcal{C}_{i_l}$ that connect the initial point $\boldsymbol{q}_0^0$ to the final point $\boldsymbol{q}_T^0$. Once these regions are fixed, the resulting convex restriction (9.9) reads

$$\begin{aligned} \text{minimize} \quad & \sum_{j=0}^{l} \|\boldsymbol{p}_{j+1} - \boldsymbol{p}_j\|_2 \\ \text{subject to} \quad & \boldsymbol{p}_0 = \boldsymbol{q}_0^0, \\ & \boldsymbol{p}_{l+1} = \boldsymbol{q}_T^0, \\ & \boldsymbol{p}_j \in \mathcal{C}_{i_{j-1}} \cap \mathcal{C}_{i_j}, \qquad\qquad \forall j = 1, \ldots, l, \end{aligned}$$

where the variables $\boldsymbol{p}_0, \ldots, \boldsymbol{p}_{l+1}$ represent the "kinks" of the polygonal trajectory $\boldsymbol{q}$. Since this convex restriction is always feasible, the rounding method from Section 9.5 is guaranteed to find a solution.

11.3 Bézier curves

In the previous section we have restricted our attention to polygonal trajectories, which are conveniently parameterized through a finite number of points. Bézier curves give us a natural way of generalizing the approach taken above to the design of smooth trajectories.

Bézier curves enjoy several properties that make them particularly well suited for convex optimization, and have been widely used in motion planning and optimal control over the last fifteen years (early works in this direction are, e.g., [65, 37, 111]). In this section we provide some basic notions and results about this family of curves.

11.3.1 Definition

A **Bézier curve** is constructed using **Bernstein polynomials**. The Bernstein polynomials of degree N are defined over the interval $[a, b] \subset \mathbb{R}$, with $b > a$, as

$$\beta_n(\tau) := \binom{N}{n} \left(\frac{\tau - a}{b - a}\right)^n \left(\frac{b - \tau}{b - a}\right)^{N-n}, \qquad \forall n = 0, \ldots, N.$$

For all $\tau \in [a, b]$ the Bernstein polynomials are nonnegative and, by the binomial theorem, they sum up to one. Therefore, the scalars $\beta_0(\tau), \ldots, \beta_N(\tau)$ can be thought of as the coefficients of a convex combination. Using these coefficients to combine a

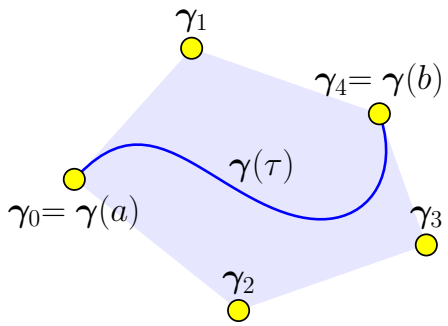

Figure 11-2: Bézier curve with control points $\boldsymbol{\gamma}_0, \ldots, \boldsymbol{\gamma}_N$, with $N = 4$. The curve starts at $\boldsymbol{\gamma}(a) = \boldsymbol{\gamma}_0$, ends at $\boldsymbol{\gamma}(b) = \boldsymbol{\gamma}_N$, and is entirely contained in the shaded control polytope (convex hull of the control points).

given set of **control points** $\boldsymbol{\gamma}_0, \ldots, \boldsymbol{\gamma}_N \in \mathbb{R}^d$, we obtain a Bézier curve:

$$\boldsymbol{\gamma}(\tau) := \sum_{n=0}^{N} \beta_n(\tau) \boldsymbol{\gamma}_n.$$

The Bézier curve $\boldsymbol{\gamma} : [a, b] \to \mathbb{R}^d$ is a (vector-valued) polynomial function of degree N. An example of a Bézier curve is shown in Figure 11-2, for $d = 2$ and $N = 4$.

A **composite Bézier curve** is a sequence of Bézier curves that connect smoothly up to some derivative. Below we list some important properties that we will use later in this section.

11.3.2 Endpoints

A Bézier curve starts at its first control point and ends at its last control point:

$$\boldsymbol{\gamma}(a) = \boldsymbol{\gamma}_0, \qquad \boldsymbol{\gamma}(b) = \boldsymbol{\gamma}_N.$$

With this property, a composite Bézier curve (our trajectory) can be made continuous simply by equating the last control point of each curve piece with the first control point of the next piece.

11.3.3 Control polytope

Since each point on a Bézier curve is a convex combination of the control points, the entire curve is contained in the convex hull of the control points:

$$\boldsymbol{\gamma}(\tau) \in \operatorname{conv}(\{\boldsymbol{\gamma}_0, \ldots, \boldsymbol{\gamma}_N\}), \qquad \forall \tau \in [a, b].$$

This convex hull is called the **control polytope** of the Bézier curve $\boldsymbol{\gamma}$. From this property it follows that if all control points lie in a convex set, then so does the whole Bézier curve.

11.3.4 Derivatives

The derivative $\boldsymbol{\gamma}^{(1)}$ of a Bézier curve $\boldsymbol{\gamma}$ is also a Bézier curve. It has degree $N-1$ and its control points are given by the difference equation

$$\boldsymbol{\gamma}_n^{(1)} = \frac{N}{b-a}(\boldsymbol{\gamma}_{n+1} - \boldsymbol{\gamma}_n), \qquad \forall n = 0, \dots, N-1.$$

Iterating this, we see that the derivative $\boldsymbol{\gamma}^{(k)}$ of any order k is a Bézier curve of degree $N-k$. Moreover, the derivatives of a piecewise Bézier curve are also piecewise Bézier curves, and their continuity can be enforced using the endpoint property above.

11.3.5 Squared $\mathcal{L}_2$ norm

The square of the $\mathcal{L}_2$ norm of a Bézier curve $\boldsymbol{\gamma}$ can be expressed as an explicit function of the control points using the following expression [60, Section 3.3]:

$$\int_a^b \|\boldsymbol{\gamma}(\tau)\|_2^2 d\tau = (b-a)Q(\boldsymbol{\gamma}_0, \dots, \boldsymbol{\gamma}_N), \tag{11.4}$$

where Q is a convex quadratic function of the control points defined as

$$Q(\gamma_0, \dots, \gamma_N) := \frac{1}{2N+1} \sum_{n=0}^{N} \sum_{m=0}^{N} \frac{\binom{N}{n}\binom{N}{m}}{\binom{2N}{n+m}} \boldsymbol{\gamma}_n^T \boldsymbol{\gamma}_m.$$

This formula allows us also to compute the squared $\mathcal{L}_2$ norm of a composite Bézier curve and its derivatives.

11.3.6 Integral upper bound

Unlike the squared $\mathcal{L}_2$ norm, the integral of a generic convex function of a Bézier curve cannot be expressed in closed form. However, we can easily derive a computationally

cheap upper bound. Let $f : \mathbb{R}^d \to \mathbb{R}$ be a convex function. We have

$$\begin{aligned}\int_a^b f(\boldsymbol{\gamma}(\tau))d\tau &= \int_a^b f\left(\sum_{n=0}^N \beta_n(\tau)\boldsymbol{\gamma}_n\right) d\tau \\ &\leq \int_a^b \sum_{n=0}^N \beta_n(\tau) f(\boldsymbol{\gamma}_n) d\tau \\ &= \sum_{n=0}^N f(\boldsymbol{\gamma}_n) \int_a^b \beta_n(\tau) d\tau \\ &= \sum_{n=0}^N f(\boldsymbol{\gamma}_n) \frac{b-a}{N+1} \\ &= \sum_{n=0}^N f(\boldsymbol{\gamma}_n).\end{aligned}$$

Here the inequality follows from the convexity of f as in (2.10), and the second to last equality uses the formula of the integral of a Bernstein polynomial.

11.4 Smooth trajectories

Bézier curves allow us to generalize the construction in Section 11.2, and give us a way to tackle our original planning problem (11.2). The main caveat is that all the desirable convexity properties of Bézier curves hold only if the time duration $b - a$ of the curve is fixed. Let us then discuss how we can solve problem (11.2) under the following (quite restrictive) assumption.

Assumption 11.1. For all $i \in \mathcal{I}$ the amount of time that our trajectory can spend within the convex region $\mathcal{C}_i$ is either zero or a fixed scalar $T_i > 0$.

Under this hypothesis, the approach in Section 11.2 is easily generalized. We still use the intersection graph G as a skeleton for our GCS. Before we paired each vertex $i \in \mathcal{I}$ with just two points, now we pair it with the following continuous variables:

- A sequence of $N_i + 1$ control points $\boldsymbol{q}_{i,0}, \ldots, \boldsymbol{q}_{i,N_i}$. These define a Beźier curve $\boldsymbol{q}_i$ with given time duration T_i.
- A scalar τ_i that represents the time at which our trajectory enters the region $\mathcal{C}_i$.

The source s is now coupled with the singleton

$$\mathcal{X}_s := \{(\boldsymbol{q}_0^0, \ldots, \boldsymbol{q}_0^K, 0)\}.$$

The first entries collect the initial conditions and the last entry is the initial time of the trajectory. The target t is paired with the set

$$\mathcal{X}_t := \{(\boldsymbol{q}_T^0, \dots, \boldsymbol{q}_T^K)\} \times [T_{\min}, T_{\max}],$$

which enforces the final conditions and the bounds on the time duration of the trajectory. The convex set $\mathcal{X}_i$ for each $i \in \mathcal{I}$ enforces the following constraints:

- Each control point $\boldsymbol{q}_{i,n}$ must be contained in $\mathcal{C}_i$. By the convex-hull property of Beźier curves, this implies that the whole curve $\boldsymbol{q}_i$ is contained in $\mathcal{C}_i$.
- The control points of the derivatives $\boldsymbol{q}_i^{(k)}$ must lie in the convex sets $\mathcal{C}^k$. Again by the convex-hull property, this ensures that the constraints (11.2e) are satisfied. Note also that the control points of $\boldsymbol{q}_i^{(k)}$ are linear functions of the ones of $\boldsymbol{q}_i$.
- The initial time τ_i must be nonnegative and not larger than $T_{\max}$.

In addition to the initial position, now the convex sets $\mathcal{X}_e$ with $e = (s, i)$ take care also of the following constraints:

- Initial conditions for the curve derivatives $\boldsymbol{q}_i^{(k)}$, as prescribed by the constraint (11.2b).
- Initial value of the time counter $\tau_i = 0$.

The edges $(i, t) \in \mathcal{E}$ incoming the target play a specular role: they enforce the terminal conditions (11.2c) and constrain the final time $T = \tau_i + T_i$ in the interval $[T_{\min}, T_{\max}]$. All the other edges of the form $(i, j) \in \mathcal{E} \cap \mathcal{I}^2$ enforce:

- The continuity of the curves $\boldsymbol{q}_i$ and $\boldsymbol{q}_j$ as well as their derivatives $\boldsymbol{q}_i^{(k)}$ and $\boldsymbol{q}_j^{(k)}$. Again, this is done by equating the final control point the first curve with the initial control point of the second curve.
- Accumulate the time spent along the trajectory: $\tau_j = \tau_i + T_i$.

Finally, convex costs on the vertices of the graph can be enforced either exactly (e.g., in the case of the squared $\mathcal{L}_2$ norm) or approximately by using the upper bound derived above. The cost $\varphi_0(T)$ on the trajectory duration is enforced through the cost f_t of the target vertex.

Under Assumption 11.1, this SPP in GCS models exactly our motion-planning problem. Also, any feasible solution of this SPP in GCS yields a feasible solution for the motion-planning problem (11.2). In this case the rounding strategy from

Section 9.5 might not yield a feasible solution, unless the upper bound $T_{\max}$ on the trajectory duration is sufficiently large to make any path through our graph feasible.[1]

11.4.1 Joint optimization of trajectory shape and timing

In practice, the trajectories generated with the method above can be quite suboptimal. Since the time T_i pre-allocated for each convex region $\mathcal{C}_i$ can be far from the optimal one. The approach that we take in the numerical experiments below is to pair with each vertex $i \in \mathcal{I}$ two Bézier curves: $\boldsymbol{q}_i : [0,1] \to \mathbb{R}^d$ and $h_i : [0,1] \to \mathbb{R}$. The former decides the shape of our trajectory within each convex set, the second dictates the timing at which the curve $\boldsymbol{q}_i$ is swept. Mathematically, the trajectory within the convex set $\mathcal{C}_i$ is reconstructed as the composite function $\boldsymbol{q}_i \circ h_i^{-1}$.

With this parameterization we can still design smooth trajectories, by enforcing the differentiability of the curves $\boldsymbol{q}_i$ and h_i independently. However, this approach makes it complicated to enforce costs and constraints on the trajectory derivatives. Convex costs and constraints on the trajectory velocity are still convex when expressed in terms of the curves $\boldsymbol{q}_i$ and h_i. However, convex costs and constraints on the second and higher derivatives are not convex in $\boldsymbol{q}_i$ and h_i. A practical approach to prevent, e.g., the acceleration from growing too large is to add a small penalty directly on the magnitudes of $\boldsymbol{q}_i^{(2)}$ and $h_i^{(2)}$, and constrain $h_i^{(1)}$ to not approach zero too closely. A similar strategy can be used to regularize the higher-order derivatives of our trajectory.

11.5 Numerical experiments

We demonstrate our motion-planning method (which we call GCS after the underlying optimization framework) on a variety of robot platforms. First, to illustrate some key advantages of GCS over existing planners, we consider a motion-planning problem in an intricate maze. Then we consider a quadrotor flying through randomly-generated buildings, and we show that in almost every problem instance GCS synthesizes a globally optimal trajectory. For a robot arm with seven joints, we show that GCS can outperform widely used sampling-based planners by finding higher-quality trajectories in less time. Finally, we demonstrate the scalability of GCS on real hardware with a bimanual manipulation problem in a fourteen-dimensional configuration space.

All experiments are run on a computer with a Threadripper 3990x processor and 256 GB of RAM. The solver used for the convex optimization problems is `MOSEK 10.0`.

[1] This also assumes that the boundary conditions, together with the derivative constraints, do not make the problem infeasible.

The code necessary to reproduce the results below is available at

`https://github.com/RobotLocomotion/gcs-science-robotics`.

For an efficient implementation of the techniques discussed in this chapter see also the class `GcsTrajectoryOptimization` provided by the software `Drake` [183].

11.5.1 Motion planning in a maze

Designing a smooth trajectory across a maze is a very challenging problem for most motion planners. Consider the maze with $50^2 = 2{,}500$ cells shown in Figure 11-3. Through small perturbations of an initial trajectory, local optimization has almost no chance of finding a way across this maze. Existing mixed-integer planners would also fail, since they would parameterize a trajectory as a sequence of curves and use a binary variable to assign each curve to each cell [47]. Given that a trajectory might need to visit every cell, this would require $2,500^2 \approx 10^7$ binary variables: a quantity well beyond the capabilities of any solver. Sampling-based methods could easily discover a path through the maze in Figure 11-3, but they would struggle with continuous differential costs and constraints. GCS, on the other hand, can efficiently design smooth trajectories while explicitly leveraging the graph structure beneath this problem.

To put the problem in Figure 11-3 in the form required by GCS, we let each maze cell be a safe set $\mathcal{C}_i$. We then have a total of $|\mathcal{I}| = 2,500$ safe sets, each of which is a unit square. The initial $\boldsymbol{q}_0^0$ and final $\boldsymbol{q}_T^0$ points are the entry (bottom left) and exit (top right) of the maze. We consider two problems: first we look for the shortest continuous trajectory across the maze as in problem (11.3), then we minimize the trajectory duration ($\varphi_0(T) := T$ and $\varphi_k := 0$ for all $k \geq 1$ in problem (11.2)) together with a small acceleration penalty as described in Section 11.4.1. In the second problem, we also require that the trajectory be differentiable twice, and we enforce the velocity limits $\mathcal{C}^1 := [-1, 1]^2$ and the boundary conditions $\boldsymbol{q}_0^1 := \boldsymbol{q}_T^1 := \mathbf{0}$.

The trajectories designed by GCS are illustrated in Figure 11-3, in dashed red for the first problem and in solid blue for the second. In both cases, the convex relaxation and the rounded solution have equal cost, thus GCS automatically certifies that these trajectories are optimal ($\delta_{\text{opt}} = \delta_{\text{relax}} = 0$). When the objective is to minimize the trajectory length, our convex relaxation is a Second-Order Cone Program (SOCP) and is solved in 0.98s. The minimum-time problem is a Linear Program (LP) but, because of the high-degree trajectory parameterization, it is substantially larger than the SOCP and takes 9.1s. We highlight that these are relaxations of mixed-integer

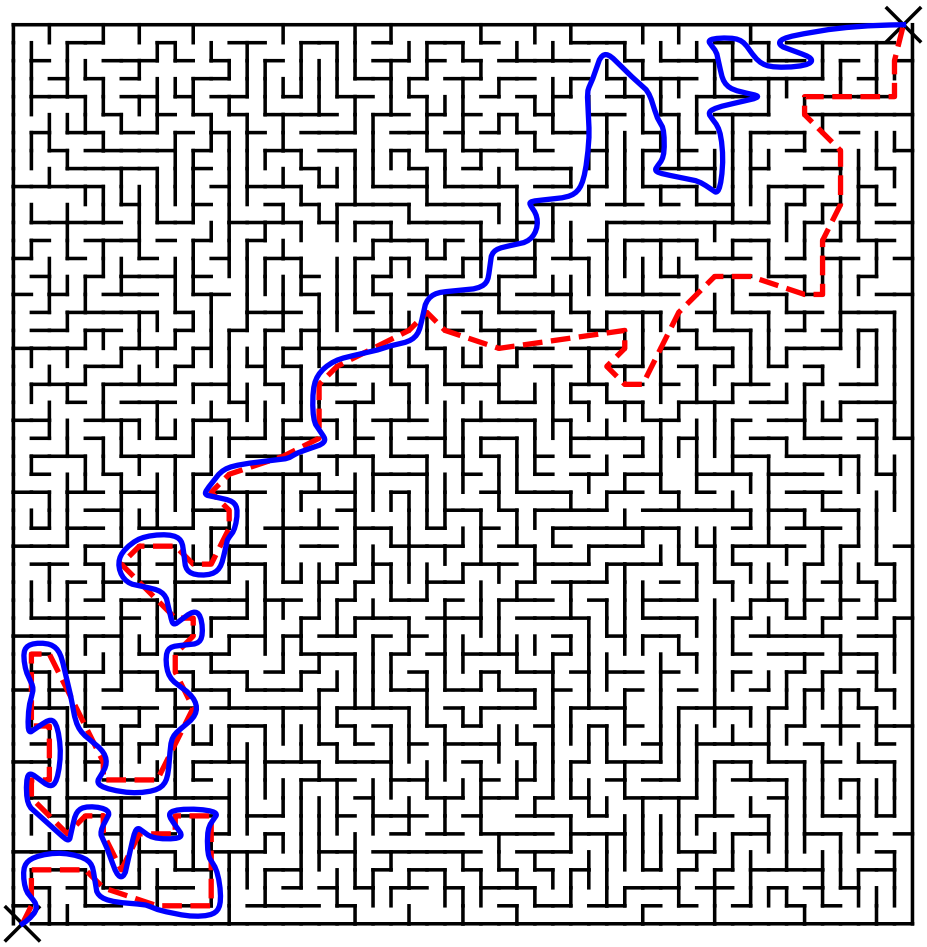

Figure 11-3: Motion planning in a maze. The dashed-red and solid-blue trajectories are generated by GCS and correspond, respectively, to a minimum-length objective and to a minimum-time objective with velocity constraints and acceleration penalties. For both problems, GCS identifies an optimal trajectory, and certifies global optimality, via a single convex optimization problem.

programs with approximately 5,000 binary variables, as opposed to, e.g., the 10^7 binaries that the mixed-integer formulation from [47] would use.

This example highlights the transparency with which GCS blends discrete and continuous optimization. Finding a discrete sequence of cells through a maze is a simple graph search, and mild differential costs and constraints should not make the problem dramatically harder, especially if we seek approximate solutions. While existing algorithms fail in merging the discrete and continuous natures of motion planning, GCS can efficiently design smooth trajectories while explicitly leveraging the graph structure underlying our problems.

11.5.2 Quadrotor flying through buildings

We use GCS to plan the flight of a quadrotor through randomly generated buildings. An example of such a task is illustrated in Figure 11-4: while moving from the starting to the ending platform, the quadrotor needs to fly around trees, and through doors and windows. For each building, the number of rooms, the shape of the walls, the positioning of doors, windows, and trees are selected randomly. The starting point is always outside the building and the target is always inside.

Although the configuration space of a quadrotor is six dimensional, the differential-flatness property of this system allows us to plan trajectories directly in the three-dimensional Cartesian space. In fact, given any trajectory for the center of mass that is differentiable four times, a dynamically feasible trajectory for the quadrotor's orientation, together with the necessary control thrusts, can always be reconstructed [135].

Given that all the obstacles are polygonal, the decomposition of the free space into convex sets $\mathcal{C}_i$ is done exactly. The collision geometry of the quadrotor is taken to be a sphere. The cost penalizes the trajectory duration and length, and includes a small acceleration penalty. To leverage the differential flatness, we require our trajectories to be differentiable four times, $K = 4$. The velocity has boundary conditions $\boldsymbol{q}_0^1 = \boldsymbol{q}_T^1 = 0$ and is constrained in the box $\mathcal{C}^1 = [-10, 10]^3$.

We plan the motion of the quadrotor through 100 random buildings and we analyze the optimality gaps δ_{opt} and δ_{relax} defined in (3.4) and obtained through the rounding strategy discussed in Section 9.5. The value of δ_{opt} is computed, just for analysis purposes, by solving the MICP to global optimality using BB. The gap $\delta_{\text{relax}} \geq \delta_{\text{opt}}$ is automatically returned by GCS. The histograms of these values across the experiments are reported in Figure 11-5. On 95% of the environments GCS designs a trajectory with optimality gap $\delta_{\text{opt}} \leq 1\%$, and, even in the worst case, we only have $\delta_{\text{opt}} = 2.9\%$. On 68% (respectively 84%) of the problems GCS certifies that the designed trajectory is within $\delta_{\text{relax}} = 4\%$ (respectively $\delta_{\text{relax}} = 7\%$) of the optimum.

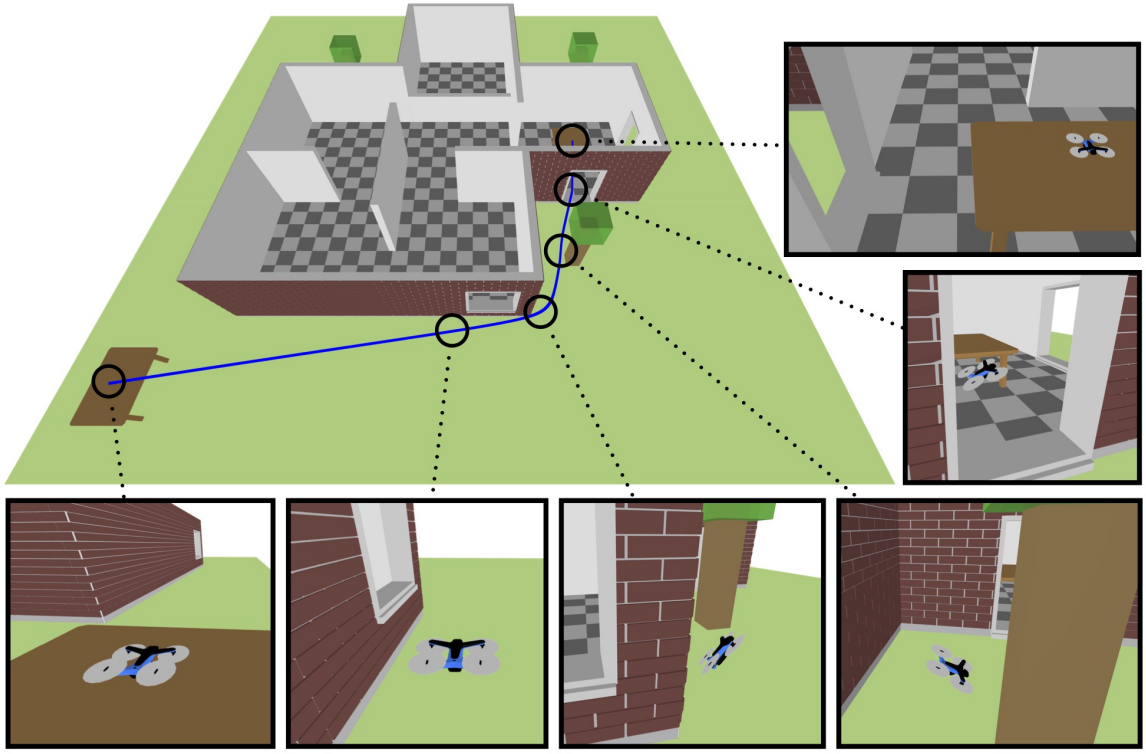

Figure 11-4: Quadrotor flying through a building. The trajectory generated by GCS is depicted in blue in the top-left panel, and is designed through a single convex optimization problem followed by a rounding step. The snapshots show the starting and ending configurations, as well as the quadrotor flying close to the obstacles in the environment.

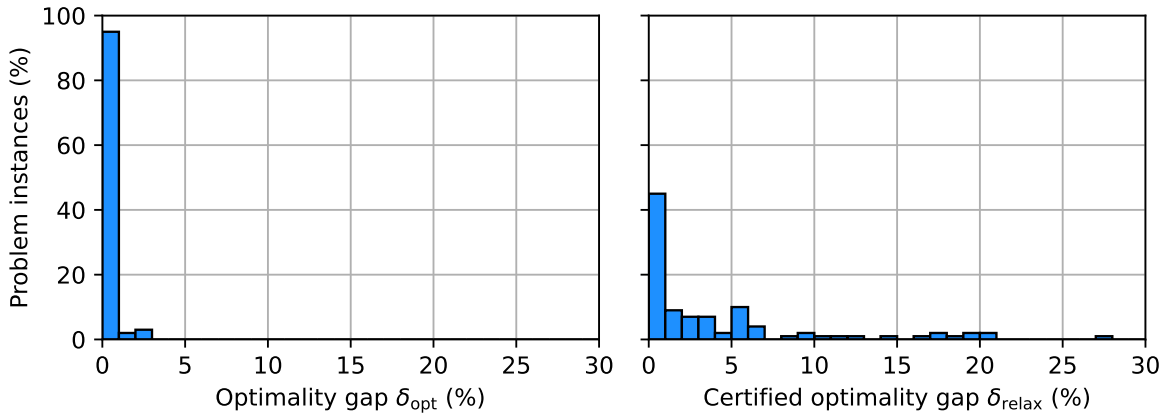

Figure 11-5: GCS is used to plan the flight of a quadrotor through one hundred randomly generated buildings. For these trajectories, the two histograms illustrate the actual optimality gap δ_{opt} and the optimality gap $\delta_{\text{relax}} \geq \delta_{\text{opt}}$ automatically certified by our method.

The largest optimality gap certified by GCS is $\delta_{\text{relax}} = 27.1\%$, and corresponds to an environment where we have $\delta_{\text{opt}} = 2.3\%$. Therefore, even for this instance, the moderately large value of δ_{relax} is mostly due to the convex relaxation being slightly loose, rather than the trajectory being suboptimal. The relaxations of these problems are SOCPs, and the average runtime of GCS is 2.65s (including rounding).

11.5.3 Comparison with sampling-based planners

Our algorithm is a multiple-query method, as the same data structure (the GCS) can be used to plan motions between many initial and final conditions. Its natural sampling-based competitor is then the Probabilistic-RoadMap (PRM) algorithm [105]. Here we compare GCS with PRM methods on a robot arm (KUKA LBR iiwa) with $d = 7$ degrees of freedom. We choose a relatively low-dimensional configuration space for this comparison since PRM can struggle in higher-dimensional spaces.

The robot environment is shown in Figure 11-6, and is composed of multiple shelves and two bins. An exact decomposition of the free space is infeasible here, therefore we adopt the approximate algorithm from [152]. Given a "seed configuration" of the robot, this algorithm inflates a polytope of robot configurations that are not in collision with the obstacles. Here these seeds are selected manually, although the seeding can be made automatic [192]. We construct $|\mathcal{I}| = 8$ safe polytopes $\mathcal{C}_i$ using this workflow: five are seeded to cover the configurations for which the gripper is close to the shelves and the bins, three are seeded to fill the rest of the free space. The construction of the safe regions is parallelized, and takes 50s.

In practice, the trajectories generated by PRM can be very suboptimal and are

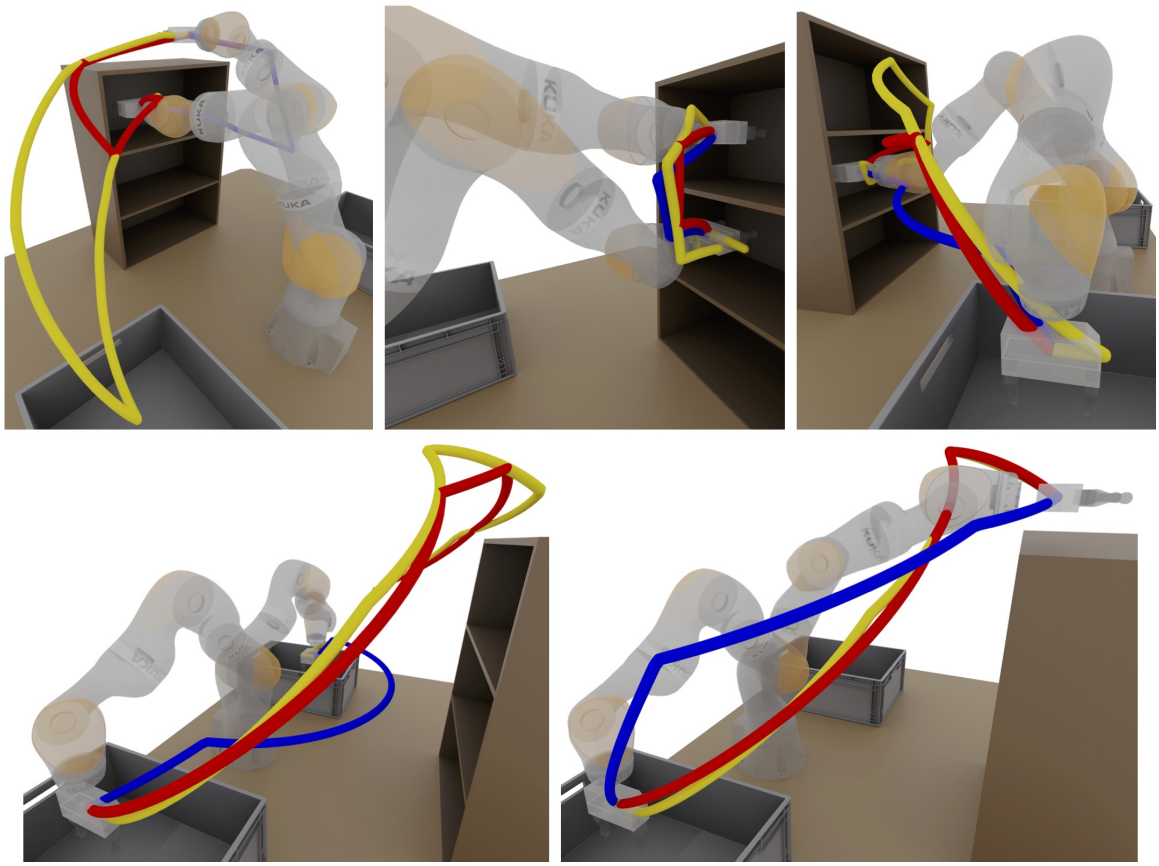

Figure 11-6: GCS is benchmarked against PRM on five tasks. Two variants of PRM are considered: the standard PRM, and the PRM followed by a shortcutting algorithm. The gripper trajectories are shown in blue for GCS, yellow for PRM, and red for PRM with shortcutting.

rarely sent to the robot directly. Even though asymptotically optimal versions of PRM exist [102], in our experience, these variants are most effective in very low-dimensional spaces (e.g., two or three dimensions). A common workaround is to post-process the PRM trajectories with a simple shortcutting algorithm. This samples pairs of points along a trajectory and connects them via straight segments: if a segment is collision free then the trajectory is successfully shortened. This step can substantially shorten the PRM trajectories but it requires many collision checks: for this reason we compare GCS with both the regular PRM and the PRM with shortcutting. We use a high-performance implementation of PRM based on [154], and soon to be included in the open-source software Drake [183]. The PRM in this comparison has 10,000 nodes, and its construction takes 0.54s. Note that this time is in line with (or even faster than) state-of-the-art PRM implementations [155, 143, 96].

The tasks in this comparison are illustrated in Figure 11-6 and require the arm to move between five configurations $\boldsymbol{\rho}_1, \dots, \boldsymbol{\rho}_5 \in \mathcal{S}$, while avoiding the shelves and the bins. The configurations $\boldsymbol{\rho}_1$, $\boldsymbol{\rho}_2$, and $\boldsymbol{\rho}_3$ correspond to the gripper being above the shelves, in the first shelf, and in the second shelf. The waypoints $\boldsymbol{\rho}_4$ and $\boldsymbol{\rho}_5$ correspond to the left and the right bin. For $\nu = 1, \dots, 4$, task ν asks the robot to move from $\boldsymbol{\rho}_\nu$ to $\boldsymbol{\rho}_{\nu+1}$. Task 5 requires moving from $\boldsymbol{\rho}_5$ back to $\boldsymbol{\rho}_1$. The objective is to connect the start and the goal configurations with a continuous trajectory of minimum length as in problem (11.3). We adopt the rounding strategy discussed in Section 9.5 for the solution of this problem.

Figure 11-6 illustrates the trajectories of the robot gripper for each planner and each task. The blue curves correspond to GCS, the yellow to PRM, and the red to PRM with shortcutting. The lengths of these trajectories, together with the offline and online runtimes of each planner, are reported in Figure 11-7 with matching colors. (Although the rounding phase of GCS is randomized, Figure 11-7 reports only one value for the length of its trajectories since repeating the experiments many times GCS always finds the same solutions.) In all the tasks, GCS designs shorter trajectories and, online, it runs as fast or substantially faster than both the PRM competitors.

For these problems our convex relaxation is an SOCP, which is solved very fast thanks to the low number $|\mathcal{I}| = 8$ of safe regions. Within the conservatism of the space decomposition, all the trajectories designed by GCS are optimal ($\delta_{\text{opt}} = 0$). The certified optimality gap δ_{relax} is 4.1% on average, and it achieves a maximum of 13.0% in the first task.

Although limited to five tasks, this comparison shows a general tendency. The offline computations of GCS are more demanding than the ones of PRM. However, this extra effort pays back in the online phase, where GCS can find better trajectories

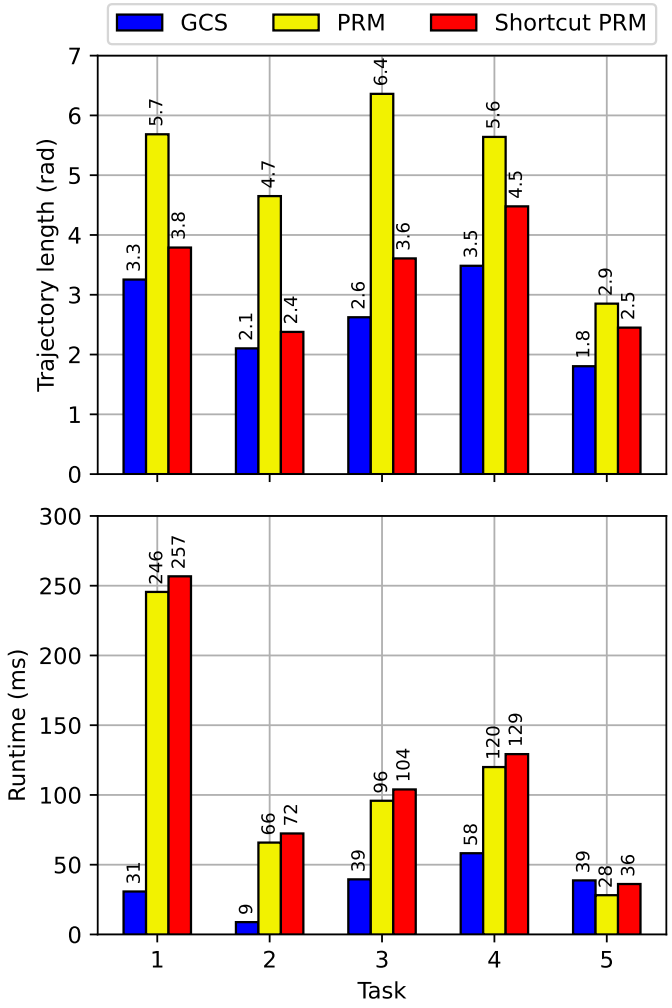

Figure 11-7: Trajectory lengths and computation times for GCS and the PRM competitors. GCS requires more expensive offline computations than highly optimized PRM implementations, but online it designs shorter trajectories in a fraction of the time.

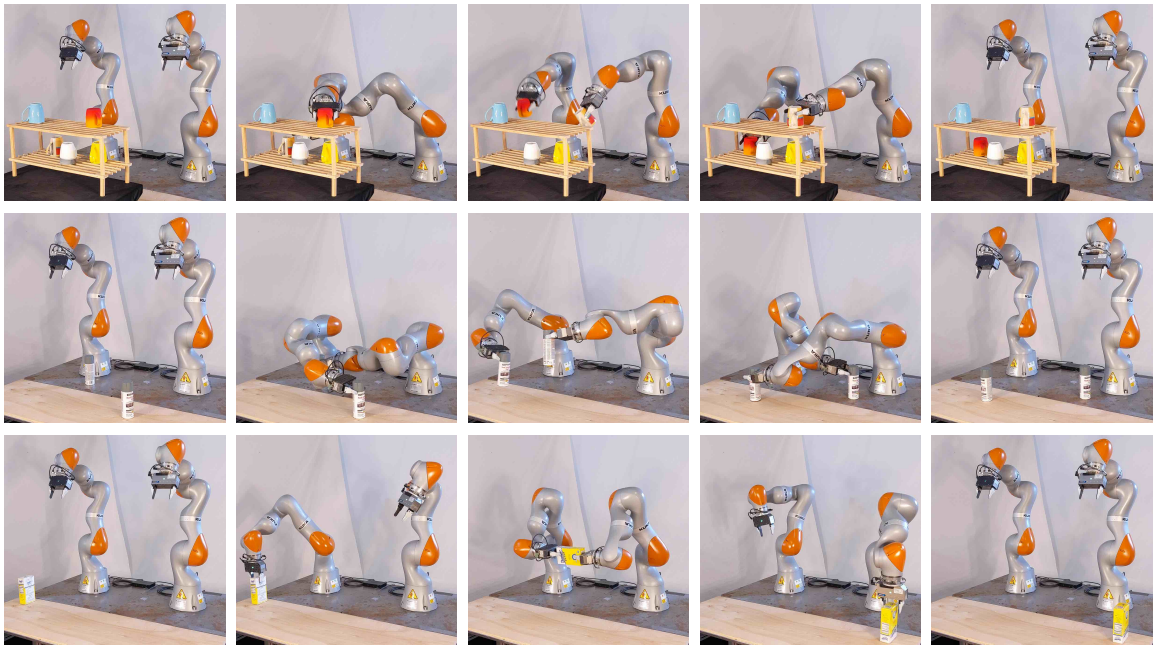

Figure 11-8: Coordinated motion planning of two robot arms using GCS. *Top row:* the arms grasp two mugs on the shelves and place them back in inverted positions. *Center row:* two spray paints are swapped. *Bottom row:* one arm grasps a sugar box and hands it to the other arm.

with stronger collision-avoidance guarantees in less time. This can be very valuable in those applications where the environment is practically static, and improving the online performance is worth investing more effort in the offline preprocessing.

11.5.4 Coordinated Planning of Two Robot Arms

In the previous comparison we have chosen a robot with $d = 7$ joints because PRM methods perform poorly in higher dimensions. To demonstrate the scalability of GCS, here we plan the motion of a dual-arm manipulator with $d = 14$ degrees of freedom, shown in Figure 11-8. Besides avoiding collisions with the environment, now GCS must also prevent self-collisions between the arms. (Note that single-query sampling-based algorithms have been applied before for problems like this [151]. However, the multiple-query problem considered here is significantly more challenging.)

We consider the three tasks shown in Figure 11-8. In the first task the robot grasps two mugs from the shelves in front of it, and places them back in a position that requires the arms to cross. In the second task it moves two spray paints and reaches configurations that are very close to self collision. In the third task one arm hands a small box to the other. Each task is solved as a single convex program: when an arm grasps, places, or hands off an object its configuration is prespecified,

but the order in which the objects are manipulated is optimized. For example, in the first task, GCS can decide which mug to grasp first, or whether to grasp them simultaneously.

The free space is again decomposed using the algorithm from [152]. For the first environment we construct $|\mathcal{I}| = 35$ polytopes, seeded to approximately cover the workspace under the four possible collision geometries (mugs on shelves, mug in left hand, mug in right hand, mugs in both hands). We proceed similarly for the second and third tasks, which feature 12 and 14 safe polytopes. Since here we are not comparing to PRM, we are not limited to polygonal curves and we plan twice-differentiable trajectories with equal penalty on length and duration, and with a small acceleration cost. The robot velocity limits are enforced through a box-shaped constraint set $\mathcal{C}^1$.

As Figure 11-8 shows, GCS designs collision-free trajectories that efficiently accomplish each of the three tasks. The largest optimality gap δ_{relax} is in the second task, and it is only 13.9%. After running a BB solver, we verify that the actual optimality gap for each trajectory is not larger than $\delta_{\text{opt}} = 8.4\%$. Since the convex relaxations of these problems are very large SOCPs, which push the limits of what GCS is currently capable of, the runtimes for these tasks are 185s, 103s, and 21s. However, we are confident that these times can decrease substantially in the near future.

Chapter 12

Conclusions

In this thesis we have introduced Graph of Convex Sets (GCS): a new modelling and computational framework for problems at the intersection of discrete and continuous decision making. GCS problems blend graph search and convex optimization, and generalize a broad variety of decision-making problems. They have great modelling power and applications in multiple engineering disciplines.

For the solution of a GCS problem we have proposed a unified method that leverages tools from polyhedral combinatorics and convex analysis. We start from the formulation of a purely discrete graph optimization problem as an integer linear program, and we automatically translate it into an efficient mixed-integer formulation for the corresponding GCS problem.

Compared to existing frameworks for discrete-continuous decision making, GCS has three main strengths. First, GCS is a simple framework that captures the essence of discrete-continuous decision making. The graph and the combinatorial goal (e.g., finding a path through a graph) describe the high-level discrete skeleton of a problem, while the convex costs and constraints model the low-level continuous details. Second, thanks to the techniques we have developed, GCS problems can be solved using efficient optimization methods. Third, GCS is a high-level tool for the users: although it leverages advanced optimization techniques, it does not require to be familiar with them.

GCS has numerous applications; in this thesis we focused on optimal control of dynamical systems and robot motion planning. In optimal control, we have demonstrated that GCS yields mixed-integer formulations that outperform the state of the art, or recover it as a special case. In motion planning, our optimization methods can design better trajectories in a fraction of the time of algorithms that have been developed for decades and are widely used in academia and industry.

We are optimistic that GCS will find more and more applications in the future, and

will become a widely-used optimization tool in many engineering disciplines. Current investigations show that GCS is a powerful language to work on a variety of problems that go beyond the ones studied in this thesis. These include contact-rich robot manipulation [83], control with temporal-logic specifications [109], and motion-planning problems with moving targets [153]. Extensions of the results presented in this thesis to graphs composed of geodesically convex manifolds have been developed in [40]. Also a high-performance motion-planning method similar to GCS, but specialized to safe sets that are axis-aligned boxes, has been presented in [127]. See the following section from [181] for an up-to-date list of the robotics applications of GCS:

`https://underactuated.mit.edu/optimization.html#gcs`.

The optimization methods introduced in this thesis are now finding application also in industry. For example, Dexai Robotics is using a variation of our techniques in their production lines to compute optimal motions for food-preparation robots [72].

Index

Bibliography

[1] Warren P Adams and Hanif D Sherali. Linearization strategies for a class of zero-one mixed integer programming problems. *Operations Research*, 38(2):217–226, 1990.

[2] Amazon Robotics. Amazon.com announces first quarter results, 2023. Available at `https://ir.aboutamazon.com/news-release/news-release-details/2023/Amazon.com-Announces-First-Quarter-Results/default.aspx`.

[3] Alexandre Amice, Hongkai Dai, Peter Werner, Annan Zhang, and Russ Tedrake. Finding and optimizing certified, collision-free regions in configuration space for robot manipulators. In *Algorithmic Foundations of Robotics XV*, pages 328–348. Springer, 2022.

[4] Esther M Arkin and Refael Hassin. Approximation algorithms for the geometric covering salesman problem. *Discrete Applied Mathematics*, 55(3):197–218, 1994.

[5] Federico Augugliaro, Angela P Schoellig, and Raffaello D'Andrea. Generation of collision-free trajectories for a quadrocopter fleet: A sequential convex programming approach. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1917–1922. IEEE, 2012.

[6] Daniel Axehill and Anders Hansson. A dual gradient projection quadratic programming algorithm tailored for model predictive control. In *Decision and Control, 2008. CDC 2008. 47th IEEE Conference on*, pages 3057–3064. IEEE, 2008.

[7] Daniel Axehill, Lieven Vandenberghe, and Anders Hansson. Convex relaxations for mixed integer predictive control. *Automatica*, 46(9):1540–1545, 2010.

[8] Nora Ayanian and Vijay Kumar. Abstractions and controllers for groups of robots in environments with obstacles. In *International Conference on Robotics and Automation*. IEEE, 2010.

[9] Egon Balas. Disjunctive programming: Properties of the convex hull of feasible points. *Discrete Applied Mathematics*, 89(1-3):3–44, 1998.

[10] Alberto Bemporad, Giancarlo Ferrari-Trecate, and Manfred Morari. Observability and controllability of piecewise affine and hybrid systems. *IEEE transactions on automatic control*, 45(10):1864–1876, 2000.

[11] Alberto Bemporad, Domenico Mignone, and Manfred Morari. An efficient branch and bound algorithm for state estimation and control of hybrid systems. In *Proceedings of the European Control Conference*, pages 557–562. Citeseer, 1999.

[12] Alberto Bemporad and Manfred Morari. Control of systems integrating logic, dynamics, and constraints. *Automatica*, 35(3):407–427, 1999.

[13] Aharon Ben-Tal and Arkadi Nemirovski. *Lectures on modern convex optimization: analysis, algorithms, and engineering applications*. SIAM, 2001.

[14] Dmitry Berenson, Pieter Abbeel, and Ken Goldberg. A robot path planning framework that learns from experience. In *2012 IEEE International Conference on Robotics and Automation*, pages 3671–3678. IEEE, 2012.

[15] Dimitri Bertsekas. *Convex optimization theory*, volume 1. Athena Scientific, 2009.

[16] Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

[17] John T Betts. Survey of numerical methods for trajectory optimization. *Journal of guidance, control, and dynamics*, 21(2):193–207, 1998.

[18] Lars Blackmore. Autonomous precision landing of space rockets. In *Frontiers of Engineering: Reports on Leading-Edge Engineering from the 2016 Symposium*, volume 46, pages 15–20. The Bridge Washington, DC, 2016.

[19] Víctor Blanco, Elena Fernández, and Justo Puerto. Minimum spanning trees with neighborhoods: Mathematical programming formulations and solution methods. *European Journal of Operational Research*, 262(3):863–878, 2017.

[20] Richard Bormann, Florian Jordan, Joshua Hampp, and Martin Hägele. Indoor coverage path planning: Survey, implementation, analysis. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1718–1725. IEEE, 2018.

[21] Francesco Borrelli, Mato Baotić, Alberto Bemporad, and Manfred Morari. Dynamic programming for constrained optimal control of discrete-time linear hybrid systems. *Automatica*, 41(10):1709–1721, 2005.

[22] Francesco Borrelli, Alberto Bemporad, Michael Fodor, and Davor Hrovat. An MPC/hybrid system approach to traction control. *Transactions on Control Systems Technology*, 14(3):541–552, 2006.

[23] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[24] Boston Dynamics. Leaps, bounds, and backflips, 2021. Available at `https://bostondynamics.com/blog/leaps-bounds-and-backflips/`.

[25] S. Boyd and L. Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[26] Michael S Branicky, Vivek S Borkar, and Sanjoy K Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE transactions on automatic control*, 43(1):31–45, 1998.

[27] Jack Brimberg and George O Wesolowsky. Locating facilities by minimax relative to closest points of demand areas. *Computers & Operations Research*, 29(6):625–636, 2002.

[28] Rohan Budhiraja, Justin Carpentier, and Nicolas Mansard. Dynamics consensus between centroidal and whole-body models for locomotion of legged robots. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6727–6733. IEEE, 2019.

[29] Joel W Burdick, Amanda Bouman, and Elon Rimon. From multi-target sensory coverage to complete sensory coverage: An optimization-based robotic sensory coverage approach. In *International Conference on Robotics and Automation*, pages 10994–11000. IEEE, 2021.

[30] Eduardo F Camacho, Daniel R Ramírez, Daniel Limón, D Muñoz De La Peña, and Teodoro Alamo. Model predictive control techniques for hybrid systems. *Annual reviews in control*, 34(1):21–31, 2010.

[31] John Canny. *The complexity of robot motion planning*. MIT press, 1988.

[32] John Canny and John Reif. New lower bound techniques for robot motion planning problems. In *28th Annual Symposium on Foundations of Computer Science*, pages 49–60. IEEE, 1987.

[33] Federico D Carvallo, Arthur W Westerberg, and Manfred Morari. MILP formulation for solving minimum time optimal control problems. *International Journal of Control*, 51(4):943–947, 1990.

[34] Sebastián Ceria and João Soares. Convex programming for disjunctive convex optimization. *Mathematical Programming*, 86(3):595–614, 1999.

[35] Runqi Chai, Antonios Tsourdos, Al Savvaris, Senchun Chai, and Yuanqing Xia. Two-stage trajectory optimization for autonomous ground vehicles parking maneuver. *IEEE Transactions on Industrial Informatics*, 15(7):3899–3909, 2018.

[36] Wei-pang Chin and Simeon Ntafos. Optimum watchman routes. In *Proceedings of the second annual symposium on Computational geometry*, pages 24–33, 1986.

[37] Ji-wung Choi, Renwick Curry, and Gabriel Elkaim. Path planning based on Bézier curve for autonomous ground vehicles. In *Advances in Electrical and Electronics Engineering-IAENG Special Edition of the World Congress on Engineering and Computer Science 2008*, pages 158–166. IEEE, 2008.

[38] Howie Choset. Coverage for robotics–a survey of recent results. *Annals of mathematics and artificial intelligence*, 31(1):113–126, 2001.

[39] Chris Coey, Miles Lubin, and Juan Pablo Vielma. Outer approximation with conic certificates for mixed-integer convex problems. *Mathematical Programming Computation*, 12(2):249–293, 2020.

[40] Thomas Cohn, Mark Petersen, Max Simchowitz, and Russ Tedrake. Non-euclidean motion planning with graphs of geodesically-convex sets. *arXiv preprint arXiv:2305.06341*, 2023.

[41] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. Extended formulations in combinatorial optimization. *4OR*, 8(1):1–48, 2010.

[42] Michele Conforti, Gérard Cornuéjols, and Giacomo Zambelli. *Integer programming*. Springer, 2014.

[43] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 4th edition, 2022.

[44] Hongkai Dai, Alexandre Amice, Peter Werner, Annan Zhang, and Russ Tedrake. Certified polyhedral decompositions of collision-free configuration space. *The International Journal of Robotics Research*, page 02783649231201437, 2023.

[45] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[46] Robin Deits and Russ Tedrake. Computing large convex regions of obstacle-free space through semidefinite programming. In *Algorithmic Foundations of Robotics XI*, pages 109–124. Springer, 2015.

[47] Robin Deits and Russ Tedrake. Efficient mixed-integer planning for UAVs in cluttered environments. In *International Conference on Robotics and Automation*, pages 42–49. IEEE, 2015.

[48] Marc Demange, Tınaz Ekim, Bernard Ries, and Cerasela Tanasescu. On some applications of the selective graph coloring problem. *European Journal of Operational Research*, 240(2):307–314, 2015.

[49] Ashwin Deshpande. *Exact geometry algorithms for robotic motion planning*. PhD thesis, Massachusetts Institute of Technology, 2019.

[50] Steven Diamond and Stephen Boyd. CVXPY: A python-embedded modeling language for convex optimization. *The Journal of Machine Learning Research*, 17(1):2909–2913, 2016.

[51] Steven Diamond, Reza Takapoui, and Stephen Boyd. A general system for heuristic minimization of convex functions over non-convex sets. *Optimization Methods and Software*, 33(1):165–193, 2018.

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

[53] Yann Disser, Matúš Mihalák, Sandro Montanari, and Peter Widmayer. Rectilinear shortest path and rectilinear minimum spanning tree with neighborhoods. In *International Symposium on Combinatorial Optimization*, pages 208–220. Springer, 2014.

[54] Moshe Dror, Alon Efrat, Anna Lubiw, and Joseph SB Mitchell. Touring a sequence of polygons. In *35th annual ACM symposium on Theory of computing*, pages 473–482, 2003.

[55] Moshe Dror and Mohamed Haouari. Generalized Steiner problems and other variants. *Journal of Combinatorial Optimization*, 4(4):415–436, 2000.

[56] Moshe Dror, Mohamed Haouari, and J Chaouachi. Generalized spanning trees. *European Journal of Operational Research*, 120(3):583–592, 2000.

[57] Jack Edmonds. Maximum matching and a polyhedron with 0,1-vertices. *Journal of research of the National Bureau of Standards B*, 69(125-130):55–56, 1965.

[58] Jack Edmonds. Submodular functions, matroids, and certain polyhedra. *Combinatorial structures and their applications*, pages 69–87, 1970.

[59] Bachir El Khadir, Jean Bernard Lasserre, and Vikas Sindhwani. Piecewise-linear motion planning amidst static, moving, or morphing obstacles. In *2021 IEEE International Conference on Robotics and Automation*, pages 7802–7808. IEEE, 2021.

[60] Rida Farouki and V Rajan. Algorithms for polynomials in Bernstein form. *Computer Aided Geometric Design*, 5(1):1–26, 1988.

[61] Corinne Feremans, Martine Labbé, and Gilbert Laporte. Generalized network design problems. *European Journal of Operational Research*, 148(1):1–13, 2003.

[62] Corinne Feremans, Martine Labbé, and Gilbert Laporte. The generalized minimum spanning tree problem: Polyhedral analysis and branch-and-cut algorithm. *Networks: An International Journal*, 43(2):71–86, 2004.

[63] Giancarlo Ferrari-Trecate, Marco Muselli, Diego Liberati, and Manfred Morari. A clustering technique for the identification of piecewise affine systems. *Automatica*, 39(2):205–217, 2003.

[64] Matteo Fischetti, Juan José Salazar González, and Paolo Toth. The symmetric generalized traveling salesman polytope. *Networks*, 26(2):113–123, 1995.

[65] Melvin Flores. *Real-time trajectory generation for constrained nonlinear dynamical systems using non-uniform rational b-spline basis functions*. California Institute of Technology, 2008.

[66] Christodoulos A Floudas. *Nonlinear and mixed-integer optimization: fundamentals and applications*. Oxford University Press, 1995.

[67] Maria Fox and Derek Long. PDDL2.1: An extension to pddl for expressing temporal planning domains. *Journal of artificial intelligence research*, 20:61–124, 2003.

[68] Antonio Frangioni and Claudio Gentile. Perspective cuts for a class of convex 0–1 mixed integer programs. *Mathematical Programming*, 106(2):225–236, 2006.

[69] Michael L Fredman and Robert Endre Tarjan. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[70] Damian Frick, Alexander Domahidi, and Manfred Morari. Embedded optimization for mixed logical dynamical systems. *Computers & Chemical Engineering*, 72:21–33, 2015.

[71] Damian Frick, Angelos Georghiou, Juan L Jerez, Alexander Domahidi, and Manfred Morari. Low-complexity method for hybrid mpc with local guarantees. *SIAM Journal on Control and Optimization*, 57(4):2328–2361, 2019.

[72] Dexai Robotics (funded by ARM Institute). Time-optimal motion planning using convex sets, 2024. Available at `https://arminstitute.org/news/motion-planning-convex-sets/`.

[73] Enric Galceran and Marc Carreras. A survey on coverage path planning for robotics. *Robotics and Autonomous systems*, 61(12):1258–1276, 2013.

[74] Jonathan D Gammell, Siddhartha S Srinivasa, and Timothy D Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004. IEEE, 2014.

[75] Caelan Reed Garrett, Rohan Chitnis, Rachel Holladay, Beomjoon Kim, Tom Silver, Leslie Pack Kaelbling, and Tomás Lozano-Pérez. Integrated task and motion planning. *Annual review of control, robotics, and autonomous systems*, 4:265–293, 2021.

[76] Iacopo Gentilini, François Margot, and Kenji Shimada. The travelling salesman problem with neighbourhoods: MINLP solution. *Optimization Methods and Software*, 28(2):364–378, 2013.

[77] Tobias Geyer, Georgios Papafotiou, and Manfred Morari. Hybrid model predictive control of the step-down DC–DC converter. *Transactions on Control Systems Technology*, 16(6):1112–1124, 2008.

[78] Tobias Geyer, Fabio D Torrisi, and Manfred Morari. Optimal complexity reduction of polyhedral piecewise affine systems. *Automatica*, 44(7):1728–1740, 2008.

[79] Gianpaolo Ghiani and Gennaro Improta. An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research*, 122(1):11–17, 2000.

[80] Mukulika Ghosh, Nancy M Amato, Yanyan Lu, and Jyh-Ming Lien. Fast approximate convex decomposition using relative concavity. *Computer-Aided Design*, 45(2):494–504, 2013.

[81] Rafal Goebel, Ricardo G. Sanfelice, and Andrew R. Teel. *Hybrid Dynamical Systems*. Princeton University Press, Princeton, 2012.

[82] Gustavo Goretkin, Alejandro Perez, Robert Platt, and George Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 2429–2436. IEEE, 2013.

[83] Bernhard P Graesdal, Shao YC Chia, Tobia Marcucci, Savva Morozov, Alexandre Amice, Pablo A Parrilo, and Russ Tedrake. Towards tight convex relaxations for contact-rich manipulation. *arXiv preprint arXiv:2402.10312*, 2024.

[84] Michael Grant and Stephen Boyd. Cvx: Matlab software for disciplined convex programming, version 2.1, 2014.

[85] Michael Grant, Stephen Boyd, and Yinyu Ye. Disciplined convex programming. *Global optimization: From theory to implementation*, pages 155–210, 2006.

[86] Martin Grötschel, László Lovász, and Alexander Schrijver. *Geometric algorithms and combinatorial optimization*. Springer Science & Business Media, 2012.

[87] Oktay Günlük and Jeff Linderoth. Perspective reformulations of mixed integer nonlinear programs with indicator variables. *Mathematical programming*, 124(1-2):183–205, 2010.

[88] David Hallac, Christopher Wong, Steven Diamond, Abhijit Sharang, Stephen Boyd, Jure Leskovec, et al. Snapvx: A network-based convex optimization solver. *Journal of Machine Learning Research*, 18(4):1–5, 2017.

[89] Shaoning Han, Andrés Gómez, and Alper Atamtürk. 2×2-convexifications for convex quadratic optimization with indicator variables. *Mathematical Programming*, 202(1-2):95–134, 2023.

[90] Weiqiao Han and Russ Tedrake. Feedback design for multi-contact push recovery via lmi approximation of the piecewise-affine quadratic regulator. In *2017 IEEE-RAS 17th International Conference on Humanoid Robotics (Humanoids)*, pages 842–849. IEEE, 2017.

[91] Weiqiao Han and Russ Tedrake. Local trajectory stabilization for dexterous manipulation via piecewise affine approximations. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 8884–8891. IEEE, 2020.

[92] Wilhemus PMH Heemels, Bart De Schutter, and Alberto Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, 2001.

[93] Andreas B Hempel, Paul J Goulart, and John Lygeros. Inverse parametric optimization with an application to hybrid system control. *IEEE Transactions on automatic control*, 60(4):1064–1069, 2015.

[94] Jean-Baptiste Hiriart-Urruty and Claude Lemaréchal. *Convex analysis and minimization algorithms I: Fundamentals*, volume 305. Springer science & business media, 2013.

[95] Mamane Souley Ibrahim, Nelson Maculan, and Michel Minoux. A strong flow-based formulation for the shortest path problem in digraphs with negative cycles. *International Transactions in Operational Research*, 16(3):361–369, 2009.

[96] Jeffrey Ichnowski and Ron Alterovitz. Scalable multicore motion planning using lock-free concurrency. *IEEE Transactions on Robotics*, 30(5):1123–1136, 2014.

[97] Jordan Jalving, Sungho Shin, and Victor M Zavala. A graph-based modeling abstraction for optimization: Concepts and implementation in plasmo.jl. *Mathematical Programming Computation*, 14(4):699–747, 2022.

[98] Lucas Janson, Edward Schmerling, Ashley Clark, and Marco Pavone. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7):883–921, 2015.

[99] Mikael Johansson and Anders Rantzer. Computation of piecewise quadratic Lyapunov functions for hybrid systems. In *Control Conference (ECC), 1997 European*, pages 2005–2010. IEEE, 1997.

[100] Mrinal Kalakrishnan, Sachin Chitta, Evangelos Theodorou, Peter Pastor, and Stefan Schaal. STOMP: Stochastic trajectory optimization for motion planning. In *2011 IEEE International Conference on Robotics and Automation*, pages 4569–4574. IEEE, 2011.

[101] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *49th IEEE conference on decision and control (CDC)*, pages 7681–7687. IEEE, 2010.

[102] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[103] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *2011 IEEE international conference on robotics and automation*, pages 1478–1483. IEEE, 2011.

[104] Richard M Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer, 1972.

[105] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[106] Jeong-Jung Kim and Ju-Jang Lee. Trajectory optimization with particle swarm optimization for manipulator motion planning. *IEEE transactions on industrial informatics*, 11(3):620–631, 2015.

[107] Jongrae Kim and Joao P Hespanha. Discrete approximations to continuous shortest-path: Application to minimum-risk path planning for groups of UAVs. In *International Conference on Decision and Control*, volume 2, pages 1734–1740. IEEE, 2003.

[108] Bernhard Korte and Jens Vygen. *Combinatorial Optimization: Theory and Algorithms*, volume 21. Springer Berlin Heidelberg, 6th edition, 2018.

[109] Vince Kurtz and Hai Lin. Temporal logic motion planning with convex optimization via graphs of convex sets. *IEEE Transactions on Robotics*, 2023.

[110] Jean B Lasserre. Global optimization with polynomials and the problem of moments. *SIAM Journal on Optimization*, 11(3):796–817, 2001.

[111] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Kinodynamic motion planning for mobile robots using splines. In *International Conference on Intelligent Robots and Systems*, pages 2427–2433. IEEE/RJS, 2009.

[112] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. *TR 98-11, Computer Science Department, Iowa State University*, 1998.

[113] Der-Tsai Lee and Franco P Preparata. Euclidean shortest paths in the presence of rectilinear barriers. *Networks*, 14(3):393–410, 1984.

[114] Fajie Li and Reinhard Klette. *Euclidean shortest paths*. Springer, 2011.

[115] Guangzhi Li and Rahul Simha. The partition coloring problem and its application to wavelength routing and assignment. In *Proceedings of the First Workshop on Optical Networks*, volume 1. Citeseer, 2000.

[116] Wen-Jui Li, H-S Jacob Tsao, and Osman Ulular. The shortest path with at most/nodes in each of the series/parallel clusters. *Networks*, 26(4):263–271, 1995.

[117] Daniel Liberzon. *Switching in systems and control*, volume 190. Springer, 2003.

[118] Jyh-Ming Lien and Nancy Amato. Approximate convex decomposition of polygons. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 17–26, 2004.

[119] Ziang Liu, Genggeng Zhou, Jeff He, Tobia Marcucci, Fei-Fei Li, Jiajun Wu, and Yunzhu Li. Model-based control with sparse neural dynamics. *Advances in Neural Information Processing Systems*, 36, 2024.

[120] László Lovász and Alexander Schrijver. Cones of matrices and set-functions and 0–1 optimization. *SIAM Journal on Optimization*, 1(2):166–190, 1991.

[121] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[122] John Lygeros, Karl Henrik Johansson, Slobodan N Simic, Jun Zhang, and S Shankar Sastry. Dynamical properties of hybrid automata. *IEEE Transactions on automatic control*, 48(1):2–17, 2003.

[123] Anirudha Majumdar and Russ Tedrake. Funnel libraries for real-time robust feedback motion planning. *The International Journal of Robotics Research*, 36(8):947–982, 2017.

[124] Tobia Marcucci, Robin Deits, Marco Gabiccini, Antonio Bicchi, and Russ Tedrake. Approximate hybrid model predictive control for multi-contact push recovery in complex environments. In *International Conference on Humanoid Robotics*, pages 31–38. IEEE, 2017.

[125] Tobia Marcucci, Marco Gabiccini, and Alessio Artoni. A two-stage trajectory optimization strategy for articulated bodies with unscheduled contact sequences. *IEEE Robotics and Automation Letters*, 2(1):104–111, 2016.

[126] Tobia Marcucci, Manolo Garabini, Gian Maria Gasparri, Alessio Artoni, Marco Gabiccini, and Antonio Bicchi. Parametric trajectory libraries for online motion planning with application to soft robots. In *Robotics Research: The 18th International Symposium ISRR*, pages 1001–1017. Springer, 2020.

[127] Tobia Marcucci, Parth Nobel, Russ Tedrake, and Stephen Boyd. Fast path planning through large collections of safe boxes. *arXiv preprint arXiv:2305.01072*, 2023.

[128] Tobia Marcucci, Mark Petersen, David von Wrangel, and Russ Tedrake. Motion planning around obstacles with convex optimization. *Science Robotics*, 8(84):eadf7843, 2023.

[129] Tobia Marcucci and Russ Tedrake. Mixed-integer formulations for optimal control of piecewise-affine systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 230–239, 2019.

[130] Tobia Marcucci and Russ Tedrake. Warm start of mixed-integer programs for model predictive control of hybrid systems. *Transactions on Automatic Control*, 66(6):2433–2448, 2021.

[131] Tobia Marcucci, Jack Umenberger, Pablo Parrilo, and Russ Tedrake. Shortest paths in graphs of convex sets. *SIAM Journal on Optimization*, 34(1):507–532, 2024.

[132] R Kipp Martin. Using separation algorithms to generate mixed integer model reformulations. *Operations Research Letters*, 10(3):119–128, 1991.

[133] Rajesh Matai, Surya Prakash Singh, and Murari Lal Mittal. Traveling salesman problem: an overview of applications, formulations, and solution approaches. *Traveling salesman problem, theory and applications*, 1(1):1–25, 2010.

[134] Garth P McCormick. Computability of global solutions to factorable nonconvex programs: Part i—convex underestimating problems. *Mathematical programming*, 10(1):147–175, 1976.

[135] Daniel Mellinger and Vijay Kumar. Minimum snap trajectory generation and control for quadrotors. In *International Conference on Robotics and Automation*, pages 2520–2525. IEEE, 2011.

[136] Daniel Mellinger, Alex Kushleyev, and Vijay Kumar. Mixed-integer quadratic program trajectory generation for heterogeneous quadrotor teams. In *2012 IEEE International Conference on Robotics and Automation*, pages 477–483. IEEE, 2012.

[137] Nicholas Moehle and Stephen Boyd. A perspective-based convex relaxation for switched-affine optimal control. *Systems & Control Letters*, 86:34–40, 2015.

[138] Young-Soo Myung, Chang-Ho Lee, and Dong-Wan Tcha. On the generalized minimum spanning tree problem. *Networks*, 26(4):231–241, 1995.

[139] Vihangkumar V Naik and Alberto Bemporad. Embedded mixed-integer quadratic optimization using accelerated dual gradient projection. *IFAC-PapersOnLine*, 50(1):10723–10728, 2017.

[140] George L Nemhauser and Laurence A Wolsey. *Integer and combinatorial optimization*, volume 55. John Wiley & Sons, 1999.

[141] Charles E Noon and James C Bean. An efficient transformation of the generalized traveling salesman problem. *INFOR: Information Systems and Operational Research*, 31(1):39–44, 1993.

[142] Simeon Ntafos. Watchman routes under limited visibility. *Computational Geometry*, 1(3):149–170, 1992.

[143] Michael Otte and Nikolaus Correll. C-FOREST: Parallel shortest path planning with superlinear speedup. *IEEE Transactions on Robotics*, 29(3):798–806, 2013.

[144] Simone Paoletti, Aleksandar Lj Juloski, Giancarlo Ferrari-Trecate, and René Vidal. Identification of hybrid systems a tutorial. *European journal of control*, 13(2-3):242–260, 2007.

[145] Christos H Papadimitriou. An algorithm for shortest-path motion in three dimensions. *Information processing letters*, 20(5):259–263, 1985.

[146] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[147] Diego Pardo, Lukas Möller, Michael Neunert, Alexander W Winkler, and Jonas Buchli. Evaluating direct transcription and nonlinear optimization methods for robot motion planning. *IEEE Robotics and Automation Letters*, 1(2):946–953, 2016.

[148] Pablo A Parrilo. *Structured semidefinite programs and semialgebraic geometry methods in robustness and optimization*. California Institute of Technology, 2000.

[149] Pablo A Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96(2):293–320, 2003.

[150] Pablo A Parrilo and Sanjay Lall. Semidefinite programming relaxations and algebraic optimization in control. *European Journal of Control*, 9(2-3):307–321, 2003.

[151] Alejandro Perez, Sertac Karaman, Alexander Shkolnik, Emilio Frazzoli, Seth Teller, and Matthew R Walter. Asymptotically-optimal path planning for manipulation using incremental sampling-based algorithms. In *2011 IEEE/RSJ international conference on intelligent robots and systems*, pages 4307–4313. IEEE, 2011.

[152] Mark Petersen and Russ Tedrake. Growing convex collision-free regions in configuration space using nonlinear programming. *arXiv preprint arXiv:2303.14737*, 2023.

[153] Allen George Philip, Zhongqiang Ren, Sivakumar Rathinam, and Howie Choset. A mixed-integer conic program for the moving-target traveling salesman problem based on a graph of convex sets. *arXiv preprint arXiv:2403.04917*, 2024.

[154] Calder Phillips-Grafflin. Common robotics utilities.

[155] Erion Plaku, Kostas E Bekris, Brian Y Chen, Andrew M Ladd, and Lydia E Kavraki. Sampling-based roadmap of trees for parallel motion planning. *IEEE Transactions on Robotics*, 21(4):597–608, 2005.

[156] Petrica C Pop. *Generalized network design problems: Modeling and optimization*, volume 1. Walter de Gruyter, 2012.

[157] Michael Posa, Cecilia Cantu, and Russ Tedrake. A direct method for trajectory optimization of rigid bodies through contact. *The International Journal of Robotics Research*, 33(1):69–81, 2014.

[158] Abraham P Punnen, Piyashat Sripratak, and Daniel Karapetyan. The bipartite unconstrained 0–1 quadratic programming problem: Polynomially solvable cases. *Discrete Applied Mathematics*, 193:1–10, 2015.

[159] Anders Rantzer and Mikael Johansson. Piecewise linear quadratic optimal control. In *American Control Conference, 1997. Proceedings of the 1997*, volume 3, pages 1749–1753. IEEE, 1997.

[160] John H Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science*, pages 421–427. IEEE Computer Society, 1979.

[161] Arthur Richards and Jonathan P How. Aircraft trajectory planning with collision avoidance using mixed integer linear programming. In *Proceedings of the 2002 American Control Conference (IEEE Cat. No. CH37301)*, volume 3, pages 1936–1941. IEEE, 2002.

[162] R Tyrrell Rockafellar. *Convex analysis*. Number 28. Princeton University Press, 1970.

[163] Thomas Rothvoss. The matching polytope has exponential extension complexity. *Journal of the ACM (JACM)*, 64(6):1–19, 2017.

[164] Heinz Peter Rothwangl. Numerical synthesis of the time optimal nonlinear state controller via mixed integer programming. In *Proceedings of the 2001 American Control Conference.(Cat. No. 01CH37148)*, volume 4, pages 3201–3205. IEEE, 2001.

[165] Ricardo G Sanfelice. *Hybrid feedback control*. Princeton University Press, 2021.

[166] Tom Schouwenaars, Bart De Moor, Eric Feron, and Jonathan How. Mixed integer programming for multi-vehicle path planning. In *2001 European Control Conference*, pages 2603–2608. IEEE, 2001.

[167] Alexander Schrijver et al. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer, 2003.

[168] John Schulman, Yan Duan, Jonathan Ho, Alex Lee, Ibrahim Awwal, Henry Bradlow, Jia Pan, Sachin Patil, Ken Goldberg, and Pieter Abbeel. Motion planning with sequential convex optimization and convex collision checking. *The International Journal of Robotics Research*, 33(9):1251–1270, 2014.

[169] Hanif D Sherali and Warren P Adams. A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM Journal on Discrete Mathematics*, 3(3):411–430, 1990.

[170] Hanif D Sherali and Amine Alameddine. A new reformulation-linearization technique for bilinear programming problems. *Journal of Global optimization*, 2:379–410, 1992.

[171] Yasser Shoukry, Pierluigi Nuzzo, Alberto L Sangiovanni-Vincentelli, Sanjit A Seshia, George J Pappas, and Paulo Tabuada. Smc: Satisfiability modulo convex programming. *Proceedings of the IEEE*, 106(9):1655–1679, 2018.

[172] Skydio. Skydio 2+, 2022. Available at `https://www.skydio.com/skydio-2-plus-enterprise`.

[173] Eduardo Sontag. Nonlinear regulation: The piecewise linear approach. *IEEE Transactions on automatic control*, 26(2):346–358, 1981.

[174] Bartolomeo Stellato, Vihangkumar V Naik, Alberto Bemporad, Paul Goulart, and Stephen Boyd. Embedded mixed-integer quadratic optimization using the OSQP solver. In *European Control Conference*, pages 1536–1541. IEEE, 2018.

[175] David Stewart and Jeffrey C Trinkle. An implicit time-stepping scheme for rigid body dynamics with Coulomb friction. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 162–169. IEEE, 2000.

[176] Martin Stolle and Christopher G Atkeson. Policies based on trajectory libraries. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 3344–3349. IEEE, 2006.

[177] Robert A Stubbs and Sanjay Mehrotra. A branch-and-cut method for 0-1 mixed convex programming. *Mathematical Programming*, 86(3):515–532, 1999.

[178] Leonardo Taccari. Integer programming formulations for the elementary shortest path problem. *European Journal of Operational Research*, 252(1):122–130, 2016.

[179] Reza Takapoui, Nicholas Moehle, Stephen Boyd, and Alberto Bemporad. A simple effective heuristic for embedded mixed-integer quadratic programming. *International Journal of Control*, pages 1–11, 2017.

[180] Yuval Tassa, Tom Erez, and Emanuel Todorov. Synthesis and stabilization of complex behaviors through online trajectory optimization. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4906–4913. IEEE, 2012.

[181] Russ Tedrake. *Underactuated Robotics*. 2023.

[182] Russ Tedrake, Ian R Manchester, Mark Tobenkin, and John W Roberts. LQR-trees: Feedback motion planning via sums-of-squares verification. *The International Journal of Robotics Research*, 29(8):1038–1052, 2010.

[183] Russ Tedrake and the Drake Development Team. Drake: Model-based design and verification for robotics, 2019.

[184] Marc Toussaint. Logic-geometric programming: An optimization-based approach to combined task and motion planning. In *IJCAI*, pages 1930–1936, 2015.

[185] John N Tsitsiklis. Efficient algorithms for globally optimal trajectories. *IEEE Transactions on Automatic Control*, 40(9):1528–1538, 1995.

[186] Madeleine Udell, Karanveer Mohan, David Zeng, Jenny Hong, Steven Diamond, and Stephen Boyd. Convex optimization in Julia. In *2014 First Workshop for High Performance Technical Computing in Dynamic Languages*, pages 18–28. IEEE, 2014.

[187] Arjan J Van Der Schaft and Hans Schumacher. *An introduction to hybrid dynamical systems*, volume 251. springer, 2007.

[188] Mrinal Verghese, Nikhil Das, Yuheng Zhi, and Michael Yip. Configuration space decomposition for scalable proxy collision checking in robot planning and control. *IEEE Robotics and Automation Letters*, 7(2):3811–3818, 2022.

[189] Yang Wang and Stephen Boyd. Fast model predictive control using online optimization. *IEEE Transactions on control systems technology*, 18(2):267–278, 2009.

[190] Dustin J Webb and Jur Van Den Berg. Kinodynamic RRT*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 IEEE International Conference on Robotics and Automation*, pages 5054–5061. IEEE, 2013.

[191] Chin Wei-Pang and Simeon Ntafos. The zookeeper route problem. *Information Sciences*, 63(3):245–259, 1992.

[192] Peter Werner, Alexandre Amice, Tobia Marcucci, Daniela Rus, and Russ Tedrake. Approximating robot configuration spaces with few convex sets using clique covers of visibility graphs. *arXiv preprint arXiv:2310.02875*, 2023.

[193] Yang Yang, Mingen Lin, Jinhui Xu, and Yulai Xie. Minimum spanning tree with neighborhoods. In *International Conference on Algorithmic Applications in Management*, pages 306–316. Springer, 2007.

[194] Boyan Yordanov, Jana Tumova, Ivana Cerna, Jiří Barnat, and Calin Belta. Temporal logic control of discrete-time piecewise affine systems. *IEEE Transactions on Automatic Control*, 57(6):1491–1504, 2012.

[195] Xiaojing Zhang, Alexander Liniger, and Francesco Borrelli. Optimization-based collision avoidance. *IEEE Transactions on Control Systems Technology*, 29(3):972–983, 2020.

[196] Jianzhe Zhen, Danique de Moor, and Dick den Hertog. An extension of the reformulation-linearization technique to nonlinear optimization. *Available at Optimization Online*, 2021.

[197] Günter M Ziegler. *Lectures on polytopes*, volume 152. Springer Science & Business Media, 2012.

[198] Matt Zucker, Nathan Ratliff, Anca D Dragan, Mihail Pivtoraiko, Matthew Klingensmith, Christopher M Dellin, J Andrew Bagnell, and Siddhartha S Srinivasa. Chomp: Covariant Hamiltonian optimization for motion planning. *The International journal of robotics research*, 32(9-10):1164–1193, 2013.