On the Shortest Path Problem with Pair Constraints

Master's thesis

by

Michael Brückner

Michael Brückner
Matrikelnr.: 4366915
Ringstraße 16
14979 Großbeeren
m.brueckner@fu-berlin.de

Submission date:
February 25th 2015
Referees:
Prof. Dr. Ralf Borndörfer
Dr. Nam Dũng Hoàng

Abstract

This thesis investigates the *shortest path problem with pair constraints* or the *pair constraint problem* (PCP) for short. We consider two types of pair constraints, namely *forbidden pairs* and *binding pairs* consisting of two distinct vertices each. A path respects a forbidden pair if it uses at most one of the two vertices and it respects a binding pair (x, y) if it uses also y, if x is used.

Within this thesis, we bring together and compare several formulations and variants of the pair constraint problem and their complexities. We also collect existing recursive algorithms and present their running times. Most of the presented contributions only consider forbidden pairs. We introduce a new recursive algorithm also handling binding pairs and prove its theoretical complexity of $\mathcal{O}(n^4)$. We implemented the algorithm and tested it on real-world instances provided by Lufthansa Systems AG. Therefore we needed to develop a heuristic translating the real-world data into an instance of the shortest path problem with pair constraints. This heuristic is presented as well as all computational results.

In Chapter 4, we start investigating the associated polytope of an integer program formulation of the shortest path problem with pair constraints. For the case of one forbidden or binding pair, we find a complete linear description of the associated polytope. We prove that the number of facets grows exponentially in $|V|$ even in these simple cases. However, separation is still possible in polynomial time. The complete linear description can be extended to the case of *contiguously disjoint pairs*.

Eigenständigkeitserklärung

Hiermit erkläre ich, nachstehende Masterarbeit eigenständig und ohne Hilfe Dritter angefertigt zu haben. Alle Übernahmen aus der Literatur sind als solche gekennzeichnet und im Literaturverzeichnis auf Seite 74 aufgelistet.

Hereby I confirm that I developed the following Master's thesis entirely on my own without help of third parties. All adoptions from the literature are tagged as such and listed in the references section on Page 74.

Berlin, February 25, 2015

Michael Brückner

Acknowledgements

This thesis arose from the project Flight Trajectory Optimization on Airway Networks at the Zuse-Institute Berlin, where I worked as a student researcher. I would like to express my gratitude to the project team, namely Prof. Dr. Ralf Borndörfer, Marco Blanco, Dr. Nam Dũng Hoàng and Dr. Thomas Schlechte. They offered me the opportunity to work in their team and supervised me during my research and writing of this thesis. I want to emphasize the helpful assistance of Marco, who essentially promoted many of the results presented in this work and was always available for all the time-consuming problems and questions. I also want to thank my family and friends, who supported me writing this thesis.

Thanks to the Federal Ministry of Education and Research in Germany and the Lufthansa Systems AG, who supported this thesis.

Contents

1 Introduction

1.1 Motivation

Air traffic is one of the most important means of transport, especially in Europe. There are over 120.000 commercial flights a month only in Germany [7], which is more than 4.000 a day. Moreover, the air traffic increases every year by around 5% [1]. Air traffic is connected with very high costs such as fuel, personnel or fees e.g. for overflying a country.

Before the year 2000, flight planning was very static. This means, for example in Germany there were fixed routes for every pair of airports that could be looked up in a document containing the standard routes. Beginning around the year 2000, the philosophy came up that every trajectory is allowed. Since many airlines tended to optimize their flight routes, there were regions of very high traffic density. This was quite hard to handle for all the airlines and especially the agencies managing, regulating and controlling the air traffic. The solution was to distribute the traffic over the sky and to separate different traffic flows based on their direction, height or speed.

Mainly over Europe, this is achieved by settling a bunch of rules and restrictions to be respected during planning of the flight routes. The rules base on many different aspects of a flight, such as its departure or arrival, daytime, the season, the flight height or the usage of a concrete airspace or waypoint. Once a month, the European air traffic regulating organisation *Eurocontrol* publishes the so-called Route Availability Document (RAD) [11] of around 700 pages containing only these rules. Figure 1.1 shows how the page number of the RAD has grown in recent years.

Most of the current flight trajectory optimization software was developed before these rules are made. At this time, there were only a very few rules, which then could be involved manually or by simple methods. During the last years, the number of restrictions grew essentially to over 10.000. Whereas the previously mentioned rules are a good way distributing the air traffic over the sky, mathematically, they are very hard to involve in the optimization process of flight trajectories. It is already a hard problem to find just *any* route respecting all rules not to mention computing the *optimal* one. For this reason, computing the most cost-efficient flight trajectory is a mathematical problem of great intererest.

The project "Flight Trajectory Optimization on Airway Networks" is a research project funded by the Federal Ministry of Education and Research in collaboration with the Lufthansa Systems AG. It focuses on developing algorithms that

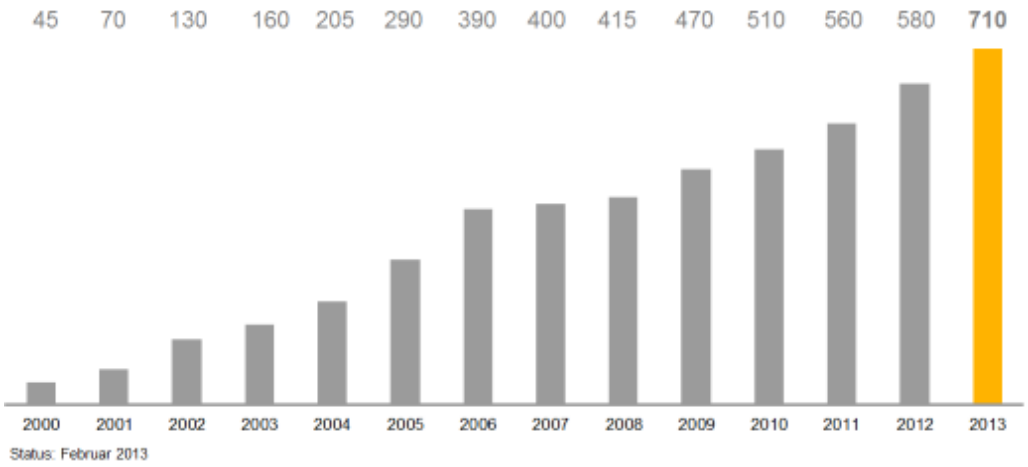

Figure 1.1: Average number of pages in the RAD per year. Source: Lufthansa Systems AG

compute optimal flight trajectories for passenger or cargo airplanes through an airway network, which is a discrete and finite graph spanned over the whole world. It contains around 90,000 points and around 400,000 directed connections between the points.

The project belongs to the joint project "E-Motion – energy-efficient mobility". In contrast to the computations in this airway network graph, another sub-project of E-Motion is located for instance at the Helmut-Schmidt-University in Hamburg. It is concerned with the same route optimization, but instead of a discrete graph, it deals with free flight. Whereas the airway network optimization is restricted to this network, free flight means that only origin and destination of the route are given; every trajectory connecting these two is allowed. The free flight system is used mostly over the oceans, whereas the restriction to the graph is usually used over mainland.

The optimal trajectory in both optimization ways is highly depending on several factors such as the weather conditions. Since there is no connection to the ground, airplane motion is not measured relatively to the ground, but to the surrounding air. Thus, wind has an essential effect on the actual "length" of a trajectory and hence on its costs. A similar influence holds for instance for the temperature and pressure of the air, but also for many other conditions. Beside weather conditions, there are many governmental restrictions influencing the price of flight trajectories as well. There are overflight fees for most countries, which have to be respected. These fees are calculated by different methods and not all of them are computationally easy to include in the complete optimization method.

Also relevant for the optimization of the trajectory are properties of the air-

craft itself, for instance the initial weight, the airplane type or the speed mode. The performance of aircrafts is given as tables determining how fast an airplane rises or how much fuel it uses depending on the mentioned weather conditions and its current weight. The conversion of these tables into usable data for the optimization algorithm is a non trivial mathematical problem for itself, see [25].

This thesis is concerned with the rules of the RAD document. These rules are of various formats and types and parsing them is highly not trivial. This is why we decided to focus on two particular cases, called *forbidden pair constraints* and *binding pair constraints*. Together, they are called *pair constraints* and they are the main mathematical concept of this thesis.

The problem of this thesis is the *shortest path problem with pair constraints*. We are given a directed graph $D = (V, A)$, a source node $s \in V$, a destination node $t \in V$ and two sets $\mathcal{F} \subseteq V \times V$ and $\mathcal{B} \subseteq V \times V$ of forbidden and binding pairs, respectively. The problem is to find the shortest path $\mathcal{P}$ through the graph connecting s and t and respecting all pair constraints. A forbidden pair $(f_1, f_2) \in \mathcal{F}$ is respected if the path either uses f_1 or f_2 or none of them, but not both. The path respects a binding pair $(b_1, b_2) \in \mathcal{B}$, if it also uses b_2 if b_1 is visited. Section 2.2 on page 11 states the problem formally. The next section illustrates the importance of this problem also for other disciplines than finding the shortest route in an airway network. Section 1.3 explains the golden thread of this thesis.

1.2 Literature Survey

The shortest path problem with pair constraints has a background in several fields of mathematics. Its origin seems to be the area of software testing, but as we will see, the problem became interesting for several other research areas. The technical contents of several of the following papers are presented in Section 2.2.3 on page 23. Table 1.1 summarizes the papers introduced here.

Krause, Smith and Goodwin [15] seem to be the first stating the problem of finding a shortest path avoiding forbidden pairs. We did not suffice to find their publication. According to Gabow, they formulated the forbidden pairs as arc pair constraints and called them impossible pairs. The problem had the shortcut IPP for the impossible pairs problem.

In 1976, Gabow, Maheshwari and Osterweil [12] showed the NP-completeness of IPP for vertex pair constraints by reduction of 3-SAT to IPP. Their purpose was automatic software testing. For a given software, they searched for a set of input parameter combinations in a manner that every part of the source code will be executed at least once by running the software for each of the chosen combinations. To keep this set small, they took architectural information of the source code into account. They built a directed graph from the source code: Every set of consecutive statements gets a vertex in the graph and if one of them executes the other, a directed arc is inserted. The impossible pairs modelled pairs

Year	Author	Origin	Contribution
1973	Krause et al. [15]	–	Formally introduced the problem
1976	Gabow et al. [12]	Software Testing	Proved NP-hardness
1979	Ntafos et al. [23]	Software Testing	Introduced Binding pairs NP-hardnes for $\mathcal{B}$
1997	Yinnone [27]		Skew symmetric graphs Polynomial complexity
2001	Chen et al. [8]	Comp. Biology	Uses PCP for Peptide Sequencing
2009	Kolman et al. [18]		halving, hierarchical structure Polynomial Complexity
2009	Kováč [21]	Bioinformatics	Gene Prediction using RP-PCR
2013	Kováč [20]	Bioinformatics	Several Constraint Structures and their Complexity

Table 1.1: Survey on the shortest path problem with pair constraints.

of code statements, which can't be executed within one software run. A typical example are `if` and `else` pairs.

Again coming from software industry, binding pairs are first introduced by Ntafos and Hakimi in 1979 [23]. They called them *must pairs* and the problem of determining whether there is a path respecting the binding pairs had the shortcut MUSTPR. MUSTPR is NP-complete as well; they gave a proof by reduction of the 3-SAT problem similar to the one of Gabow et al..

The first seemingly pure academical interest in the shortest path problem with vertex pair constraints came from Yinnone in 1997 [27]. Yinnone was the first finding a restriction of the problem making it solvable whithin polynomial time. During the paper, a skew symmetry condition was introduced. Further details to this condition can be found in Definition 2.19.

In 2001, Chen [8] extended the area of application of the shortest path problem with pair constraints to computational biology. They were studying the de novo peptide sequencing problem, whose goal is to reconstruct the peptide sequence from the result data of a tandem mass spectrometry. There are several ions in the sequence, for which it has to be determined whether they are so called N-terminal or C-terminal ions. The technique they published constructs a graph from the result data and with forbidden pairs, they assure that no ion is determined as

both N- and C-terminal.

The problem seems to have awaked academical interest in the past few years; in 2009, Kolman and Pangrác began subdividing the problem according to structural specialities of the constraint set $\mathcal{F}$. They distinguished two particular structures, namely *halving* and *hierarchical* structure. The problem remains NP-complete if $\mathcal{F}$ is halving, but in the hierarchical case they found a "surprisingly simple" polynomial algorithm, as they say. The algorithm will be presented in Section 3.1.2.

At the same time, the shortest path problem with pair constraints found another application in bioinformatics, namely gene prediction using RP-PCR tests. RP-PCR is a combination of biochemical methods, which is used to detect or quantify several DNA molecules. Kováč [21] investigated, how these RP-PCR results can be used for gene prediction, whose aim the finding of special genes is. Kováč constructed a graph and reduced the biochemics to an instance of the path-avoiding-forbidden-pairs problem (called PAFP). The following years, Kováč used for intensive studies on the PAFP problem, a systematical distinction of several cases of structures of the constraint set $\mathcal{F}$ and their hardnesses. These results are published in [20] and form the most valuable work on the PAFP problem.

1.3 Outline

The following chapter introduces the pair constraint problem. We start with some mathematical preliminaries followed by the formal statemant of the problem. We present results to the complexity as well as variants and structural conditions to the problem.

To the already known problem of paths avoiding forbidden pairs, there are plenty of combinatorial and recursive approaches, which will be presented from the literature in Chapter 3. The last algorithm we present is a new algorithm to also handle binding pairs (Section 3.2 on page 35).

Our second approach to solve the PCP concentrates on the integer program formulation of the problem. The problem can be easily formulated as an integer program. We tried to re-gain the linear relaxation by investigating the convex hull of all feasible solutions to the integer program. We present several new complete linear descriptions of small special cases of the problem in Chapter 4 on page 39.

The last aspect contains some computational results. We implemented the contraction algorithm we presented in Section 3.2 and tested it on real-world problem instances provided by Lufthansa Systems. The conversion of real-world data to instances of the PCP was highly non trivial. The chapter also explains the complexity of these data sets and how we managed it to convert them into problem instances of the shortest path problem with pair constraints. This chapter presents all computational results we gained by our implementation.

2 The Shortest Path Problem with Pair Constraints (PCP)

2.1 Preliminaries

This section captures some graph theoretical basics and lays the foundation of the notation used in the following pages. For the understanding of this thesis, we recommend basic knowledge in discrete mathematics and combinatorial optimization. The chosen notation follows mostly [22].

Definition 2.1 (Directed graph)**.** A directed graph D is a tuple $D = (V, A)$ with a finite vertex set V and a set of arcs $A \subseteq V \times V$. If it is not totally clear, which graph is meant, we clearify this using the notation $V(D)$ and $A(D)$ for the sets of vertices and arcs of a graph D. Usually we will use $n := |V|$ and $m := |A|$.

An arc (v_1, v_2) represents a directed edge from v_1 to v_2. We assume all graphs to be simple, i.e., there are no duplicate arcs and no loops (starting in and pointing to the same vertex). For an easy access to vertices and their neighbors, we define the following for each vertex and arc.

Definition 2.2 (δ^+ and δ^-, a^- and a^+)**.** Let $D = (V, A)$ be a directed graph and $v \in V$ a vertex. Then let $\delta^-(v) := (V \times \{v\}) \cap A$ and $\delta^+(v) := (\{v\} \times V) \cap A$. For every arc $a \in A$, let a^- refer to its tail and a^+ to its head, such that $a = (a^-, a^+)$ holds trivially.

The set $\delta^-(v)$ contains all incoming arcs of vertex $v \in V$ and $\delta^+(v)$ all outgoing ones. For the search after shortest paths in our graphs, we need a certain criterion for measuring length of paths. This measurement is typically called weight and defined as follows.

Definition 2.3 (Weight function)**.** Let D be a directed graph. The map $w : A \to \mathbb{R}_+$ is called a *weight function* on D.

A weight function w gives every arc a length, which makes it possible to compare different paths by the sum of their arc lengths. Let us define this formally:

Definition 2.4 (Path)**.** Let D be a directed graph and $s, t \in V$ two vertices. A path connecting s and t is a set $\mathcal{P} \subseteq A$ with

$$\mathcal{P} := \{a_1, \ldots, a_k\} = \{(s, v_1), (v_1, v_2), \ldots, (v_{k-1}, t)\}.$$

We also call $\mathcal{P}$ an s-t-path. Let $\mathcal{P}^v \subseteq V$ denote the set of vertices, $\mathcal{P}$ visits, i.e.,

$$\mathcal{P}^v = \{s, v_1, \ldots, v_{k-1}, t\}$$

Consequently, a path has the weight

$$w(\mathcal{P}) := \sum_{a \in \mathcal{P}} w(a).$$

Using the notion of paths, we can formalize when a vertex is reachable from another.

Definition 2.5 (Reachability of vertices)**.** Let D be a directed graph. Let $\prec$: $V \times V \to \{0, 1\}$ be a relation on V, such that $x \prec y$ holds if there exists a path from x to y.

If D contains no cycles, we call it *acyclic*. Let us for short assume, D is acyclic. In acyclic graphs, the situation $x \prec y$ and $y \prec x$ for two vertices $x, y \in V$ can only occur in the case $x = y$. This property is called *antisymmetry*. Under this condition, $\prec$ is called a *partial ordering* on V. Partial means in this case that not every two vertices are comparable in at least any of the two directions. Besides antisymmetry, a partial ordering has to be *reflexive* and *transitive*. Reflexive means $x \prec x$, which holds trivially for all vertices in our purpose. Transitivity means

$$a \prec b \wedge b \prec c \Rightarrow a \prec c$$

for every three vertices $a, b, c \in V$. This can be achieved by concatenating an a-b-path with an b-c-path, which shall both exist. This is an a-c-path verifying the transitivity of $\prec$. From set theory we know that we can extend every partially ordered set to a totally ordered set. In terms of graph theory, this extension of the reachability relation $\prec$ is called a topological sorting.

Definition 2.6 (Topological Sorting)**.** Let D be a directed graph. A total order $<$ on the vertex set V is called a *topological sorting* if for all arcs $(u, v) \in A$ holds $u < v$. Let us call vertices u and v *neighbours concerning* $<$ if either $x < u$ or $v < x$ holds for all vertices $x \in V \setminus \{u, v\}$.

Topological sortings are in general not unique if the graph contains no path traversing all vertices. The set theoretical argument about partial and total orderings can be reformulated in graph theoretical terms as follows.

Lemma 2.1 (Knuth et al. [16])**.** A topological sorting on a directed graph $D = (V, A)$ exists if and only if it is acyclic.

The proof of this lemma is of constructive nature; it gives an algorithm computing a topological sorting from a directed acyclic graph. This algorithm is relevant for this thesis at two points, namely for the proof of Proposition 2.2 and

for our own implementation, which is presented in section 5.1 on page 57. A detailed explanation of partial orderings and the proof can be found in [16] or in [24] (in German).

In Chapter 4, we often need to work with arc sets, which guarantee to intersect certain paths in a graph. This is why we define an arc cut.

Definition 2.7 (Arc cut)**.** Let $D = (V, A)$ be a directed graph and $C \subseteq A$ a set of arcs. C is called an *arc cut* of D if it disconnects D. Or in other words, if for every source s and every sink t in D and every s-t-path $\mathcal{P} \subseteq A$ there holds $\mathcal{P} \cap C \neq \emptyset$.

With these graph notations, we are now able to state one of the most common combinatorial problems, the *Shortest Path Problem*.

Problem 2.1 (Shortest Path Problem)**.** *Let D be a directed graph and $s, t \in V$ two different vertices in D. Let $w : A \to \mathbb{R}_+$ be a weight function on all arcs in D. The* shortest path problem *is to find a path $\mathcal{P} = \{a_1, \ldots, a_k\}$ from s to t such that $w(\mathcal{P})$ is minimal among all s-t-paths.*

The shortest path problem has been studied widely during the past seventy years. In 2005, the Center for discrete mathematics and theoretical computer science (called DIMACS) in New Jersey sponsored an implementation challenge concerning the shortest path problem. Afterwards, they published the book "The Shortest Path Problem" [10]. The first pages include a paper of Santos, which gives a great overview over the development of shortest path algorithms and other techniques for solving the problem. This is why we omit a detailed survey here and refer to this paper for further information.

The following integer program is a formulation of the shortest path problem.

Algorithm 1 Procedure `topSort` sorts a directed acyclic graph topologically.

Input: Graph $D = (V, A)$
Output: List S of topologically sorted vertices

```
procedure topSort(D)
    List S := ∅
    while |V| > 0 do
        Choose x ∈ V with δ⁻(x) = ∅
        Add x to S
        D := D − x
    return S
```

$$\begin{aligned}
&\text{maximize} && z = \sum_{a \in A} w_a x_a && && \\
&\text{subject to} && \sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = \begin{cases} 1 & i = s \\ -1 & i = t \\ 0 & \text{otherwise} \end{cases} && \forall i \in V && (2.1) \\
& && x_a \in \{0,1\} && \forall a \in A &
\end{aligned}$$

Equation (2.1) ensures that in every vertex the same amount of paths is incoming, as it leaves the vertex. For this reason, these are called the flow conservation constraints. Let the feasible set of this problem be denoted by P. Define

$$x(C) := \sum_{a \in C} x_a$$

for a subset $C \subseteq A$. To get an overview of how hard linear and integer programs actually are, let us consider some basics to the topic. Therefore we follow Nemhauser and Wolsey [22], a great introduction into the basics and advanced results of *integer and combinatorial optimization*. Another good overview with pronounciation on *integer optimization* can be found in [6]. All definitions, which we do not provide here can be found in these two books.

Definition 2.8 (Solution Set)**.** Let $P = \{x \in \mathbb{R}^n_+ \mid Ax \leq b\}$ be the solution set of a linear system

$$Ax \leq b. \tag{2.2}$$

P is called the *associated polytope* of (2.2) or the *feasible region* of (2.2). Let P now be nonempty. Then, P is called *integral* if all of its nonempty faces intersect $\mathbb{Z}^{|A|}$.

For a detailed introduction to polytopes and faces, see [28, 22].

Definition 2.9 (Total Unimodularity)**.** Let $A \in \mathbb{R}^{m \times n}$ be a matrix. A is called *totally unimodular* if the determinant of every square submatrix of A equals -1, 0 or 1.

Proving that a matrix is *not* totally unimodular can be done by simply giving a square matrix with determinant not in $\{-1, 0, 1\}$. But showing that a matrix is totally unimodular is quite hard, since the number of square submatrices grows exponentially in the size of the matrix. This is why the problem of determining whether a matrix is totally modular is Co-NP.

Lemma 2.2 (Nemhauser et al. [22])**.** Let $A \in \mathbb{R}^{m \times n}$ be a matrix and $P(b) = \{x \in \mathbb{R}^n_+ \mid Ax \leq b\}$ be the associated polytope depending on b. If A is totally unimodular, then $P(b)$ is integral for every $b \in \mathbb{Z}^m$ if it is not empty.

This has an immense meaning to integer programming. Consider an integer program

$$z_{\text{IP}} = \max\{cx \mid Ax \leq b\, x \in \mathbb{Z}_{\geq 0}^{|A|}\} \tag{2.3}$$

with a totally unimodular matrix A. By the last lemma, independently of b and c, if z_{IP} is finite, there is always an optimal integral solution $x^* \in \mathbb{Z}_+^n$ with $cx^* = z_{\text{IP}}$ as long as b is integral. There is no known algorithm solving an integer program in polynomial time. In contrast to that, this is the case for linear programs by for example the ellipsoid method. This means, if the matrix of our integer program is totally unimodular, we can solve the problem in polynomial time by relaxing the integrality condition and solving the corresponding linear programm. This linear program is called the *LP-relaxation* of the integer program (2.3).

Let

$$\min\left\{wx \mid Ax \leq b\, x \in \mathbb{Z}_{\geq 0}^{|A|}\right\}$$

be the integer program formulation of the shortest path problem as stated in (2.1). Then, A is the incidence matrix of the graph D and b has a 1 in the row of s, a -1 in the row of t and 0 everywhere else. The following lemma now comes without surprise:

Lemma 2.3 (Bertsimas et al. [6])**.** The incidence matrix A of a directed acyclic graph is totally unimodular.

2.2 Problem Statement

From now on, let all graphs considered in this thesis be acyclic. Let us now consider several types of constraints for a shortest path problem.

Definition 2.10 (Forbidden Pair)**.** A forbidden pair $(\ell, r) \in V \times V$ is a pair of two vertices. A path $\mathcal{P}$ respects the forbidden pair (ℓ, r) if

$$\ell \notin \mathcal{P}^v \vee r \notin \mathcal{P}^v$$

holds. This means, $\mathcal{P}$ uses at most one of the two vertices ℓ and r.

Definition 2.11 (Binding Pair)**.** A binding pair $(\ell, r) \in V \times V$ is again a pair of two vertices and a path $\mathcal{P}$ respects the binding pair (ℓ, r) if

$$\ell \in \mathcal{P}^v \Rightarrow r \in \mathcal{P}^v$$

is satisfied. In other words, it also visits r if ℓ is used.

Definition 2.12 (Obligatory Pair)**.** An obligatory pair $(\ell, r) \in V \times V$ is respected by a path $\mathcal{P}$ if

$$\ell \in \mathcal{P}^v \vee r \in \mathcal{P}^v$$

holds. Thus, at least one of the two vertices ℓ or r is used by $\mathcal{P}$.

Although obligatory pairs are mentioned here, we only consider forbidden and binding pairs during this thesis, see Section 5.2 on page 58 for detailed reasons. From now on, $\mathcal{F}, \mathcal{B} \subseteq V \times V$ shall denote the sets of forbidden and binding pairs. $\mathcal{C}$ is always the union of those, $\mathcal{C} := \mathcal{F} \cup \mathcal{B}$. For better access to the individual constraints, let us denote $\mathcal{C} = \{(\ell_1, r_1), \ldots, (\ell_k, r_k)\}$ and for individual constraints the set $\mathcal{C}_i := \{(\ell_i, r_i)\}$. Let the same notation also hold for $\mathcal{F}$ and $\mathcal{B}$: $\mathcal{F}_i := \{(\ell_i, r_i)\}$ denotes the i-th forbidden pair and $\mathcal{B}_i := \{(\ell_i, r_i)\}$ the i-th binding pair. As stated here, all three constraint types are vertex pairs. Section 2.2.2 on page 21 is concerned with an analogous formulation using arc pairs. All technical issues concerning the choice of vertex or arc pairs can be found there. The following definition is picked from [27].

Definition 2.13 (Yinnone [27])**.** Let $\mathcal{C}$ be a constraint set on any graph D. We assume that every vertex is contained in at most one constraint in $\mathcal{C}$. Then we define v' for every vertex $v \in V$ as follows:

1. $v' = v$ if v is not contained in any constraint in $\mathcal{C}$ or
2. $v' = w$ if $(v, w) \in \mathcal{C}$ or $(w, v) \in \mathcal{C}$.

We call v' the *mate* of v if $v' \neq v$. Otherwise we call v *isolated with respect to* $\mathcal{C}$.

The combination of the previously mentioned shortest path problem and these new constraints is the key problem in this thesis.

Problem 2.2 (Shortest path problem with pair constraints)**.** *Let $D = (V, A)$ a directed acyclic graph and let $s, t \in V$ be a source and a sink. Let further $\mathcal{F} \subseteq V \times V$ be a set of forbidden pairs and $\mathcal{B} \subseteq V \times V$ a set of binding pairs. The* shortest path problem with pair constraints *is to find a shortest path $\mathcal{P} = \{a_1, \ldots, a_l\}$ such that*

$$\ell \notin \mathcal{P}^v \vee r \notin \mathcal{P}^v$$

holds for all $(\ell, r) \in \mathcal{F}$ and

$$\ell \in \mathcal{P}^v \Longrightarrow r \in \mathcal{P}^v$$

holds for all $(\ell, r) \in \mathcal{B}$. Let Pair constraint Problem *be a shorter name for the shortest path problem with pair constraints and accordingly* PCP *be the abbreviation for it.*

We assume that s and t do not belong to any forbidden or binding pair. Otherwise, we could easily get rid of such a pair by altering the graph. If for example s is contained in a forbidden pair, s' can never be on a feasible s-t-path and hence, we can remove s' from D as well as the forbidden pair (s, s') from $\mathcal{F}$. This problem can be formulated as an integer program simply by extending the existing formulation of the basic shortest path problem.

Problem 2.3. *The shortest path problem with pair constraints can be modelled by the following integer program:*

$$
\begin{aligned}
\text{maximize} \quad & z = \sum_{a \in A} w_a x_a \\
\text{subject to} \quad & \sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = \begin{cases} 1 & i = s \\ -1 & i = t \\ 0 & \text{otherwise} \end{cases} && \forall i \in V \\
& \sum_{a \in \delta^+(i)} x_a + \sum_{a \in \delta^+(j)} x_a \leq 1 && \forall (i,j) \in \mathcal{F} \\
& \sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^+(j)} x_a \leq 0 && \forall (i,j) \in \mathcal{B} \\
& x_a \geq 0 && \forall a \in A.
\end{aligned}
$$

Let the feasible set of this problem be denoted by $P(\mathcal{C})$, whereas P is the feasible set of the underlying unconstrained shortest path problem.

Note that the integer program formulation of the shortest path problem had as matrix the incidence matrix of the underlying graph. This was why the unimodularity of the incidence matrix directly made the problem easy to solve. To take the forbidden and binding pairs into account, we need additional constraints to the feasible set of the integer program. This leads to additional rows in the matrix of the integer program. The price for this purpose is the loss of the total unimodularity. The following example shows a counterexample.

Example 2.1. Let $V = \{s, u, v, t\}$ and $A = \{a_1, a_2, a_3\} = \{(s,u), (u,v), (v,t)\}$. With $D = (V, A)$ and $\mathcal{F} = \{(u, v)\}$, we have a pair constraint problem (let $\mathcal{B} = \emptyset$ here). The matrix of the pair constraint problem integer program formulation is:

$$
A = \begin{array}{c} \\ s \\ u \\ v \\ t \\ (u,v) \end{array}
\begin{array}{c}
\begin{array}{ccc} a_1 & a_2 & a_3 \end{array} \\
\begin{pmatrix} -1 & 0 & 0 \\ \boxed{1} & \boxed{\text{-1}} & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \\ \boxed{1} & \boxed{1} & 0 \end{pmatrix}
\end{array}.
$$

The four highlighted entries form a 2×2 submatrix with determinant 2. Thus, this matrix is not totally unimodular.

After the insights concerning the integer program formulation of the PCP, there are several ways to approach the problem. The pair constraints intuitively relate to the logical `or`, since all constraints can be modelled as an `or` condition:

- A forbidden pair $(f_1, f_2) \in \mathcal{F}$ is respected if either f_1 is not used `or` f_2 is not used.

- A binding pair $(b_1, b_2) \in \mathcal{B}$ is respected if either b_2 is used `or` b_1 is not used.
- An obligatory pair $(o_1, o_2) \in \mathcal{O}$ is respected if either o_1 is used `or` o_2 is used.

Generally, all constraints in the theory of linear programming are combined with an `and`. This motivates the usage of *disjunctive programming* [3], a technique enabling the usage of logical `or` between constraints of a linear program. This gives now rise to several formulations of problems using `and` and `or`. In the following, we will use the symbols $\vee$ for the logical `or` and $\wedge$ for the logical `and`. There are two normal forms called *conjunctive normal form* (CNF) and *disjunctive normal form* (DNF). A disjunctive program in CNF has the feasible set

$$F_c = \bigcap_{i \in I} \left(\bigcup_{j \in I_i} \{x \in \mathbb{R}^n \mid a_j x \leq b_j\} \right),$$

where for each I_i, at least one of the constraints $a_j x \leq b_j$ with $j \in I_i$ has to be fulfilled. In general, $\cap$ and $\cup$ are the set theoretic equivalents to $\wedge$ and $\vee$. In contrast to that, the feasible set of a DNF looks as follows:

$$F_d = \bigcup_{i \in I} \{x \in \mathbb{R}^n \mid A_i x \leq b_i\}$$

In this case, for at least one $i \in I$, all of the constraints $A_i x \leq b_i$ have to be satisfied. Note that this contains a hidden $\wedge$, which leads to the desired symmetry between CNF and DNF.

The shortest path problem with vertex pair constraints clearly relates to a conjunctive normal form, since every pair constraint has to be satisfied individually ($\wedge$), but internally, there are two possibilities to achieve that ($\vee$). Unfortunately, Balas [4] needs a disjunctive program in DNF for the conversion into a linear program. If we reformulate our problem to a DNF, the outer disjunction would combine a set of $2^{|\mathcal{C}|}$ linear programs. Each of these linear programs represents one possibility, how to satisfy all the constraints in $\mathcal{C}$. This is comparable with a brute-force approach to find a path satisfying $\mathcal{C}$. Since all the general constraints of the pure shortest path problem have to be copied in all these linear programs, the resulting linear program will grow into immense dimensions. This is why we avoided the disjunctive programming approach.

2.2.1 Complexity

There are two main complexity results. Both results concern the path-avoiding-forbidden-pairs problem, called PAFP. These results and their proofs easily extend to the PCP problem due to the following observation. We will restate the both results from the literature and then conclude for PCP.

Observation 2.1. An instance x of the PAFP problem can be reduced to an instance y of shortest path problem with pair constraints. The size of y is polynomial in the size of x.

This can be easily achieved by setting y exactly as x with $\mathcal{B} = \emptyset$. Gabow et al. [12] gained in 1976 the following result concerning the PAFP. The results of Gabow base on forbidden vertex pairs $\mathcal{F} \subseteq V \times V$. For a detailed introduction to complexity theory, see [2]. Especially Sections 1.2 and 1.3 focus on NP and NP-completeness.

Theorem 2.1 (Gabow et al. [12])**.** *The PAFP problem is NP-Complete.*

Proof. Let us first show that PAFP can be solved by a nondeterministic turing machine and hence lies in NP. Consider the following procedure 2 `detPath`. The method is a kind of brute-force. It chooses a set J of forbidden pairs. All vertices ℓ of a forbidden pair $(\ell, r) \in J$ are removed from D in D'. This ensures that from these forbidden pairs, only r is used if any. For the remaining pairs, r is removed. Thus, the graph D' allows per forbidden pair only one of the two vertices ℓ or r, depending on J and hence, it only returns `true` if a feasible path exists. Conversely, if there is a feasible path $\mathcal{P}$, then it uses at most one vertex per forbidden pair. Hence, there is at least one set J, for which $\mathcal{P}$ is fully contained in D'. Hence, `detPath` will finds a path if it exists.

Algorithm 2 Procedure `detPath` finds a path respecting all forbidden pairs. [12]

Input: Graph $D = (V, A)$, forbidden pairs $\mathcal{F}$
Output: Boolean value, wether a feasible path exists

procedure DETPATH(D, $\mathcal{F}$)
  for all subsets $J \subseteq \mathcal{F}$ **do**
    $S := \{l \mid (l, r) \in J\} \cup \{r \mid (l, r) \in \mathcal{F} \setminus J\} \subseteq V$
    $D' := D - S$
    if $\exists$ s-t-path in D' **then return** true
  end for
  return false

The `detPath` routine uses $O(|E| \cdot 2^{|\mathcal{F}|})$ asymptotic time. The $2^{|\mathcal{F}|}$ term arises by the iteration over all subsets of $\mathcal{F}$. Once, the correct choice for J is found, the procedure needs $O(|E|)$ asymptotic time in the second step to find a path. This can be achieved for example via depth-first-search. If we see `detPath` as a nondeterministic procedure (i.e., it cleverly guesses instantly a proper J), the running time reduces to $O(|E|)$, which is polynomial and hence PAFP is in NP.

Now, we need to prove that PAFP is NP-hard (i.e., belongs to the hardest problems in NP). We do this by reduction of an NP-hard hard problem, 3-SAT, to PAFP.

Problem 2.4 (3-SAT)**.** *The* 3-Satisfiability problem *is to determine whether a Boolean expression B in 3-conjunctive normal form is satisfiable. Let B be of the form*

$$B := \bigwedge_{i=1}^{m} (p_{r1} \vee p_{r2} \vee p_{r3}).$$

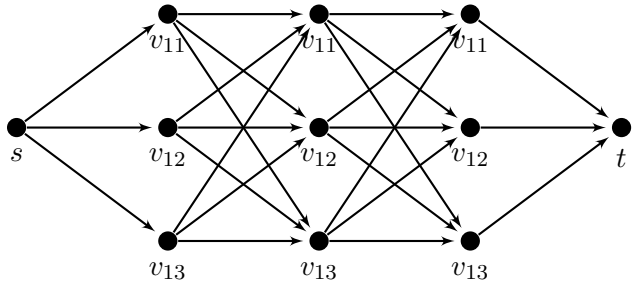

Figure 2.1: The graph constructed from a Boolean expression with $m = 3$ [12]

Each literal p_{ij} represents either a Boolean variable x_k or its negation $\bar{x}_k$. B is now satisfiable if there is an assignment of Boolean values to the variables x_k, such that B evaluates to true.

We now find a reduction of 3-SAT to PAFP, which makes it possible to solve an instance of 3-SAT by constructing a PAFP instance, solving this and then concluding back to the solution of the initial 3-SAT instance. Let G be the constructed graph. The vertex set V constists of source and sink vertices s and t and vertices v_{ij} for all literals p_{ij} in B. The arc set A is defined as

$$\begin{aligned} A := &\{(s, v_{1j}) \mid 1 \leq j \leq 3\} \\ &\cup \{(v_{ij}, v_{i+1,k} \mid 1 \leq i < m, 1 \leq j, k \leq 3\} \\ &\cup \{(v_{mj}, t) \mid 1 \leq j \leq 3\}. \end{aligned}$$

See Figure 2.1 for an illustration with $m = 3$. A shortest s-t-path P in this graph shall choose one of the literals in each clause evaluating to true. To prevent P from taking two contradicting literals of the same variable, we define all those pairs as forbidden. That is,

$$\mathcal{F} := \{(v_{ij}, v_{kl}) \mid p_{ij} = \bar{p}_{kl}\}.$$

Consider now an s-t-path P respecting $\mathcal{F}$. Let P traverse the following vertices:

$$s, v_{1l_1}, v_{2l_2}, \ldots, v_{ml_m}, t$$

We now set the variables x_k in a way that all the literals p_{il_i} are true. Because P respects $\mathcal{F}$, this is possible without any conflicts. Hence, B is satisfiable if such a path P exists. Analogously, if there is no path P, B is not satisfiable. This can be seen via contraposition. □

We now saw that 3-SAT can be solved using PAFP. If PAFP is polynomially solvable, then 3-SAT also will be polynomially solvable and therefore not be NP-hard, because the reduction from 3-SAT to PAFP was also polynomially. This is not the case and hence PAFP is NP-hard. Together with the previous result that PAFP is in NP, we have that PAFP is NP-complete.

Corollary 2.1. PCP is NP-complete.

Proof. Since PAFP is trivially reducable to PCP, the only point left for proving is that PCP actually lies in NP. Take a procedure enumerating all s-t-paths and then checking them for validity. The validity check runs in linear time in $|\mathcal{C}| \cdot |V|$, since it only runs through the vertices used by the path and checks the individual conditions of forbidden and binding pairs. Hence, this procedure is nondeterministically polynomial and therefore PCP lies in NP. □

The second part of this section deals with a completely different set of complexity classes. The previous considerations all dealed with algorithms *exactly* solving a problem. A similar classification for the complexity of algorithms exists depending on how good can optimal solutions be approximated. [2] gives a good introduction to plenty of approximative algorthms as well as their complexity classes. We will shortly explain the necessary terms and thereby follow the mentioned book.

Definition 2.14 (Performance Ratio [2])**.** Let $\mathfrak{P}$ be an optimization problem, x an instance of $\mathfrak{P}$. We define the *performance ratio* of a feasible solution y of x as

$$R(x, y) = \max\left(\frac{m(x, y)}{m^*(x)}, \frac{m^*(x)}{m(x, y)}\right),$$

where $m^*(x)$ denotes the value of an optimal solution of x and $m(x, y)$ the value of solution y to problem instance x.

By this definition, a performance ratio of r for solution y means in fact

$$m(x, y) \in \left[\frac{1}{r} m^*(x), r \cdot m^*(x)\right],$$

since r is a value not smaller than 1.

Definition 2.15 (Ausiello [2])**.** Let $\mathfrak{P}$ be an optimization problem and $\mathcal{A}$ an approximation algorithm solving $\mathfrak{P}$. Let $r \in \mathbb{R}_+$ be a bound, such that

$$R(x, \mathcal{A}(x)) \leq r$$

for all instances x. Then we call $\mathcal{A}$ an r-*approximate algorithm*.

Using this classification of optimization problems, we can now define the complexity class APX.

Definition 2.16. The complexity class of all problems $\mathfrak{P}$, for which there is an $r \geq 1$ and an r-approximate algorithm, is called APX.

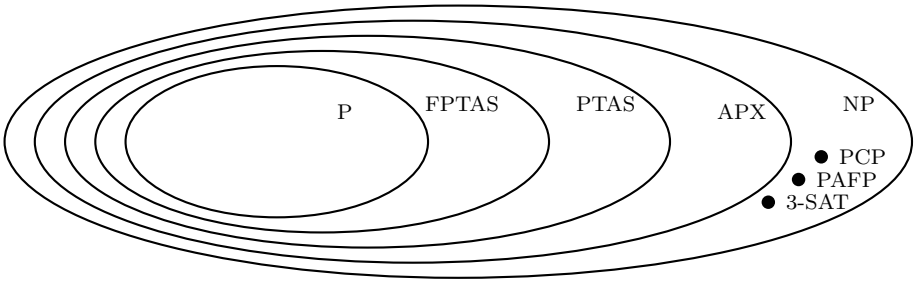

Figure 2.2: The complexity classes for approximating algorithms.

Figure 2.2 illustrates that APX is the largest class of approximatively solvable problems. Thus, not belonging to APX means beeing one of the hardest problems to solve approximatively. See [2] for definitions of FPTAS and PTAS.

Let us now consider the following problem MAX-REP. Its first occurence was in 1999 in [19]. This paper of Kortsarz was about finding k-spanners of a graph. A k-spanner is a subgraph G' of $G = (V, E)$ with vertex set V and a subset of the edges E, where the distance between any two vertices u and v in G' is not larger than k times the distance between u and v in G. The paper shows that finding k spanners with a number of edges close to the optimum is at least as hard as approximating the set cover problem.

Problem 2.5. *Let $A_1, \ldots, A_k$ and $B_1, \ldots, B_k$ all pairwise disjoint sets of vertices and let*

$$V_1 := \bigcup_{i=1}^{k} A_i \quad \text{and} \quad V_2 := \bigcup_{i=1}^{k} B_i.$$

The sets A_i and B_i all have size n. Let further $G = (V_1, V_2, E)$ be an undirected bipartite graph. We now construct a super-graph $\mathcal{H} = (V_\mathcal{H}, E_\mathcal{H})$ with $V_\mathcal{H} = \{A_1, \ldots, A_k, B_1, \ldots, B_k\}$. Two vertices A_i and B_j in $V_\mathcal{H}$ are adjacent if there is an edge in G between any two vertices $a \in A_i$ and $b \in B_j$. Since G is bipartite, $\mathcal{H}$ is bipartite as well.

The problem MAX-REP *is about of finding exactly one representative $a_i \in A_i$ and $b_j \in B_j$. A super-edge (A_i, B_j) is said to be covered if the representatives a_i and b_j have an edge in G, i.e. $(a_i, b_j) \in E$. The problem is to find representatives maximizing the number of covered super-edges in $\mathcal{H}$.*

Figure 2.3 contains an example instance with $n = 3$ and $k = 2$. The red diamonds are a possible choice of representatives solving the MAX-REP instance. In this case, all super-edges

$$\{(A_1, B_2), (A_2, B_1), (A_2, B_2)\}$$

are covered. The crux of this problem is that MAX-REP is not in APX. The proof of this goes by reduction of SAT to MAX-REP. It is published in 2001 [14]. To

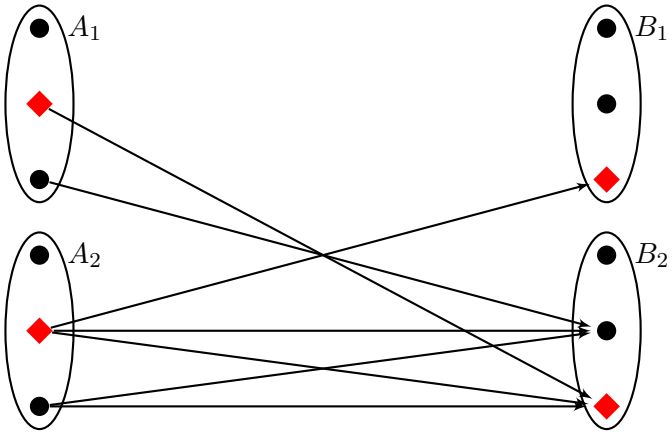

Figure 2.3: An instance of the MAX-REP problem. The red diamonds cover all super-edges.

avoid getting of the track, we omit the proof here and focus on the next theorem and its proof. At this point, we need to mention that the following results are concerned with a variant of the PAFP problem, namely the minimization of the number of violated forbidden pairs PAFP'.

Theorem 2.2 (Hajiaghayi [13])**.** *PAFP' is not in APX.*

The proof of this theorem goes by reduction as well, this time we reduce MAX-REP to PAFP'. The proof was originally published in [13].

Proof. Let G, A_i, B_j and $\mathcal{H}$ be a given instance of the MAX-REP problem. Let $X_1, \ldots, X_{2k}$ be an arbitrary enumeration of the vertices of $\mathcal{H}$, i.e. the sets A_i and B_j. We now create the directed acyclic graph $D = (V, A)$ with vertex set

$$V := \{s\} \cup \left(\bigcup_{i=1}^{2k} X_i \right) \cup \{t\}.$$

In fact, except for s and t, this are exactly the vertices of G. This definition shall pronounce the ordering of the vertices according to the sets X_i as illustrated in Figure 2.4. The red diamonds are the same as in the previous Figure 2.3. The arc set of D is

$$A := \{(s, x) \mid x \in X_1\} \cup \bigcup_{i=1}^{2k-1} \{(x, y) \mid x \in X_i, y \in X_{i+1}\} \cup \{(x, t) \mid x \in X_{2k}\},$$

which consists of a complete bipartite graph between consecutive super-vertices X_i and X_{i+1} and connections to s and t. The last point are now the forbidden pairs

$$\mathcal{F} := \{(f_1, f_2) \mid f_1 \in A_i, f_2 \in B_j, (f_1, f_2) \notin E, (A_i, B_j) \in E_{\mathcal{H}}\}.$$

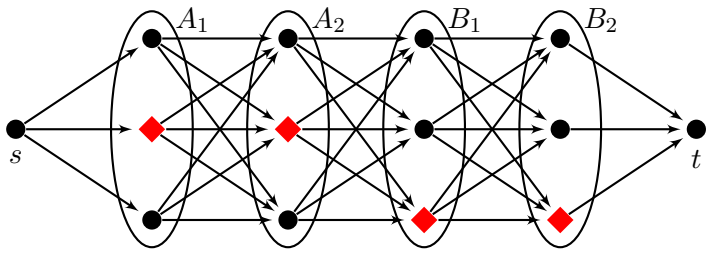

Figure 2.4: Transformation of the MAX-REP instance (Figure 2.3) to PAFP'.

By this definition, forbidden pairs correspond to pairs of vertices which are *not* adjacent in G, but there super-vertices *are* in $\mathcal{H}$. The size of this new PAFP' instance is polynomially in the size of the given MAX-REP instance.

Clearly, there is a one-to-one correspondence between a set of representatives for the MAX-REP instance and a path through the graph of the PAFP' instance, since they use the same vertices. Let now $R \subset V(G)$ be a choice of representatives in G and $\mathcal{P}$ be the corresponding path in D. We now prove that if $\mathcal{P}$ violates t forbidden pairs, then R covers $h-t$ super-edges, where $h = e(\mathcal{H})$ is the number of super-edges in $\mathcal{H}$. In other words, minimizing the number of violated forbidden pairs is equivalent to maximizing the number of covered super-edges and hence an optimal solution of PAFP' relates to an optimal solution of MAX-REP and vice-versa.

In fact, a violated forbidden pair (a_i, b_j) occurs in $\mathcal{P}$ if a_i and b_i are not connected in G, but A_i and B_j are connected in $\mathcal{H}$ and thus $\{A_i, B_j\}$ is a violated super-edge. Conversely, if a super-edge $\{A_i, B_j\}$ is not covered in $\mathcal{H}$, then $\mathcal{P}$ chooses two vertices $a \in A_i$, $b \in B_j$ which are not adjacent in G and hence (a, b) is a forbidden pair violated by $\mathcal{P}$.

We saw that MAX-REP can be reduced to PAFP' and the solution of PAFP' returns information to the solution of MAX-REP. This means PAFP' is computationally harder than MAX-REP and since MAX-REP is not in APX, PAFP' is also not. □

We can now conclude to the variant PCP' of the pair constraint problem, which minimizes the number of violated pair constraints.

Corollary 2.2. The PCP' does not lie in APX as well.

We have now seen two results concerning the complexity of the shortest path problem with pair constraints. The first one states that it is NP-complete and hence belongs to the hardest problems to solve exactly. The second one is about how good the optimal solution can be approximated and we saw that even in this case, we are in the hardest possible complexity. Thus, the pair constraint problem counts to the hardest problems.

2.2.2 Problem Variants

Most of the literature we studied decided to define the pair constraints $\mathcal{B}$ and $\mathcal{F}$ as vertex pair constraints. This is why we followed them and did it as well. Nevertheless, there are arc pair formulations as well in the literature. This section addresses these two variants.

Theorem 2.3. *An instance of the shortest path problem with vertex pair constraints can be transformed into an instance of the shortest path problem with arc pair constraints. Thereby the instance grows by at most $2k$ vertices and at most $2k$ arcs, where $k = |\mathcal{C}|$ is the number of constraints in the instance. The same holds conversely.*

Proof. We restrict the proof to the transformation of vertex pairs into arc pairs. The opposite direction works similar. The core of the transformation is the operation `splitVertex`, which is given in pseudo code in Algorithm 3. The two for loops in the code reorganize the incoming and outgoing arcs of v. Afterwards, the incoming arcs point to v^- and the outgoing arcs start in v^+. v is now isolated and can be removed. The procedure alters the graph; this means for an implementation that the graph should be called by reference or as a pointer. Figure 2.5 illustrates it with an example. The figure also shows for the opposite direction, how an arc can be divided into two arcs separated by a vertex.

We now apply the method `splitVertex` to every vertex contained in a vertex pair constraint. Then, we define the new arc pair constraints with the arcs `splitVertex` returns. Clearly, the method adds exactly one new vertex and one new arc for every splitted vertex, which explains the increase of at most $2k$ vertices and arcs, since every vertex pair constraint consists of two distinct vertices. □

Algorithm 3 Procedure `splitVertex` replaces a vertex by an arc.

Input: Graph $D = (V, A)$, vertex v
Output: The arc, in which v was splitted.

procedure splitVertex(D, v)
  Add new vertices v^- and v^+ to D
  for $(x, v) \in A$ **do**
    Change arc to (x, v^-) in D
  for $(v, y) \in A$ **do**
    Change arc to (v^+, y) in D
  Add new arc (v^-, v^+) to D
  Remove v from D
  return (v^-, v^+)

This allows the conclusion that for problem instances with $|\mathcal{C}| \in o(|V|)$, we can consider the both description variants with vertex pairs and respectively with arc

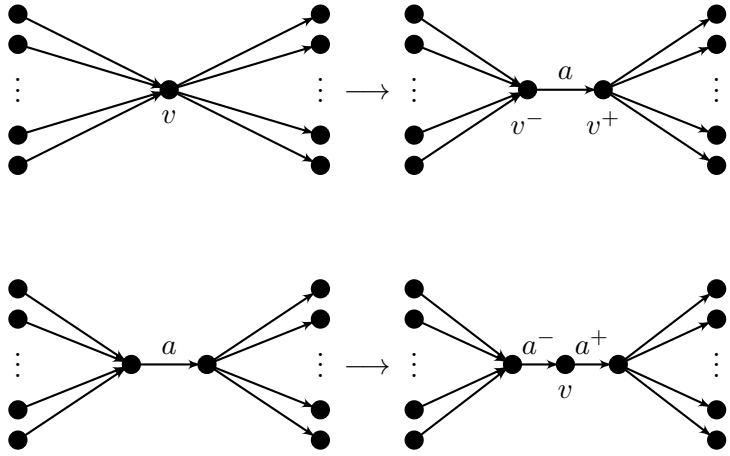

Figure 2.5: Splitting a vertex to an arc (above) and vice-versa (below).

pairs as almost equally hard. But the similarity of the two goes a step further. As we will see in Section 2.2.3, reachability plays an important role for the relations between the pair constraints. We already defined vertex reachability; here is the arc equivalent.

Definition 2.17 (Reachability of arcs [26])**.** Let D be a directed acyclic graph. Let $\prec_A$: $A \times A \to \{0, 1\}$ be a relation on A, such that $a_1 \prec_A a_2$ holds if there exists a path using first a_1 and later a_2.

Note that both operations *increase* the size of the instance and concatenating them will not result in the original instance, although they can be identified by simple preprocessing techniques. The following proposition ensures that both operations do not change reachability among the pair constraints.

Proposition 2.1. *Let D be a directed acyclic graph and $\mathcal{C} \subseteq V \times V$ a pair constraint set. Let D' and $\mathcal{C}' \subseteq A \times A$ be the instance obtained by* `splitVertex`*. Then, we have*

$$x \prec y \Leftrightarrow (x^-, x^+) \prec_A (y^-, y^+)$$

for all vertices $x, y \in V$ beeing contained in any forbidden pair.

Proof. For the direction "⇒", let $x \prec y$ hold for any two properly chosen vertices. Let $\mathcal{P}^v = \{x = p_1, \ldots, p_s = y\}$ be the vertex set of a path connecting x and y. By the operation, a new vertex x^+ is generated and with it a new arc (x^+, p_2), since (x, p_2) is an arc in D. The same holds for y^- and the new arc (p_{s-1}, y^-). Thus, $\mathcal{P}' = \{x^-, x^+, p_2, \ldots, p_{s-1}, y^-, y^+\}$ is a valid path in D', since also the arcs (x^-, x^+) and (y^-, y^+) are inserted. The existence of this path is exactly the definition of $(x^-, x^+) \prec_A (y^-, y^+)$.

For the opposite direction, we can analogousy take a path and reconstruct the original path from it. □

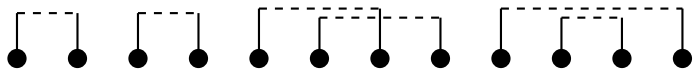

Figure 2.6: Two disjoint pairs, two halving pairs and two nested pairs. The lines shall denote membership to the same pair constraint.

The proposition including its proof can be easily altered to match the case of transforming arc pair constraints to vertex pair constraints. The similarity of these to propblem types will become important in Chapter 4.

2.2.3 Structural Conditions

This section introduces some conditions on vertex pair constraints, as they occur in the literature. The most comprehensive distinction of structural cases was contributed by Kováč in 2013 [20]. He defined three relations in which two forbidden pairs can be related to each other and results in a distinction of in total seven different structures, which will be shortly explained now. Afterwards, we will encounter other structural definitions and try to relate them between each other.

Kováč used a fixed topological sorting $<$ on the graph D, which allowed an explicit relation of any two pairs.

Definition 2.18 (Kováč [20])**.** Let $G = (V, A)$ be a directed, acyclic graph and $\mathcal{F} \subseteq V \times V$ a set of forbidden pairs. Let further $<$ be a fixed topological sorting on V. Two forbidden pairs $(f_1, f_2), (g_1, g_2) \in \mathcal{F}$ are called

- *disjoint* if $f_1 < f_2 < g_1 < g_2$ or $g_1 < g_2 < f_1 < f_2$ holds,
- *halving* if $f_1 < g_1 < f_2 < g_2$ or $g_1 < f_1 < g_2 < f_2$ holds or
- *nested* if $f_1 < g_1 < g_2 < f_2$ or $g_1 < f_1 < f_2 < g_2$ holds.

Figure 2.6 shows examples to the three definitions.

Observe that every two forbidden pairs have exactly one of these three structures. By allowing only a subset of them, we gain seven possibilities to restrict the set of forbidden pairs. The general problem allows all three relations. If only disjoint pairs are prohibited, Kováč calls the constraints *overlapping* if nested is avoided, they are *ordered* and *well-paranthesized* if there are no halving pairs. By these definitions, Kováč gave an overview with Table 2.1. The table also contains complexities for all these cases. Most of them are found and proven in the mentioned paper of Kováč.

In 1997, Yinnone [27] investigated the shortest path problem with forbidden pairs under a skew symmetry condition.

Definition 2.19 (Yinnone [27])**.** Let $D = (V, A)$ be a directed graph, $s, t \in V$ the unique source and sink and $\mathcal{F} \subseteq V \times V$ a set of forbidden pairs. The graph D is called *skew symmetric* if

$$(u, v) \in A \Rightarrow (v', u') \in A$$

Name	disjoint	halving	nested	Complexity
general	✓	✓	✓	NP-hard
overlapping	×	✓	✓	NP-hard
ordered	✓	✓	×	NP-hard
well-paranthesized	✓	×	✓	$O(n^\omega)$
disjoint	✓	×	×	$O(n+m)$
halving	×	✓	×	$O(n^{\omega+1})$
nested	×	×	✓	$O(n^\omega)$

Table 2.1: Overview over structures of forbidden pair sets. This table is taken from Kováč [20].

holds for all $u, v \in V \setminus \{s, t\}$.

Note that Yinnone allows circles in his graphs. Nevertheless, the problem of finding a feasible s-t-path in D (called SFP within the paper) remains polynomial. The given proof shows that the problem is polynomially equivalent to the augmenting path problem [5, 27].

If we restrict a skew symmetric graph to be acyclic, we are able to relate this definition to the already seen ones of Kováč.

Proposition 2.2. *Let D be a skew symmetric directed graph with the set $\mathcal{F}$ of forbidden pairs. Let D be acyclic. Then, there is a topological sorting concerning which $\mathcal{F}$ has nested structure. Furthermore, all vertices not contained in a forbidden pair can be sorted into the center of the nested pairs.*

Algorithm 1 in Section 2.1 provided a topological sorting of a directed acyclic graph. We are going to change this algorithm a little bit using the skew symmetry. The pseudo code can be found in Algorithm 4. Whereas the original procedure `topSort` works from a source vertex to a sink, the altered method `topSortSkewSymmetric` sorts the vertices simultaneously from the source and the sink side. This way we guarantee that the result sorting has a nested structure. List S denotes the beginning of the topological sorting, whereas T is concurrently filled from the sink side of D. Lines 2 to 5 do the first step manually: Source s and sink t are removed from the graph and stored in the appropriate lists. The while loop in line 6 then procedes with the removal of all forbidden pairs. To facilitate the proof of this proposition, let us proof the next lemma first.

Lemma 2.4. Let D be the graph obtained after an iteration of the while loop in Line 6 of Algorithm 4. If there are vertices left in the graph which belong to a forbidden pair, then there is always a vertex v belonging to a forbidden pair with $\delta^-(v) = \emptyset$ and its mate v' has $\delta^+(v') = \emptyset$.

Proof. For the first part, assume for contradiction, the claim does not hold. Arbitrarily choose any vertex $v \in V$ contained in a forbidden pair. Since it is no source, we now iteratively switch over to one of v's predecessors searching for a source. Since the graph is acyclic and finite, this clearly terminates after some steps. To contradict the claim, the only possibility is, we end up in a vertex not contained in a forbidden pair. If we now add some cleverness to the choice of the predecessor, the claim immediately follows. We always choose a vertex belonging to a forbidden pair and only take one without forbidden pair if there is no other possibility. Assume, we went from v to an vertex x (with $x' = x$), then the skew symmetry tells us that x has v' as a predecessor and we can proceed to v'. Thus, this search procedure cannot end up in a vertex with $x' = x$, which contradicts the assumption. The second part of the claim trivially follows by the definition of skew symmetry. □

Now, the proof of Proposition 2.2 follows.

Proof. of Proposition 2.2. By Lemma 2.4, we know that after the while loop in line 6, only the vertices are left, which aren't contained in any forbidden pair. They will now be sorted using the existing method `topSort` in line 12. Finally, the three lists are concatenated and returned. The topological sorting induced by R nests the forbidden pairs around the isolated vertices, which proves the existence of such a sorting. □

Algorithm 4 Procedure `topSortSkewSymmetric` sorts a skew symmetric graph topologically.

Input: Graph $D = (V, A)$
Output: List R with nested structure

procedure topSortSkewSymmetric(D)
  Vertex $s :=$ the unique source of D
  Vertex $t :=$ the unique sink of D
  List $S := \{s\}$, $T := \{t\}$
  $D := D - \{s, t\}$
  while $|\mathcal{F}| > 0$ **do**
    Choose $x \in V$ with $\delta^-(x) = \emptyset$ and $x' \neq x$
    Add x to S
    Add x' as first element to T
    Remove (x, x') from $\mathcal{F}$
    $D := D - \{x, x'\}$
  List $M :=$ `topsort`(D)
  List $R := S \cup M \cup T$
  return R

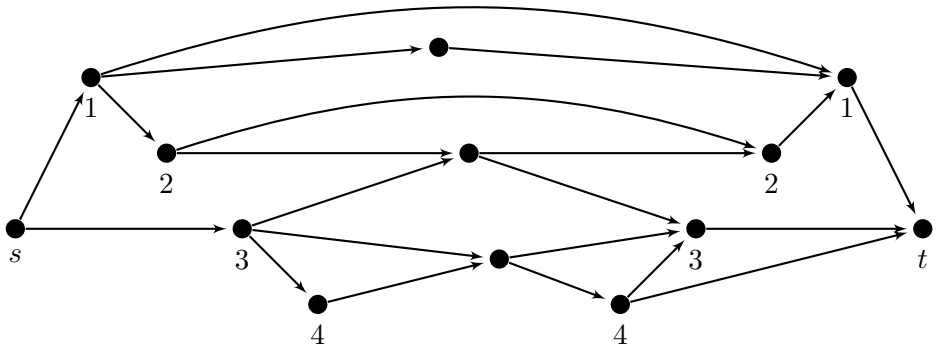

Figure 2.7: An example for a skew-symmetric graph. The topological sorting is given by the x coordinates of all vertices. The numbers denote the memberships to the four forbidden pairs.

The advanced structure of the isolated vertices inbetween all forbidden pairs emphasizes the symmetry of the graph. Figure 2.7 shows an example for a skew-symmetric graph. Two vertices labeled with the same number belong to the same forbidden pair. There is no condition to arcs starting in s or ending in t. Among the vertices not belonging to a forbidden pair, there is no arc allowed, because otherwise the backwards arc is also contained by the skew-symmetry. This would lead to a cycle. Hence, the skew-symmetric graphs are indeed a very restricted class of graphs.

In the same year Kováč first published his structural distinctions of multiple forbidden pairs, also Kolman and Pangrác worked on the problem. They went in a similar direction, but took an approach not based on a topological sorting. Instead, they based their definition on the reachability of vertices, which we mentioned in Definition 2.5.

Definition 2.20 (Kolman, Pangrác [18])**.** Let $G = (V, A)$ be a directed, acyclic graph and $\mathcal{F} \subseteq V \times V$ a set of forbidden pairs. If no two pairs $(f_1, f_2), (g_1, g_2) \in \mathcal{F}$ fulfill

- $f_1 \prec g_1 \prec f_2 \prec g_2$, the set $\mathcal{F}$ has *hierarchical structure* and
- $f_2 \prec g_1$ or $f_2 = g_1$, the set $\mathcal{F}$ has *halving structure*.

Let us compare these two definitions with the well-paranthesized and halving structure introduced by Kováč. Since halving is ambiguous now, we will always refer to the appropriate definition of halving, so there is no danger of confusion.

Lemma 2.5. If a set of vertex pairs is well-paranthesized, then it is also hierarchical. The converse is not true in general.

Proof. First, let D be a directed acyclic graph and $\mathcal{F}$ forbidden pairs on it with well-paranthesized structure. We show that it is hierarchical. In a topological

ordering, $x_k \not\prec x_l$ always implies $x_k \not\prec x_l$. Let now (u, v) and (x, y) be pairs in $\mathcal{F}$. Since they are not halving (in the Kováč-sense), one of the three relations of $u < x < v < y$ has to be violated and by the previous observation also the corresponding $\prec$-relation of those vertices. This implies that the two pairs cannot halve each other (in the Kolman and Pangrác sense). Thus, the instance is of hierarchical structure.

As a counterexample for the opposite direction, consider the graph in Figure 2.8. The forbidden pairs shall be discernible by the indices of the vertices. The next lines only refer to the halving definition for two forbidden pairs of Kováč, since we need to see that there is halving in any possible topological sorting. We need to find a topological sorting $<$, such that no two constraints halve each other. Clearly, the bottom path $\{s, a_1, a_2, a_3, b_2, t\}$ is fixed for any topological sorting and hence, only the two upper vertices b_3 and b_1 need to be sorted into this enumeration. Because of the two arcs (s, b_3) and (b_1, b_2), they both need to be located between s and b_2. Now, consider the few remaining possibilities in terms of halving. To avoid halving of the pairs 2 and 3, the vertex b_3 needs to be put after a_2. But the pairs 1 and 2 halve if b_1 is set after a_2. These both conditions contradict each other because of the arc (b_3, b_1). □

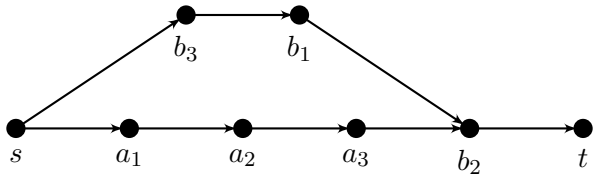

Figure 2.8: A hierarchical constraint set, which can not be well-paranthesized.

The first and according to our investigation only one considering structural constraints to an arc equivalent of the pair constraint problem is Xing Tan in his dissertation in 2012 [26].

Definition 2.21 (Tan [26])**.** Let D be a directed acyclic graph. Let $\mathcal{F}_A \subseteq A \times A$ be a set of forbidden arc pairs. $\mathcal{F}_A$ is called *arc hierarchical* if no two pairs $(a_1, a_2), (a_3, a_4) \in \mathcal{F}_A$ halve each other, i.e.,

$$a_1 \prec_A a_3 \prec_A a_2 \prec_A a_4.$$

Note that Tan called his arc set E instead of A and calls them edges instead of arcs. However, he is also concerned with directed acyclic graphs. We decided to change the notation here to be consistent with the rest of this thesis.

In Section 2.2.2 we found two ways passing over from vertex pair constraints to arc pair constraints and conversely. As seen, these two methods preserved reachability among the vertex reachability and arc reachability relations of the

members of the according pair constraint sets. Thus, the arc hierarchical case of Tan classifies as hierarchical in the sence of Kolman and Pangrác, which we observed already.

3 Recursive Algorithms for the PCP

Over the last years, there have been several approaches to solve especially the shortest path problem with forbidden vertex pairs. This chapter collects the combinatorial and recursive approaches to several variants or special cases of the shortest path problem with pair constraints.

This chapter is divided into two parts. The first part contains algorithms solving the PAFP problem. These algorithms are presented in the literature and collected here including their running times. The second part is Section 3.2, were we present a new algorithm. Our algorithm also contracts the given graph, but it is able to handle also binding pairs.

3.1 Existing Algorithms for PAFP

3.1.1 A Dynamic Programming Approach by Kováč

After the distinction into several structures, Kováč observed every case in more detail in his paper. For the well-parenthesized case, he gave a dynamic programming approach, which we will present here. The next paragraphs will shortly explain the basics of recursion and then extend to the idea of dynamic programming. Afterwards, Kováč's approach is explained.

Solving combinatorial problems often leads to sub problems, which have exactly the same structure as the given problem instance, but are of lower size. An easy example is the factorial $n!$ of an integer n, which can be defined explicitly by

$$n! := \prod_{i=1}^{n} i.$$

The same factorial can also be defined as $n! := n \cdot (n-1)!$. For this definition, we need to explicitly define $1! := 1$. Whereas the first definition allows you to explicitly compute $n!$ without knowing any lower factorials, the second approach forces you to *recursively* compute all factorials lower than n. Of course, in this easy example, the actual calculation to be done is the same in both cases, but this may change for larger or more complex problems.

Sometimes, a recursive definition needs to call itself more than once. Consider for example the Fibonacci numbers 1, 1, 2, 3, 5, 8, 13 and so on. The n-th Fibonacci number is defined as $f_n := f_{n-2} + f_{n-1}$ and the explicit beginning $f_0 = 1$ and $f_1 = 1$. To compute the Fibonacci number f_n for some $n \gg 1$, you need to compute f_{n-2}. Once this is done, your recursion will compute f_{n-1}, which

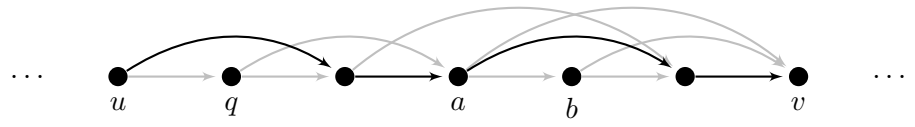

Figure 3.1: Two forbidden pairs (q, v) and (a, b). The black path is feasible and its first arc jumps over q. Thus, $J[u, v] = \texttt{true}$.

itself is defined as $f_{n-1} := f_{n-3} + f_{n-2}$. At this point, your method computes f_{n-2} the second time. Again, the Fibonacci numbers are an easy example, but for problems of huge size and complexity, this can become highly inefficient.

At this point, *dynamic programming* comes into play. The basic idea of dynamic programming is, to store the results of recursively computed solutions. If the same recursion needs to be calculated again, the program can look-up if the solution is already computed and then either return the computed one or compute it, save it and then return it. This guarantees that no sub results are computed twice. In the example of the Fibonacci numbers, this basically results in a calculation of all Fibonacci numbers from 0 to n. The only thing to be careful about is that there are no cyclic dependencies between the elements of the recursion and that the recursion is guaranteed to terminate.

Kováč presents an approach for the shortest path problem using dynamic programming. Therefore he defines the following two labels P and J on each pair of vertices. For all vertices $u, v \in V$, let

- $P[u, v]$ be `true` if a feasible u-v-path exists and
- $J[u, v]$ be `true` if v is part of a forbidden pair (q, v) with $u \prec q \prec v$ and there is a feasible u-v-path such that its first arc jumps over q concerning the topological sorting $<$. See Figure 3.1 for an example.

Note that Kováč allows the labels to have beside the values `true` and `false` also the value `undefined` and `null`. `Null` is the initial value of all entries of P and J and signals only that they have not been computed yet. The labels are defined as

$$J[u, v] = \begin{cases} \bigvee_{(u,w)\in A, q \prec w} P[w, v] & u \prec q, (q, v) \in \mathcal{F} \\ \texttt{undefined} & \text{otherwise} \end{cases}$$

and

$$P[u, v] = \begin{cases} \texttt{true} & u = v \\ \texttt{false} & (u, v) \in \mathcal{F} \\ \bigvee_{u \preceq w \prec v, (w,v)\in A} P[u, w] & v \preceq v' \vee v' \prec u \\ \bigvee_{u \preceq w \prec v} (P[u, w] \wedge J[w, v]) & \exists (q, v) \in \mathcal{F} : u \prec q \prec v. \end{cases}$$

To solve the problem, we just compute $P[s, t]$ now. Kováč defined the problem as finding a shortest feasible path, but this algorithm just finds out, whether

there is a path or not. We will now explain the given recursions and sketch their correctness.

Let us first check the label J. Label $J[u, v]$ shall only contain a value (`true` or `false`) if there is a forbidden pair (q, v) ending in v. Kováč assumes $x < y$ for a forbidden pair (x, y), so this includes already that q is before v. Thus, only in the correct case, the recursion starts computing anything. In this case, we consider all arcs (u, w) starting in u and jumping over q. If any of their endpoints w has a feasible path to v, the label $P[w, v]$ will be true and hence, this is a correct recursion for $J[u, v]$.

Consider now $P[u, v]$. The first case $u = v$ is trivially true, and if (u, v) is a forbidden pair, there cannot be any feasible path. The third case is, that either v has no forbidden pair or its mate v' is "outside of $[u, v]$". The recursion now considers all arcs (w, v) ending in v. If there is a feasible path to any of those w's, then the path can be extended by (w, v) to a path from u to v. The last case now considers a forbidden pair (q, v) at v with q "inside $[u, v]$". Here, the search for a feasible path is a bit more complicated. For all vertices w "in $[u, v]$", we check if there is a feasible path from u to w and a feasible w-v-path whose first arc jumps over q. In fact, this searches for the last vertex w of the u-v-path before q. This construction is a bit complicated. Its purpose is to guarantee, that there are no cyclic calculation dependencies between the entries of J or P.

Proposition 3.1. *There are no cyclic dependencies between the recursive definitions of J and P.*

This proposition is not contained in [20], but we state it here for clarification.

Proof. The entries of the tables J and P can be imagined as vertices of a directed graph, where an arc points from an entry $[u, v]$ to all other entries, which are needed for the computation of the entry $[u, v]$. Then, we would have to show, that this graph is acyclic. We achieve this by finding a general "direction", which all arcs follow.

All calls in a recursion at the entry $[u_1, v_1]$ (in any of the two labels) point either to another entry $[u_2, v_2]$, where $v_2 < v_1$ or where the distance concerning the topological sorting[1] is smaller than between u_1 and v_1. Calls to the same entry $[u_1, v_1]$ only point from P to J, namely in the fourth case of P, where w can be also u.

This sketches, why there cannot be any cycles in this graph and hence in the dependencies of the entries of P and J. □

The proposition ensures, that the algorithm always terminates and the considerations beforehand save the correctness of both labels. The running time of this recursion is obviously $\mathcal{O}(n^3)$, since there are $2n^2$ entries to be computed and the computation of every entry is at most $\mathcal{O}(n)$.

[1] This shall sloppily denote the number of vertices strictly between the two mentioned ones.

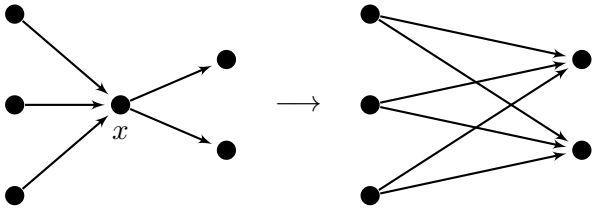

Figure 3.2: Contraction of the vertex x.

Kováč advanced his algorithm by a fast Boolean matrix multiplication technique. With some auxiliary properties, he gains a running time of less than $\mathcal{O}(n^{2.5})$. This points away from our graph theoretic problems, wherefore we leave this off here.

3.1.2 Graph Contraction by Kolman and Pangrac

Kolman and Pangrác were the first introducing a method contracting a graph to solve the shortest path problem with vertex pair constraints. They defined three rules, each contracting either vertices, arcs or forbidden pairs. Within this section, we assume that every vertex is contained in at most one forbidden pair.

- **Rule** `R1` (Contraction of a vertex). For every vertex $x \in V$ with $x' = x$ do the following: For every pair of arcs $(u, x), (x, v) \in A$, add a new arc (u, v) with weight $w(u, v) := w(u, x) + w(x, v)$ to A. In the case that (u, v) is already contained in the graph, only keep the shorter one according to w. Then, remove x from V and all arcs containing x from A. Figure 3.2 gives an example.
- **Rule** `R2` (Removal of an arc). Remove all arcs $a \in A \cap \mathcal{F}$ from A.
- **Rule** `R3` (Removal of a forbidden pair). For every pair $(x, y) \in \mathcal{F}$ with $x \not\prec y$, remove (x, y) from $\mathcal{F}$.

The power of these rules is revealed by the following lemma.

Lemma 3.1 (Kolman et al. [18])**.** Let D and $\mathcal{F}$ be an instance of the shortest path problem with pair constraints. Then, at least one of the rules `R1`, `R2` or `R3` is applicable to D and $\mathcal{F}$, unless $V \neq \{s, t\}$, where s and t are the desired source and destination.

Proof. We assume for contradiction, that none of the rules are applicable, but the graph has more than two vertices. Consider a vertex $x \in V \setminus \{s, t\}$. x must be contained in a forbidden pair, because otherwise rule `R1` is applicable. Since there is a forbidden pair left, consider a forbidden pair $(x, y) \in \mathcal{F}$ such that there is no other nested forbidden pair $(u, v) \in \mathcal{F}$ with $x \prec u \prec v \prec y$. Because of the

finiteness, this clearly exists. R3 is not applicable and hence there is a path from x to y. And due to the inapplicability of R2, this consists of at least two arcs combined by an internal vertex, say u. Again, since R1 is not possible, there has to be a forbidden pair $(u, v) \in \mathcal{F}$ or $(v, u) \in \mathcal{F}$. But by the hierarchical structure of the graph we know, that v also lies on the shortest path between x and y. This contradicts the choice of the pair (x, y). □

This lemma states, that the algorithm terminates only if the graph consists of only s and t. Since all rules remove anything from the graph, it gets smaller in every loop and hence, the algorithm cannot hang in endless loops. These two facts combined give the guarantee of a finite and correct termination. The final procedure contractGraphKP of Kolman and Pangrác is located in algorithm 5. In this state, the algorithm only determines the existence of a feasible s-t-path in D. Kolman and Pangrác proposed arc labels which hold the information during the contraction method. In the beginning, all labels are set to (u,v) for the arc $(u, v) \in A$. The rule R1 concatenates the labels of the deleted arcs and sets that as the label for the newly inserted arc. Finally, the arc (s, t) contains the whole shortest path as its label if it exists.

Lemma 3.2 (Kolman et. al. [18])**.** Simply implemented, this algorithm takes $\mathcal{O}(mn^2)$ time.

Proof. The graph shrinks in every application of one of the rules. Thus, the while loop in Line 2, has at most n iterations, since by Lemma 3.1 the application of R2 and R3 guarantee a removable vertex. Let us now bound the three rules. The contraction of vertices connects in the worst-case all predecessors with all successors. Thus, contraction of a vertex needs $\mathcal{O}(n^2)$ time. Removal of an arc is constant once it is found and hence $\mathcal{O}(1)$ for R2. The most expensive part is the detection of redundant forbidden pairs. Usual graph traversal algorithms such as depth-first-search or breadth-first-search are bound by $\mathcal{O}(m)$. The number of forbidden pairs is bounded by $\mathcal{O}(n)$, because we assumed every vertex to contain to at most one forbidden pair. Thus, we have $\mathcal{O}(mn)$ for R3. We can also demand D to be connected, since otherwise we simply remove components different from the one holding s and t. This means $n \in \mathcal{O}(m)$. Altogether, we have $\mathcal{O}(mn^2)$, because $\mathcal{O}(n^2) \subseteq \mathcal{O}(mn)$ by the last argument and the n iterations of the loop. □

3.1.3 A Contraction Variant for Arcs by Tan

Tan was concerned with an arc variant of Kolman and Pangrác's problem. He gave a polynomiality proof by altering the algorithm of Kolman and Pangrác to an arc equivalent. He defined his three rules, called steps, in the following way.

- **Step** S1. Search for an arc $a_1 = (x, y) \in A$ not contained in a forbidden pair. Add a new vertex u to D and change all arcs pointing to either x or

Algorithm 5 Procedure `contractGraphKP` contracts a graph until only s and t survive.

Input: Graph $D = (V, A)$
Output: `true` if a feasible path in D exists

```
procedure contractGraphKP(D, F)
    while |V| > 2 do
        apply R1
        apply R2
        apply R3
    if |A| = 0 then
        return false
    else
        return true
```

y, such that they now point to u. Analogously change all arcs starting in x or y. Finally, remove x and y from D.

- **Step** `S2` (Removal of an incident forbidden pair). Find a forbidden pair $((x, y), (y, z)) \in \mathcal{F}$ of two contiguous arcs. For all arcs $(y, u) \in A$ with $u \neq z$, add a new arc (x, u). Now, add a new forbidden pair for all forbidden pairs using (x, y) and all newly added arcs (x, u), whereby (x, y) is replaced by (x, u) in the new forbidden pair. Finally, (x, y) is removed from A and all forbidden pairs using (x, y) from $\mathcal{F}$. See Algorithm 6 on the facing page for a description in pseudo code.

- **Step** `S3`. For every pair $(x, y) \in \mathcal{F}$ with $x \not\prec_A y$, remove (x, y) from $\mathcal{F}$.

Note, that steps `S1` and `S2` only do *one* operation, whereas step `S3` (and the rules of Kolman and Pangrác as well) remove *all* matching forbidden pairs. The following observation of Tan directly resembles those of Kolman and Pangrác.

Lemma 3.3 (Tan [26])**.** Let D and $\mathcal{F} \subseteq A \times A$ be an instance of the shortest path problem with vertex pair constraints. Then, at least on of the steps `S1`, `S2` or `S3` is applicable to D and $\mathcal{F}$, unless $V \neq \{s, t\}$.

Proof. We assume, that the assumption does not hold. This can be simply contradicted following the proof of lemma 3.1. □

Lemma 3.4 (Tan [26])**.** This algorithm takes $\mathcal{O}(m^4)$ running time.

Proof. Clearly, the number of forbidden (arc) pairs is bounded by m^2. Let us now check the complexities of the 3 subroutines.

- Step `S1` requires $\mathcal{O}(m)$ time iterating through the arcs searching for a free one. Finding all arcs to the vertices x and y takes $\mathcal{O}(n)$ time and hence, step `S1` is bounded by $\mathcal{O}(m)$.

Algorithm 6 Step 2 of Tan's contraction method.

Input: Graph $D = (V, A)$, forbidden pairs $\mathcal{F}$
Output: nothing (the graph D is altered)

```
procedure TANSTEP2(D, F)
    find ((x, y), (y, z)) ∈ F
    if No such pair found then return
    for (y, u) ∈ δ⁺(y) \ {(y, z)} do
        Add (x, u) to A
        for (a1, a2) ∈ {(a1, a2) ∈ F | a1 = (x, y) ∨ a2 = (x, y)} do
            Forbidden pair f = (c, d)
            if a1 = (x, y) then
                c = (x, u)
                d = a2
            else
                c = a1
                d = (x, u)
            Add f to F
    Remove (x, y) from A
```

- Step `S2` searches for a connected forbidden pair. By brute force, this takes $\mathcal{O}(m^2)$ possibilities. The insertion of new forbidden pairs and arcs is bounded by $\mathcal{O}(n)$. Thus, step `S2` is bounded by $\mathcal{O}(m^2)$.
- Step `S3` Needs a reachability check for $\leq m^2$ forbidden pairs. Reachability can be checked by a depth-first-search or breadth-first-search in $\mathcal{O}(m)$ time, which leads to a bound for step `S3` of $\mathcal{O}(m^3)$.

Step `S1` reduces the number of vertices in D by one and the other two steps don't change it. If `S2` can be applied, it leaves the arc (y, z) without forbidden pair. If `S3` can be applied, there are two more arcs free. Thus, if any of the two is applicable, there is work to do for `S1` in the next iteration. And if both are not applicable, `S1` must be applicable because of Lemma 3.3. Thus, the number of iterations is bounded by n and hence the running time is bounded by $\mathcal{O}(m^4)$. □

3.2 The Pair Constraint Reduction Algorithm

The previous algorithms deal with forbidden pairs. In this section, we introduce a new algorithm similar to the contraction method of Kolman and Pangrác. Our method `contractGraph` solves instances $D = (V, A)$ with a set $\mathcal{F}$ of forbidden pairs and $\mathcal{B}$ of binding pairs if their union $\mathcal{C} := \mathcal{F} \cup \mathcal{B}$ has well-parenthesized structure according to the definition of Kováč given in section 2.2.3 on page 23. This requires a topological sorting $<$ given on the vertices V.

For our method, we need only two contraction rules, which are then iteratively applied.

- **Rule** `B1` (Contraction of a vertex). This is exactly the same as in section 3.1.2 on page 32. For every vertex $x \in V$ with $x' = x$ do the following: For every pair of arcs $(u, x), (x, v) \in A$, add a new arc (u, v) with weight $w(u, v) := w(u, x) + w(x, v)$ to A. In the case that (u, v) is already contained in the graph, only keep the shorter one according to w. Then, remove x from V and all arcs containing x from A.
- **Rule** `B2` (Removal of a pair constraint). For every vertex pair constraint $(u, v) \in \mathcal{C}$, for which u and v are neighbors concerning $<$:
  1. If $(u, v) \in \mathcal{F}$, remove (u, v) from A (if contained) and then remove the forbidden pair (u, v) from $\mathcal{F}$.
  2. If $(u, v) \in \mathcal{B}$, there are two cases. If $u < v$, remove all incoming arcs $(x, v) \in A$ except for (u, v). Otherwise (if $v < u$), remove all outgoing arcs $(u, x) \in A$ except for (v, u). In both cases, remove the binding pair (u, v) from $\mathcal{B}$.

This leads to an altered version of `contractGraphKP`. The only difference are the names of the applied rules. We call this algorithm `contractGraph` and for completeness, it is stated here again.

Algorithm 7 Procedure `contractGraph` reduces a graph respecting forbidden and binding pairs.

Input: Graph $D = (V, A)$, vertex pair constraints $\mathcal{C}$
Output: `true` if a feasible path exists in D

```
procedure contractGraph(D, C)
    while |V| > 2 do
        apply B1
        apply B2
    if |A| = 0 then
        return false
    else
        return true
```

We need to verify, that this method only terminates if $V = \{s, t\}$.

Lemma 3.5. Let $D = (V, A)$ be a directed acyclic graph and $\mathcal{F}$ and $\mathcal{B}$ the forbidden and binding pairs, which have well-parenthesized structure. Let further D be the graph obtained after application of rule R1. Then, there is at least one pair $(x, y) \in \mathcal{B} \cup \mathcal{F}$ of vertices, which are neighbors in the topological sorting.

Proof. Assume for contradiction, the lemma does not hold. Take any vertex pair constraint $(u, v) \in \mathcal{B} \cup \mathcal{F}$. Since the lemma does not hold, there is at least one

vertex x between the vertices u and v. Rule R1 was applied and hence, x belongs to a constraint (x, y) (or (y, x)). The two pairs (u, v) and (x, y) (resp. (y, x)) do not halve, wherefore y also has to lie between u and v. Now, x and y have to be neighbours and the same reflection hols for x and y again. This contradicts the finiteness of the vertex set V and the constraint sets $\mathcal{F}$ and $\mathcal{B}$. □

The lemma says, that after each application of `B1`, there is work to do for `B2`. And it can easily be seen, that in the opposite direction, `B2` leaves work for `B1`. Thus, the algorithm only stops if the condition of the while loop is not satisfied.

Our method seems to be the first algorithm finding a shortest path in a directed graph, which respects forbidden pairs *and* binding pairs. Unfortunately, we are loosing a bit generality compared to Kolman and Pangrác, since we are using well-parenthesized constraints instead of hierarchical ones. We now investigate the running time complexity of `contractGraph`. Section 5.2 explains, how this method can be extended to obligatory pairs and Chapter 5 presents our implementation of the method including running times on various instances.

Theorem 3.1. *Let $D = (V, A)$ be a directed, acyclic graph and $\mathcal{B}$ and $\mathcal{F}$ pair constraints with well-parenthesized structure. Let $n = |V|$, $m = |A|$ and $k = |\mathcal{B} \cup \mathcal{F}|$. Then, the algorithm `contractGraph` solves this instance in $\mathcal{O}(n^4)$ time.*

Proof. As usual for running time proofs of these contraction algorithms, we first check the complexity of the individual contraction rules.

- Rule `B1` iterates over all vertices. This takes $\mathcal{O}(n)$ time and then, all incoming and outgoing arcs are combined to a new arc. Since the number of incoming arcs as well as the number of outgoing arcs is bounded by n, we have at most n^2 new arcs here. This leads to the total running time of $\mathcal{O}(n^3)$.
- Rule `B2` visits all forbidden pairs and checks, whether the two vertices are neighbors. This is $\mathcal{O}(n)$, since every vertex is contained in at most one forbidden pair. Once a forbidden pair is found, the possibly existing arc between the vertices is removed. This is $\mathcal{O}(n)$, since the arc has to be found in the unordered list of outgoing arcs. For a binding pair, all other arcs have to be deleted, which is $\mathcal{O}(n)$ as well. Hence, we have $\mathcal{O}(n^2)$ here.

We already saw, that the two rules guarantee the applicability of each other. Thus, in each iteration, at least one vertex is removed and hence, we have at most n iterations. This leads to the overall running time of $\mathcal{O}(n^4)$. □

4 A Polyhedral Study of the PCP

In the previous chapter, we concentrated on combinatorial and recursive algorithms for solving the PCP. In this chapter, we are interested in deriving a good IP formulation for the problem. We already saw that the constraint matrix of the integer program formulation of the pair constraint problem given in Section 2.2 is not totally unimodular. Geometrically, this means that the given inequalities describe a polytope, which is not necessarily integral. Thus, the solution of the LP-relaxation is in general not a point corresponding to the shortest feasible path.

This problem can be solved by a obtaining complete linear description of the polytope $P(\mathcal{C})$. A complete linear description is precisely the convex hull of the integer points in $P(\mathcal{C})$ given as a system of linear inequalities. For the majority of well-known optimization problems that can be formulated as IPs, a complete linear description is unknown or contains an exponential number of inequalities.

In this chapter, we derive a complete linear description for the polytope $P(\mathcal{C})$ for some special cases of the shortest path problem with pair constraints. For ease of notation, in this chapter we consider arc pair constraints instead of vertex pair constraints. As we saw in Section 2.2.2, the problems are equivalent. Some parts of this chapter will be published in the proceedings of the Latin-American Algorithms, Graphs and Optimization Symposium (LAGOS) 2015.

To the best of our knowledge, we are the first to carry out polyhedral analysis of the PCP. In Section 4.1, we examine $P(\mathcal{C})$ for the case of exactly one forbidden or binding pair. Already in this simple case, the number of additional inequalities can be exponential in $|V|$. In Section 4.2, we extend these results to a contiguously disjoint structure of pair constraints.

4.1 A Complete Description for one Pair

For first little insights, we start by considering instances of the pair constraint problem with only one forbidden pair and one binding pair, respectively. For both, we give a complete linear description of the associated polytope. Afterwards, we examine the the number of inequalities needed for the linear description in the case of a forbidden pair. Although the number can be exponentially large in $|V|$, they can be separated in polynomial time.

4.1.1 One Forbidden Pair

During this section, we work on a given instance of the pair constraint problem. Let therefore the network $D = (V, A)$ and $\mathcal{F} \subseteq A \times A$ be fixed. As the title

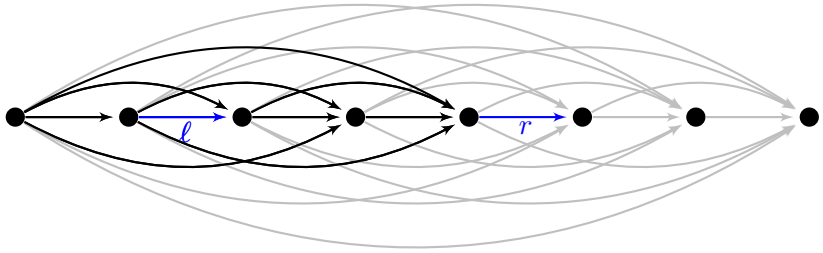

Figure 4.1: The example graph K_8 with gray arcs. The forbidden pair (ℓ, r) is highlighted in blue and the subgraph $D_{r\setminus\ell}$ is black.

suggests, we assume $\mathcal{F} = \{(\ell, r)\}$, that is, we have exactly one forbidden pair. Let also the source s and the sink t be fixed. For the following results, we need to define some terms on the graph D.

Definition 4.1. For D and the forbidden pair $\mathcal{F} = \{(\ell, r)\}$, define the following:

- Let $V_r := \{v \in V \mid v \preceq r^-\} \subseteq V$ be the set of all vertices, from which r^- can be reached. This includes also r^- itself.
- Denote by $A_r := (V_r \times V_r) \cap A$ the set of all arcs with both endpoints in V_r.
- $A_{r\setminus\ell} := A_r \setminus \{\ell\}$ is the arc set on V_r excluding ℓ.
- Let now $D_{r\setminus\ell} := (V_r, A_{r\setminus\ell})$ be the subgraph of D defined by the previous sets.
- $\Gamma(D_{r\setminus\ell}) \subseteq 2^{A_{r\setminus\ell}}$ shall denote the set of all inclusion-minimal (s, r^-)-arc-cuts. Call every cut $C \in \Gamma_{r\setminus\ell}$ an $r\setminus\ell$-cut.

As an example, let us consider $D = K_8$ the complete graph with the highlighted forbidden pair (ℓ, r). This graph will follow us through the next explanations. Figure 4.1 shows the graph including the forbidden pair and $D_{r\setminus\ell}$ highlighted.

Definition 4.2. For each $r\setminus\ell$-cut $C \in \Gamma(D_{r\setminus\ell})$, we define now the corresponding cut inequality

$$x(C) - x_r \geq 0. \tag{4.1}$$

We call the family of these inequalities the family of $r\setminus\ell$-cut inequalities. They are the basis for the following statements.

Lemma 4.1. The $r\setminus\ell$-cuts inequalities are valid for $P(\mathcal{F})$, i.e., all points $x \in P(\mathcal{F})$ fulfill the $r\setminus\ell$-cut inequalities.

Proof. Let $\mathcal{P}$ be a feasible s-t-path in D and $x^{\mathcal{P}}$ its characteristic vector. The inequalities are trivially satisfied if $\mathcal{P}$ avoids r. Hence, let us assume $r \in \mathcal{P}$. In this case $\mathcal{P}$ avoids ℓ, since it is feasible. Thus, its s-r^--subpath lies completely in $D_{r\backslash\ell}$ and hence intersects all $r\backslash\ell$-cuts $C \in \Gamma(D_{r\backslash\ell})$. For this reason, $x(C) \geq 1$ for all $C \in \Gamma(D_{r\backslash\ell})$. Thus, all $r\backslash\ell$-cut inequalities are satisfied with equality. □

The next lemma is a bit more tricky. These two lemmata form the basis for the following main theorem of this section.

Lemma 4.2. The forbidden pair inequality $x_l + x_r \leq 1$ is implied by an $r\backslash\ell$-cut inequality.

Proof. Recall the flow conservation Equalities (2.1) of the shortest path problem formulation for all vertices $v \in V$:

$$\sum_{a \in \delta^+(v)} x_a - \sum_{b \in \delta^-(v)} x_b = \begin{cases} 1 & v = s \\ -1 & v = t \\ 0 & \text{otherwise} \end{cases}$$

Let us now sum up the equalities for all the vertices $S := \{s, \ldots, \ell^-\}$[1]. Since $t \notin S$, the right side is clearly 1. All arcs appearing in the second sum are also starting in the set S as well and hence vanish because of the first sum. Thus, the remaining arcs are those jumping over or starting in ℓ^-. Let us give a name to this set: $T := \{(a, b) \mid a \preceq \ell^- \prec b\}$. Note that $\ell \in T$. Of course, T is an arc cut of D and hence $T' := T \cap A_{r\backslash\ell}$ is an arc cut in $\Gamma(D_{r\backslash\ell})$. This results in the redundant equality

$$x(T) = 1. \tag{4.2}$$

Consider the cut inequality of T', $x(T') - x_r \geq 0$. If we subtract (4.2) from it, we get

$$x(T') - x(T) - x_r \geq -1.$$

We know, that $\ell \in T$, but $\ell \notin T'$. Thus, we can summarize this to

$$-x_\ell - x_r - x(T \setminus (T' \cup \{\ell\})) \geq -1.$$

We see already the forbidden pair inequality appearing. Consider the negative of this equation

$$x_\ell + x_r + x(T \setminus (T' \cup \{\ell\})) \leq 1$$

and by $x(T \setminus (T' \cup \{\ell\}))$, this implies the forbidden pair inequality $x_\ell + x_r \leq 1$. □

The following theorem states the main result of this section.

[1] The enumeration shall be according to the topological sorting $<$ on the graph D.

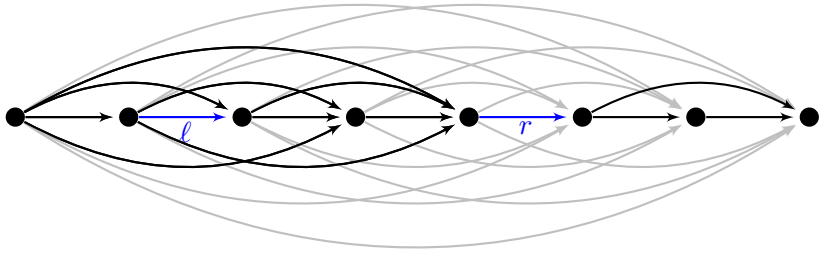

Figure 4.2: K_8 in gray, the subgraph D_r in black.

Theorem 4.1. *For one forbidden pair,*

$$P(\mathcal{F}) = \{x \in P \mid x(C) - x_r \geq 0 \; \forall C \in \Gamma_{r\setminus\ell}\} =: Q$$

holds, where P denoted the solution set of the unconstrained shortest path problem.

Proof. By Lemma 4.1, we have $P(\mathcal{F}) \subseteq Q$. Lemma 4.2 states, that all infeasible paths are excluded. Together, we have $\text{conv}(Q \cap \mathbb{Z}^{|A|}) = P(\mathcal{F})$. It only remains to prove the integrality of Q.

Let therefore x^* be a fractional vertex of Q, whose existence we have to contradict. Note that x^* corresponds to a flow in D. Let us now create a new graph D_r by removing ℓ from D as well as all arcs not lying on any s-t-path through r. In other words, no arcs on a r^+-t-path are removed and on s-r^--paths, ℓ is the only arc removed. Figure 4.2 shows the graph D_r for the example K_8. The graph D_r can be seen as an extension of the graph $D_{r\setminus\ell}$ to the complete vertex set V. Conversely, if we restrict D_r to the vertex set V_r, we get $D_{r\setminus\ell}$ again.

We now define a maximum flow problem on D_r by defining arc capacities $c_a := x^*_a$ for all arcs in $A(D_r)$. Let x^r be a maximum flow in D_r. Clearly, x^r avoids ℓ completely and hence can be decomposed into valid paths. The classical max-flow-min-cut theorem states that the value $v(x^r)$ is upper bounded by x^*_r, since $\{r\}$ is an inclusion-minimal arc cut of D_r. If we can now show that $v(x^r)$ actually is x^*_r, we know that the remaining flow $x^* - x^r$ avoids r. Thus, let us find lower bounds for $v(x^r)$.

We use the max-flow-min-cut theorem again and choose an arbitrary inclusion-minimal s-t-cut $C \in \Gamma(D_r)$. We need to show that this cut has at least capacity x^*_r. By the construction of D_r, we can distinguish three cases concerning the location of C.

Case 1 C disconnects s and r^-. Since C is inclusion-minimal, we have $C \in \Gamma(D_{r\setminus\ell})$. By the cut inequality of C, we get

$$x^*_r \leq x^*(C) = c(C).$$

Case 2 C disconnects r^+ and t. x^* is itself a feasible flow sending a value of at least x^*_r through r^+. Since this subflow also reaches t, we get

$$x^*_r \leq x^*(C) = c(C),$$

because it needs to intersect C.

Case 3 C disconnects r^- and r^+. C is inclusion-minimal and hence $C = \{r\}$. Of course, $x^*_r \leq c(C)$.

This shows, that $v(x^r)$ is indeed x^*_r. Thus, $x^\ell := x^* - x^r$ avoids r completely and with x^ℓ and x^r we found a decomposition into two feasible subflows, $x^\ell, x^r \in P(\mathcal{F})$. Since $x^* = (1 - x^*_r)x^\ell + x^*_r x^r$ is a convex combination, x^* is also contained in $P(\mathcal{F})$, which is the desired contradiction. □

By this theorem we get that the forbidden pair inequalities (4.1) form a complete linear description of the forbidden pair polytope with a single pair, together with the flow conservation constraints (2.1) and $x \geq 0$.

4.1.2 One Binding Pair

The same argument flow can be translated to the problem of one binding pair $\mathcal{B} = \{(\ell, r)\} \subseteq A \times A$ on the graph $D = (V, A)$. Let us fix this problem instance again. The additional inequalities for the binding pair problem arise from arc cuts as well, but we need to define a different graph before.

Definition 4.3. For D and the binding pair $\mathcal{B} = \{(\ell, r)\}$, define:

1. Let $V^b_{\ell,r} := \{v \in V \mid \ell^+ \preceq v \preceq r^-\}$ be the set of all vertices between the two binding pair arcs.
2. $D^b_{\ell,r} := (V^b_{\ell,r}, (V^b_{\ell,r} \times V^b_{\ell,r}) \cap A)$ The connection graph of ℓ and r.
3. $\Gamma(D^b_{\ell,r})$ shall denote the set of all inclusion-minimal (ℓ^+, r^-)-arc-cuts.

Figure 4.3 shows the graph $D^b_{\ell,r}$ for the example K_8 again. In this example, there are two possible minimal cuts in $\Gamma(D^b_{\ell,r})$, namely the two arcs starting in ℓ^+ or the two ending in r^-. Introduce the following inequalities for all cuts C in $\Gamma(D^b)$:

$$x(C) - x_\ell \geq 0 \tag{4.3}$$

These inequalities look quite similar to the ones used for one forbidden pair, but due to the different graph construction, they fulfill our needs, as we will see shortly. In addition to these cut inequalities, we need the binding pair inequality

$$x_r - x_\ell \geq 0 \tag{4.4}$$

to describe the polytope $P(\mathcal{B})$. This section goes alike the last one and the proofs differ only slightly.

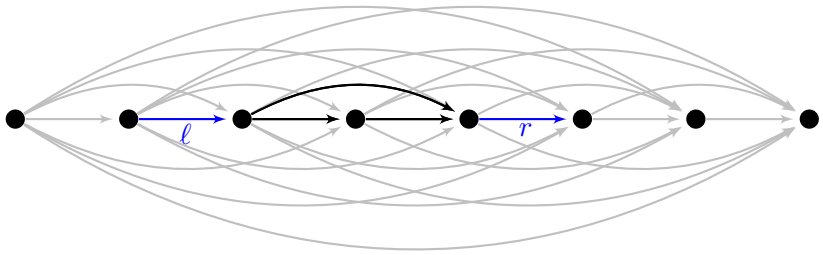

Figure 4.3: K_8 in gray, the subgraph $D^b_{\ell,r}$ in black.

Proposition 4.1. *The inequalities* (4.3) *and* (4.4) *are valid for* $P(\mathcal{B})$ *and all infeasible paths violate the binding pair inequality* (4.4)*.*

Proof. Take any feasible path $\mathcal{P}$ and its characteristic vector $x^{\mathcal{P}}$. If it avoids ℓ then the inequalities are trivially satisfied. So let us assume $x^{\mathcal{P}}_\ell = 1$. Since $\mathcal{P}$ is feasible, it also uses r and the binding pair inequality is valid again. And since both ℓ and r are used, the path needs to go from ℓ^+ to r^-, which is clearly in $D^b_{\ell,r}$. Here, it intersects all cuts and thus the cut inequalities are also valid.

Let $\mathcal{P}$ be infeasible now. The binding pair (ℓ, r) can only be violated if $\mathcal{P}$ uses ℓ, but not r. In this case, $x^{\mathcal{P}}_r = 0$ and $x^{\mathcal{P}}_\ell = 1$, which obviously violates the binding pair inequality. □

With these two results we are in the same situation as before and now able to state the theorem.

Theorem 4.2. *For* $D = (V, A)$ *and* $\mathcal{B} = \{(\ell, r)\}$*,*

$$P(\mathcal{B}) = \{x \in P \mid (4.3), (4.4)\} =: Q$$

holds.

Proof. The proposition tells us $P(\mathcal{B}) \subseteq Q$ and no additional integral points belong to Q. Thus, the only possibility is an additional fractional vertex x^* of Q whose existence we have to contradict: Let D^b be a new subgraph of D on the same vertex set. $A(D^b)$ shall be the union of all s-t-paths using both ℓ and r. D^b restricted to $[\ell^+, r^-]$ is precisely the graph $D^b_{\ell,r}$. Note, that both $\{\ell\}$ and $\{r\}$ are arc cuts in D^b. Figure 4.4 illustrates D^b.

On this graph, we are now interested in maximum flows. The capacities on the arcs shall be $c_a := x^*_a$ and let x be a maximum flow in D^b. Of course, x uses both ℓ and r, since they are cuts. x^* was a point in Q and hence it satisfies $x^r \geq x^\ell$. x^*_ℓ is again an upper bound for $v(x)$. We will now show, that $x^*_\ell = v(x)$ using the max-flow-min-cut theorem. Let C be any inclusion-minimal cut in $\Gamma(D^b)$. We can distinguish several cases regarding the allocation of C on D^b.

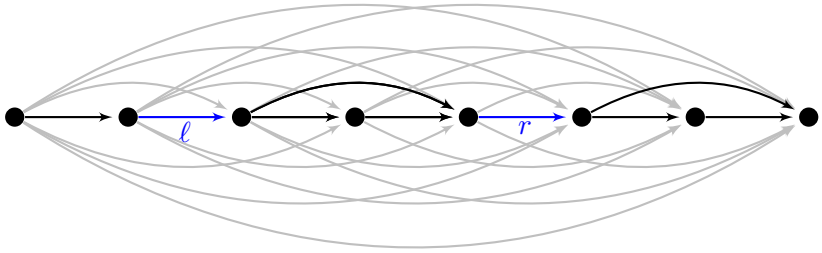

Figure 4.4: K_8 in gray, the subgraph D^b in black.

Case 1 C disconnects s and ℓ^- or r^+ and t. Since x^* is a feasible flow and no arcs between s and ℓ^- (respectively r^+ and t) are removed, it must be possible to send an amount of x^*_ℓ through C (respectively, the same holds due to the binding pair inequality (4.4) for the case C disconnects r^+ and t). Thus, $x^*_\ell \leq x^*_r \leq c(C)$.

Case 2 C disconnects $\ell+$ and r^-. Then, $C \in \Gamma(D^b_{\ell,r})$ and the corresponding cut inequality holds for x^*. Thus,

$$x^*_\ell \leq x^*(C) = c(C)$$

Case 3 $C \in \{\{\ell\}, \{r\}\}$. In this case, $c(C) = x^*_\ell$ or $c(C) = x^*_r$. By the binding pair inequality we get again $x^*_\ell \leq c(C)$.

This shows that the subflow x of x^* uses the whole of x^* on ℓ and the remaining flow $x' := x^* - x$ avoids ℓ completely. In addition, the whole flow x uses r and hence, the both subflows x and x' are feasible concerning $\mathcal{B}$ and since $x^* = (1 - x^*_\ell)x' + x^*_\ell x$ is a convex combination, all the three points x, x' and x^* are in $P(\mathcal{B})$. This contradicts the assumption, that x^* is a fractional vertex of $P(\mathcal{B})$. □

This is the same result as for a forbidden pair: The inequalities (4.3) and (4.4) form a complete linear description of the polytope associated with the shortest path problem with pair constraints for the case of one binding pair (ℓ, r).

4.1.3 Separation and the Number of Facets

As the complete linear descriptions derived in the previous sections suggest, the number of inequalities needed to describe the polytope can be very large. We are now going to show, that it can actually grow exponentially large. We show this for the special case of a complete graph $D = (V, A)$ and one forbidden pair (ℓ, r).

Definition 4.4. Define the following:

- Let Π be the set of infeasible paths in D using both ℓ and r.
- For $\mathcal{I} \in \Pi$, let

$$C_{\mathcal{I},\ell,r} := \{(u,v) \in A_{r\setminus\ell} \mid \ell^+ \preceq v \preceq r^-, u \notin \mathcal{I}^v, v \in \mathcal{I}^v\}$$

  be the cut associated to this infeasible path. Note that $C_{\mathcal{I},\ell,r} \in \Gamma(D_{r\setminus\ell})$ for all $\mathcal{I} \in \Pi$. This cut can be seen as all the arcs "entering" the path $\mathcal{I}$ between ℓ^+ and r^-.
- Let $[u,v] := \{x \in V \mid u \preceq x \preceq v\}$ be the set of all vertices between u and v and $\mathcal{P}[u,v]$ the subpath of a path $\mathcal{P}$ between u and v.
- Let $\mathcal{P}$ and $\mathcal{Q}$ be two inner vertex-disjoint u-w-paths. We call the union $\mathcal{P} \cup \mathcal{Q}$ a (u,w)-split. For two s-t-paths, let $s(\mathcal{P},\mathcal{Q})$ be the number of splits in $\mathcal{P} \cup \mathcal{Q}$ for which $\mathcal{Q}$ has at least one inner vertex.

With these definitions, we can now formulate the following theorem.

Theorem 4.3. *Let $\mathcal{I} \in \Pi$. Then, the $r\setminus\ell$-inequality associated to $C_{\mathcal{I},\ell,r}$, i.e.,*

$$x_r \leq x(C_{\mathcal{I},\ell,r}) \tag{4.5}$$

defines a facet $f(\mathcal{P}) \subseteq P(\mathcal{F})$. Furthermore, two different paths $\mathcal{I}_1, \mathcal{I}_2 \in \Pi$ define different facets of $P(\mathcal{F})$, $f(\mathcal{I}_1) \neq f(\mathcal{I}_2)$.

For an infeasible path $\mathcal{I} \in \Pi$, let $\Pi_{\mathcal{I}}$ be the set of *feasible* s-t-paths in D which are tight for $x_r \leq x(C_{\mathcal{I},\ell,r})$. The proof of this theorem gets quite technical. For this reason, let us outsource a useful claim. Recall, that $\mathcal{P}^v$ denotes the set of vertices of a path $\mathcal{P}$.

Lemma 4.3. Let $\mathcal{I} \in \Pi$. A feasible path $\mathcal{Q}$ satisfies the inequality $x_r \leq x(C_{\mathcal{I},\ell,r})$ with an equality if and only if one of the following holds:
(I) $\mathcal{Q}$ avoids both ℓ and r and $\mathcal{I}^v \cap \mathcal{Q}^v \cap [\ell^+, r^-] = \emptyset$.
(II) $\mathcal{Q}$ contains either ℓ or r and $s(\mathcal{I},\mathcal{Q}) = 0$.

Proof. Let us prove the lemma separately for the cases of whether arcs ℓ and r are used by $\mathcal{Q}$. Let therefore $\mathcal{I} \in \Pi$ be fixed and take $\mathcal{Q}$ a feasible path which avoids ℓ and r. Assume, $\mathcal{Q}$ satisfies the inequality with equality. Since $x_r^{\mathcal{Q}} = 0$, we need to contradict the existence of a vertex in $\mathcal{I}^v \cap \mathcal{Q}^v \cap [\ell^+, r^-]$. Assume this vertex w exists. Since $\mathcal{Q}$ avoids ℓ, but meets $\mathcal{I}$ in w, there needs to be an entering arc, $\mathcal{Q}$ uses to meet $\mathcal{I}$. Let this arc be (u,v). Possibly, but not for sure, we have $v = w$ here. Since $\mathcal{Q}$ avoids r, there cannot be an equation in the cut equality, which is the desired contradiction. The opposite direction works similar: If there is no vertex $v \in \mathcal{I}^v \cap \mathcal{Q}^v \cap [\ell^+, r^-]$, $\mathcal{Q}$ cannot visit any arc in $C_{\mathcal{I},\ell,r}$ and hence $x(C_{\mathcal{I},\ell,r}) = 0$.

Let $\mathcal{Q}$ now visit ℓ and fulfill the $r\setminus\ell$-cut inequality with equality. $\mathcal{Q}$ is feasible and hence avoids r. Thus, we need to show, that there is no (u,w)-split (with

an inner vertex on $\mathcal{Q} \setminus \mathcal{I}$) in $\mathcal{I} \cup \mathcal{Q}$ with $u, w \in [\ell^+, r^-]$. Assume, there is such a (u, w)-split. Let v be the largest vertex on $\mathcal{Q}$ before w. As assumed, $u \prec v \prec w$ and hence $v \in \mathcal{Q} \setminus \mathcal{I}$. Thus, $(v, w) \in C_{\mathcal{I},\ell,r}$ and we are ready. For the backwards direction, let $\mathcal{Q}$ have $s(\mathcal{I}, \mathcal{Q}) = 0$. Assume there is an arc $(v, w) \in C_{\mathcal{I},\ell,r}$ visited by $\mathcal{Q}$. Let u be the largest vertex before v that $\mathcal{I}$ and $\mathcal{Q}$ share. Since $\mathcal{Q}$ visits ℓ, $\ell^+ \leq u$ and hence $\mathcal{I}[u, w] \cup \mathcal{Q}[u, w]$ is a (u, w)-split in $[\ell^+, r^-]$, which has to be counted in $s(\mathcal{I}, \mathcal{Q})$, a contradiction.

As the last case, let $\mathcal{Q}$ visit r with an equality in the cut inequality. Assume again the existence of an (u, w)-split in $\mathcal{I} \cup \mathcal{Q}$ with $u, w \in [\ell^+, r^-]$. Let (v, w) be the arc of $\mathcal{Q}$ ending in w. This arc is of course contained in $C_{\mathcal{I},\ell,r}$. But since $\mathcal{Q}$ avoids ℓ, there have to be at least two arcs in $C_{\mathcal{I},\ell,r}$, which uses $\mathcal{Q}$. This is the desired contradiction. Let now g be the smallest vertex $\mathcal{I}$ and $\mathcal{Q}$ share with $\ell^+ \leq g$. Since $\mathcal{Q}$ avoids ℓ, the arc (f, g) (with the proper vertex f) of $\mathcal{Q}$ is also contained in $C_{\mathcal{I},\ell,r}$. Thus, the sum of $\mathcal{Q}$ over this cut is at least two, which is a contradiction. Finally, let there be no split between $\mathcal{I}$ and $\mathcal{Q}$ in $[\ell^+, r^-]$. In this case, the both paths meet only once between ℓ^+ and r^- and hence there is only one entering arc $(u, w) \in C_{\mathcal{I}_\ell,r}$ used by $\mathcal{Q}$, which leads us to the desired equality. □

Since the dimension of the polytope $P(\mathcal{F})$ is completely unknown to us, we are not able to say anything about the co dimension of a face of it. This is why we apply a non-standard proof technique here. We show that

$$P(\mathcal{F}) \subseteq \operatorname{aff}(\{x^{\mathcal{Q}}\} \cup f(\mathcal{I})) \tag{4.6}$$

holds for every infeasible path $\mathcal{I} \in \Pi$ and every feasible path $\mathcal{Q}$, which is not tight for the inequality associated to $\mathcal{I}$. This is possible by the following lemma.

Lemma 4.4. Let $\mathcal{I} \in \Pi$ be an infeasible path and $\mathcal{Q} \in P(\mathcal{F}) \setminus f(\mathcal{I})$ a feasible path not tight for the inequality of $\mathcal{I}$. Then, we can find the following affine combination

$$x^{\mathcal{I}} = \lambda_{\mathcal{Q}} x^{\mathcal{Q}} + \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda_{\mathcal{P}} x^{\mathcal{P}}$$

of $x^{\mathcal{I}}$ with $\lambda_{\mathcal{Q}} + \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda_{\mathcal{P}} = 1$. In particular, if $\mathcal{Q}$ belongs to case I of Lemma 4.3 then $\lambda_{\mathcal{Q}} = -\frac{1}{s(\mathcal{I},\mathcal{Q})}$, while for case II $\lambda_{\mathcal{Q}} = -\frac{1}{s(\mathcal{I},\mathcal{Q})+1}$ holds.

This lemma is quite complicated. After its proof, the proof of Theorem 4.3 finally follows. There is explained, how we can use this unexpected affine combination for proving (4.6).

Proof. For this proof, fix the infeasible path $\mathcal{I} \in \Pi$. Since for every path $\mathcal{Q}$, the number $s(\mathcal{I}, \mathcal{Q})$ is constant, we do an induction over this number. In other words, we find the desired affine combination with $\mathcal{Q}$ using other affine combinations of paths with a lower split number.

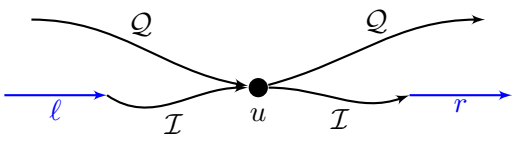

Figure 4.5: The infeasible path $\mathcal{I}$ uses both ℓ and r, whereas $\mathcal{Q}$ avoids both. The two paths meet in their only common point u.

Let us begin the induction with the cases $s(\mathcal{I}, \mathcal{Q}) = 0$. Take a path $\mathcal{Q}$, which is not tight for the cut inequality associated to $\mathcal{I}$, see Figure 4.5. By Lemma 4.3, $\mathcal{Q}$ avoids both ℓ and r. The lemma further tells us, that there is a common vertex u of the paths $\mathcal{I}$ and $\mathcal{Q}$. The affine combination is

$$\begin{aligned} x^{\mathcal{I}} = &(x^{\mathcal{I}[s,u]} + x^{\mathcal{Q}[u,t]}) \\ &+ (x^{\mathcal{Q}[s,u]} + x^{\mathcal{I}[u,t]}) \\ &- x(\mathcal{Q}). \end{aligned}$$

The two new paths are rearrangements of $\mathcal{I}$ and $\mathcal{Q}$ at u. Obviously, both are feasible. It remains to check, that they are tight for the considered inequality. This can – in this case and in the plenty following ones – be easily checked by lemma 4.3.

To facilitate the induction step, we also give the induction start for the case $s(\mathcal{I}, \mathcal{Q}) = 1$ and start the induction with at least two splits. We need to distinguish three cases for $\mathcal{Q}$. Let S be the given (u, w)-split and let v be a node in $(\mathcal{Q} \setminus \mathcal{I}) \cap S$.

1. $\mathcal{Q}$ contains neither ℓ nor r.

$$\begin{aligned} x^{\mathcal{I}} = &\frac{1}{2}(x^{\mathcal{I}[s,u]} + x^{\mathcal{Q}[u,v]} + x_{(v,t)}) \\ &+ \frac{1}{2}(x_{(s,v)} + x^{\mathcal{Q}[v,w]} + x^{\mathcal{I}[w,t]}) \\ &- \frac{1}{2}(x_{(s,v)} + x_{(v,t)}) \\ &+ \frac{1}{2}(x^{\mathcal{Q}[s,u]} + x^{\mathcal{I}[u,t]}) \\ &+ \frac{1}{2}(x^{\mathcal{I}[s,w]} + x^{\mathcal{Q}[w,t]}) \\ &- \frac{1}{2}x^{\mathcal{Q}} \end{aligned}$$

2. $\mathcal{Q}$ contains ℓ but not r.

$$\begin{aligned} x^{\mathcal{I}} = &(x^{\mathcal{Q}[s,w]} + x_{(v,t)}) \\ &+ (x_{(s,v)} + x^{\mathcal{Q}[v,w]} + x^{\mathcal{I}[w,t]}) \\ &+ x^{\mathcal{I}[s,w]} + x^{\mathcal{Q}[w,t]}) \\ &- (x_{(s,v)} + x_{(v,t)}) \\ &- x^{\mathcal{Q}} \end{aligned}$$

3. $\mathcal{Q}$ contains r but not ℓ.

$$\begin{aligned} x^{\mathcal{I}} = &(x^{\mathcal{Q}[s,u]} + x^{\mathcal{Q}[u,t]}) \\ &+ (x^{\mathcal{I}[s,w]} + x^{\mathcal{Q}[u,v]} + x_{(v,t)}) \\ &+ (x_{(s,v)} + x^{\mathcal{Q}[v,t]}) \\ &- (x_{(s,v)} + x_{(v,t)}) \\ &- x^{\mathcal{Q}} \end{aligned}$$

Here, the new paths are similarly to the ones of $s(\mathcal{P}, \mathcal{Q}) = 0$ generated. Figure 4.6 illustrates the three situations. At this point we need that D is the complete graph, which guarantees us the existence of the arcs $(0, v)$ and (v, n).

The induction step uses the same argument of rearranging the paths. We again need to distinguish the same cases as above, which will be as similar as the above ones. For this reason, we will completely prove the first one. For the other two, we only show the differences in the construction of the new paths.

As induction hypothesis, assume that the claim holds for all paths $\mathcal{Q}'$ with $s(\mathcal{I}, \mathcal{Q}') \leq k - 1$. Let therefore $\mathcal{Q}$ be a feasible path, which is not tight for the cut inequality of $C_{\mathcal{I},\ell,r}$ and with $s(\mathcal{I}, \mathcal{Q}) = k$. Let for all cases the vertices u and w define the smallest (u, w)-split of $\mathcal{I} \cup \mathcal{Q}$ and let v be an inner vertex on $\mathcal{Q} \setminus \mathcal{I}$.

Case 1: $\mathcal{Q}$ avoids both ℓ and r

We can reformulate $x^{\mathcal{Q}}$ as

$$\begin{aligned} x^{\mathcal{Q}} = &(x_{(s,v)} + x^{\mathcal{Q}[u,t]}) && =: \mathcal{Q}_1 \\ &+ (x^{\mathcal{Q}[s,u]} + x_{(v,t)}) && =: \mathcal{Q}_2 \\ &- (x_{(s,v)} + x_{(v,t)}). && =: \mathcal{Q}_3 \end{aligned}$$

Table 4.1 on page 51 lists some properties of the paths $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_3$, which can easily be verified in Figure 4.7. The table shows that for $\mathcal{Q}_1$ and $\mathcal{Q}_2$ the induction hypothesis holds, which leads us to the following two affine combinations for $x^{\mathcal{I}}$.

$$x^{\mathcal{I}} = -\frac{1}{k} x^{\mathcal{Q}_1} + \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^1_{\mathcal{P}} x^{\mathcal{P}} \tag{4.7}$$

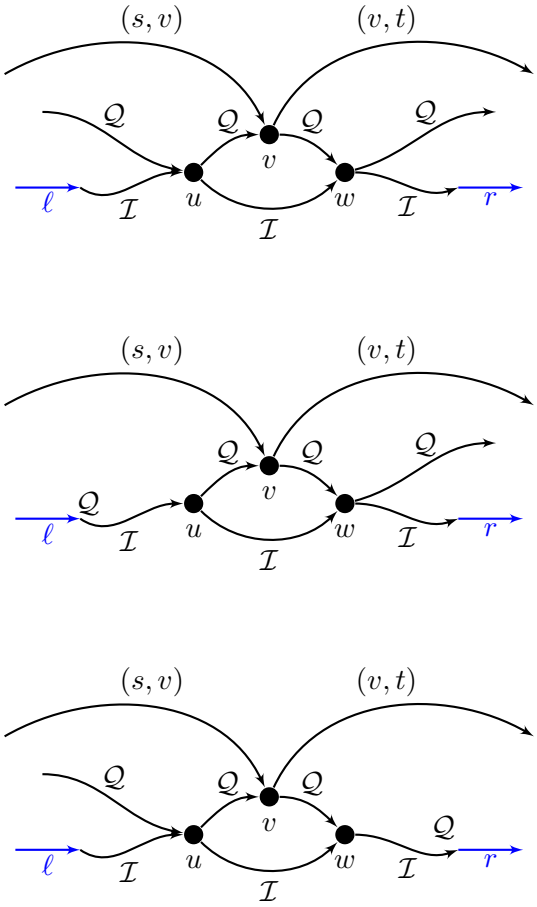

Figure 4.6: The three cases for $s(\mathcal{I}, \mathcal{Q}) = 1$.

$$x^{\mathcal{I}} = -x^{\mathcal{Q}_2} + \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^2_{\mathcal{P}} x^{\mathcal{P}} \tag{4.8}$$

With

$$\sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^1_{\mathcal{P}} = 1 + \frac{1}{k} \qquad \text{and} \qquad \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^2_{\mathcal{P}} = 2. \tag{4.9}$$

Let us now compute the equation $[k \cdot (4.7) + (4.8)]$.

$$(k+1)x^{\mathcal{I}} = -x^{\mathcal{Q}_1} + k \cdot \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^1_{\mathcal{P}} x^{\mathcal{P}} - x^{\mathcal{Q}_2} + \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^2_{\mathcal{P}} x^{\mathcal{P}}$$

From the construction of the paths $\mathcal{Q}_i$ we know $x^{\mathcal{Q}} = x^{\mathcal{Q}_1} + x^{\mathcal{Q}_2} - x^{\mathcal{Q}_3}$. Let us replace $-x^{\mathcal{Q}_1} - x^{\mathcal{Q}_2}$ in the last equation by $-x^{\mathcal{Q}} - x^{\mathcal{Q}_3}$ and divide it by $(k+1)$

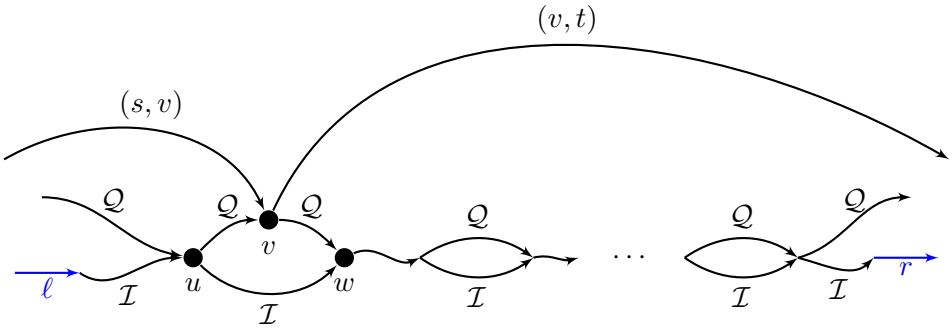

Figure 4.7: $\mathcal{Q}$ avoids both ℓ and r and $s(\mathcal{I},\mathcal{Q}) = k$.

Path	**Tight for $\mathcal{I}$**	**Number of Splits**	**Case (Lemma 4.3)**
$\mathcal{Q}_1$	not tight	$s(\mathcal{I},\mathcal{Q}_1) = k-1$	Case I
$\mathcal{Q}_2$	not tight	$s(\mathcal{I},\mathcal{Q}_2) = 0$	Case I
$\mathcal{Q}_3$	tight	$s(\mathcal{I},\mathcal{Q}_3) = 0$	Case I

Table 4.1: Properties of the three new paths $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_3$ for $\mathcal{Q}$ avoiding both ℓ and r.

to get

$$\begin{aligned} x^{\mathcal{I}} &= -\frac{1}{k+1}\cdot x^{\mathcal{Q}} - \frac{1}{k+1}\cdot x^{\mathcal{Q}_3} + \frac{k}{k+1}\cdot\sum_{\mathcal{P}\in\Pi_{\mathcal{I}}}\lambda^1_{\mathcal{P}}x^{\mathcal{P}} + \frac{1}{k+1}\cdot\sum_{\mathcal{P}\in\Pi_{\mathcal{I}}}\lambda^2_{\mathcal{P}}x^{\mathcal{P}} \\ &= -\frac{1}{k+1}\cdot x^{\mathcal{Q}} - \frac{1}{k+1}\cdot x^{\mathcal{Q}_3} + \sum_{\mathcal{P}\in\Pi_{\mathcal{I}}}\left(\frac{k\cdot\lambda^1_{\mathcal{P}}+\lambda^2_{\mathcal{P}}}{k+1}\cdot x^{\mathcal{P}}\right). \end{aligned} \tag{4.10}$$

Obviously we approach an affine combination of $x^{\mathcal{I}}$ with $x^{\mathcal{Q}}$ and the vertices in $f(\mathcal{I})$. By Table 4.1 we know that $\mathcal{Q}_3$ is tight for the inequality to $C_{\mathcal{I},\ell,r}$ and hence appears as a path $\mathcal{P}\in\Pi_{\mathcal{I}}$ in the sum. This gives rise to a definition of the new affine combination coefficients $\lambda_{\mathcal{P}}$. Define for $\mathcal{P}\in\Pi_{\mathcal{I}}$ the coefficient

$$\lambda_{\mathcal{P}} := \begin{cases} \frac{k\cdot\lambda^1_{\mathcal{P}}+\lambda^2_{\mathcal{P}}}{k+1} & \mathcal{P}\neq\mathcal{Q}_3 \\ \frac{k\cdot\lambda^1_{\mathcal{P}}+\lambda^2_{\mathcal{P}}-1}{k+1} & \mathcal{P}=\mathcal{Q}_3. \end{cases}$$

It is easy to see, that the affine combination (4.10) transforms into

$$x^{\mathcal{I}} = -\frac{1}{k+1}\cdot x^{\mathcal{Q}} + \sum_{\mathcal{P}\in\Pi_{\mathcal{I}}}\lambda_{\mathcal{P}}x^{\mathcal{P}}$$

Path	Tight for $\mathcal{I}$	Number of Splits	Case (Lemma 4.3)
$\mathcal{Q}_1$	not tight	$s(\mathcal{I}, \mathcal{Q}_1) = k - 1$	Case II
$\mathcal{Q}_2$	not tight	$s(\mathcal{I}, \mathcal{Q}_2) = 0$	Case I
$\mathcal{Q}_3$	tight	$s(\mathcal{I}, \mathcal{Q}_3) = 0$	Case I

Table 4.2: Properties of the three new paths $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_3$ for $\mathcal{Q}$ using ℓ but not r.

using the new coefficients $\lambda_{\mathcal{P}}$. This is precisely the combination we wanted to achieve. The last point to verify is now, that this is actually an *affine* combination, i.e.,

$$\sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda_{\mathcal{P}} = \frac{k+2}{k+1}.$$

This is achieved in the following calculation. The equality $(*)$ uses, that the two formulations using $\mathcal{Q}_1$ and $\mathcal{Q}_2$ were affine combinations as well. Formally, this was stated in (4.9).

$$\begin{aligned}\sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda_{\mathcal{P}} &= \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \frac{k \cdot \lambda^1_{\mathcal{P}} + \lambda^2_{\mathcal{P}}}{k+1} + \frac{1}{k+1} = \frac{1}{k+1} \cdot \left[k \cdot \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^1_{\mathcal{P}} + \sum_{\mathcal{P} \in \Pi_{\mathcal{I}}} \lambda^1_{\mathcal{P}} - 1 \right] \\ &\overset{*}{=} \frac{1}{k+1} \cdot \left[k(1 + \frac{1}{k}) + 2 - 1 \right] = \frac{k+2}{k+1}\end{aligned}$$

Case 2: $\mathcal{Q}$ uses either ℓ or r

The computations for this case go quite similar to the last one. Mainly the exact formulation of the coefficients $\lambda_{\mathcal{P}}$ differs. This is why we omit the repeated calculation. Instead we just give an explanation, how we rearrange the paths and which properties they have.

At first, let $\mathcal{Q}$ visit ℓ and not r. We do the same construction as before:

$$\begin{aligned} x^{\mathcal{Q}} &= (x_{(s,v)} + x^{\mathcal{Q}[u,t]}) && =: \mathcal{Q}_1 \\ &\quad + (x^{\mathcal{Q}[s,u]} + x_{(v,t)}) && =: \mathcal{Q}_2 \\ &\quad - (x_{(s,v)} + x_{(v,t)}). && =: \mathcal{Q}_3 \end{aligned}$$

Table 4.2 lists the properties of the three paths $\mathcal{Q}_1$, $\mathcal{Q}_2$ and $\mathcal{Q}_3$. Note, that $\mathcal{Q}_1$ is in this case of type II concerning Lemma 4.3. For type II, the coefficient of $\mathcal{Q}$ has to be $-\frac{1}{s(\mathcal{I},\mathcal{Q})}$ instead of $-\frac{1}{s(\mathcal{I},\mathcal{Q})+1}$ for type I.

The last case, that $\mathcal{Q}$ uses r looks similar, but here $\mathcal{Q}_2$ is of type II. The latter two are visualized in Figure 4.8. All three cases together prove the induction and hence the lemma. □

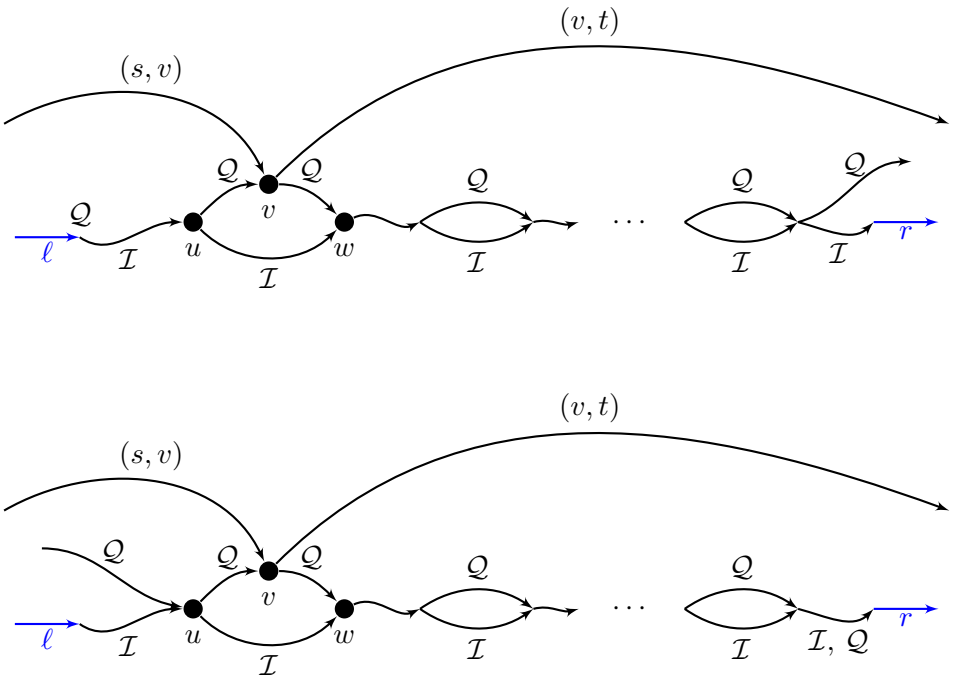

Figure 4.8: $\mathcal{Q}$ meets ℓ above and r below. In both cases, we have $s(\mathcal{I}, \mathcal{Q}) = k$.

With this lemma, we can now prove the theorem.

Proof of theorem 4.3. Actually, Lemma 4.4 tells us $x^{\mathcal{I}} \in \text{aff}(\{x^{\mathcal{Q}}\} \cup f(\mathcal{I}))$, which is the same as

$$x^{\mathcal{Q}} \in \text{aff}(\{x^{\mathcal{I}}\} \cup f(\mathcal{I})).$$

This can be achieved by a taking an affine combination of $x^{\mathcal{I}}$ and solving the equation for $x^{\mathcal{Q}}$. It can be easily seen that the result is again an affine combination. Since this holds for every feasible path $\mathcal{Q}$, we can use two of these expressions

$$x^{\mathcal{I}} \in \text{aff}(\{x_1^{\mathcal{Q}}\} \cup f(\mathcal{I})) \tag{4.11}$$

$$x^{\mathcal{Q}_2} \in \text{aff}(\{x^{\mathcal{I}}\} \cup f(\mathcal{I})) \tag{4.12}$$

for different paths $\mathcal{Q}_1$ and $\mathcal{Q}_2$ to get the final expression

$$x^{\mathcal{Q}_2} \in \text{aff}(\{x^{\mathcal{Q}_1}\} \cup f(\mathcal{I}))$$

we originally searched for. This is easy to verify by taking an affine combination of $x^{\mathcal{I}}$ from (4.11) and plugging it into one of $x^{\mathcal{Q}_2}$ of (4.12). The result is clearly an affine combination of $x^{\mathcal{Q}_2}$ in $\text{aff}(\{x^{\mathcal{Q}_1}\} \cup f(\mathcal{I}))$. Since the latter holds for all $\mathcal{Q}_2$ with a fixed path $\mathcal{Q}_1$, we can conclude

$$P(\mathcal{F}) \subseteq \text{aff}(\{x^{\mathcal{Q}_1}\} \cup f(\mathcal{I})).$$

□

The theorem allows a direct corollary, since there are exponentially many subpaths between ℓ^+ and r^-.

Corollary 4.1. The number of facets of $P(\mathcal{F})$ is exponential in the number of vertices $|V|$.

We have now seen, that the description of the convex hull of $P(\mathcal{C})$ can grow exponentially large. But this is not a hard problem because of the following result.

Proposition 4.2. *The system of inequalities*

$$\{x \in \mathbb{R}^{|A|} \mid (4.1),\ (2.1), x \geq 0\}$$

in the case of one forbidden pair or

$$\{x \in \mathbb{R}^{|A|} \mid (4.3), (4.4), (2.1), x \geq 0\}$$

for a binding pair can be separated in polynomial time.

Proof. Let $x^* \in \mathbb{R}^{|A|}$ be the point to be separated. Since (2.1) together with $x \geq 0$ describe a flow problem and the number of these constraints is polynomial, we can assume that $x^* \in P$, that is, x^* defines an (s,t)-flow of value 1. Otherwise, we can easily find the violated constraint.

Consider the case of one forbidden pair. Create the graph D_r according to the proof of Theorem 4.1 and use x^* as capacities for the arcs. Let x' be a maximum (s,t)-flow over D_r satisfying these capacities. Assume the value $v(x')$ is equal to x^*_r. That means by the max-flow-min-cut theorem that the capacity of every (s,t)-cut in D_r is at least x^*_r. As all cuts in $\mathcal{C}_{r\setminus\ell}$ are fully contained in D_r, all $r\setminus\ell$-cut-constraints (4.1) are satisfied. Contrarily, if the value of x' is strictly smaller than x^*_r, then there must exist a bottleneck corresponding to a cut with capacity smaller than x^*_r. Clearly, this cut belongs to $\mathcal{C}_{r\setminus\ell}$. And the corresponding inequality (4.1) is violated in this case.

The case of a binding pair can be proven by the same idea. In this case, the graph construction of the binding pair section is needed here. By this method we can separate the given inequality system in polynomial time. □

Obviously, the same result holds also for problem instances with exactly one binding pair.

4.2 Contiguously Disjoint Pair Constraints

This section extends our results to a first simple case of multiple constraints.

Definition 4.5. The set $\mathcal{C}$ is called *contiguously disjoint* if the following condition is satisfied:

$$s \preceq \ell_1^- \prec \ell_1^+ \preceq r_1^- \prec r_1^+ \preceq \ldots \preceq \ell_k^- \prec \ell_k^+ \preceq r_k^- \prec r_k^+ \preceq t$$

In this case, for the complete linear description, there are no more inequalities needed than the ones we got by each pair constraint for itself presented in Section 4.1.

Theorem 4.4. *Let $\mathcal{C}$ be a set of contiguously disjoint pair constraints. Then the following holds:*

$$P(\mathcal{C}) = \bigcap_{i=1}^{k} P(\mathcal{C}_i).$$

In this proof, we need to be careful, whether the considered constraint is a forbidden pair or a binding pair.

Proof. A great advantage of the proof in Section 4.1.1 for a forbidden pair is that Lemma 4.1 and Lemma 4.2 can be easily extended to the more general case of multiple pair constraints: They guarantee that all path vertices respecting the appropriate forbidden pair are kept in the new polytope $P(\mathcal{C})$ and that all invalid ones are eliminated by the $r\backslash\ell$-inequalities. An analogous argument holds for a binding pair with Proposition 4.1. This is independent of other constraints in $\mathcal{C}$. So again, the only point left to prove is the following lemma. □

Lemma 4.5. The polytope $\bigcap_{i=1}^{k} P(\mathcal{C}_i)$ is integral.

Proof. Our proof takes an inductive approach over the number of considered pair constraints. Let $\mathcal{C}_{\leq i} := \bigcup_{j=1}^{i} \mathcal{C}_i$. The induction hypothesis is: $P(\mathcal{C}_{\leq i-1}) \cap P(\mathcal{C}_i) = P(\mathcal{C}_{\leq i})$ and $P(\mathcal{C}_{\leq i})$ is integral. The induction start is already given in Section 4.1.1 for a forbidden pair and in Section 4.1.2 for a binding pair. Let us now solve the induction step.

The direction $P(\mathcal{C}_{\leq i-1}) \cap P(\mathcal{C}_i) \supseteq P(\mathcal{C}_{\leq i})$ directly follows by the definition. For the converse, let us take a fractional vertex x^* in $P(\mathcal{C}_{\leq i-1}) \cap P(\mathcal{C}_i)$ and contradict its existence. We are going to see that x^* is contained in $P(\mathcal{C}_{\leq i})$ by expressing it as a convex combination of points in $P(\mathcal{C}_{\leq i})$.

By using the split method from Sections 4.1.1 or 4.1.2, we can split the flow x^* into two flows $x^* = x^1 + x^2$. For a forbidden pair, let $x^1 = x^\ell$ and $x^2 = x^r$. For a binding pair, let $x^1 = x$ and $x^2 = x'$. In both cases, x^1 and x^2 are flows respecting $\mathcal{C}_i$.

Consider now the set $C := \{(u, v) \in A \mid u \preceq \ell_i^- \prec v\}$. Clearly, this set is an arc cut of D and since every arc in C is either jumping over ℓ_i^- or starting in ℓ_i^-, no path can intersect C twice. Let us consider now path decompositions of the three points x, x^1, x^2 and call them M, M^1 and M^2, respectively. Clearly, $M^1 \cup M^2$ is a path decomposition of x as well, but in general a different one than M.

Let us now take a closer look at C. This cut separates the first $i-1$ forbidden pairs from the i-th one, because $\mathcal{C}$ was contiguously disjoint. M is a path decomposition, where all paths respect the first $i-1$ forbidden pairs, and $M^1 \cup M^2$ a path decomposition respecting the i-th one. And clearly, both decompositions use C in the same amounts per arc. Since all the decompositions are clearly finite, we can compute a new decomposition of x by *mixing* the *left* part of M with the *right* part of $M^1 \cup M^2$: For every arc $a \in C$, consider all paths in M (and $M^1 \cup M^2$) using a. By arranging new paths out of the 0-a^--subpaths of the paths in M, the arc a itself and the a^+-n-subpaths contained in $M^1 \cup M^2$, we get our new decomposition, called M^*. The paths in M^* respect all pair constraints in $\mathcal{C}^{\leq i}$ and hence, x^* is a convex combination of these valid paths. That is, x^* is contained in $P(\mathcal{C}_{\leq i})$. Since x^* was fractional, M contained at least 2 paths and hence, x^* is no vertex, which is our desired contradiction. □

5 Computational Results

This chapter presents our algorithm introduced in chapter 3 on page 29 in a practical way. We implemented the method `contractGraph` and tested it on several instances. The following section gives a short introduction to the code of our software and the workaround that comes with it. Then, we explain, why we decided not to consider obligatory pairs. Sections 5.3 and 5.4 explain the real-world data, how we transformed them into instances of the shortest path problem with pair constraints and which test instances we chose. Finally, Section 5.5 harvests some pictures of flight trajectories and evaluates the calculation times and some more details to the calculation in a practical view.

5.1 Implementation

We decided to implement an environment for the pair constraint contraction algorithm in the programming language `C++`. In 9 files with a total of 3222 lines, we implemented 15 classes. The environment consists of several components, which we will present in the following paragraphs.

The first isolated part is a priority queue supporting a *decreaseKey* operation. The built-in `priority_queue` of `C++` does not provide a decreaseKey operation. A priority queue is a data structure supporting easy access to the first – or *smallest* – element. Each element has an integer key, which is used to compare them to determine the smallest element. The method `decreaseKey` resorts the queue, after the key of one element decreased. This is required for Dijkstra's Algorithm. Our `PriorityQueue<Item>` class is implemented as a binary heap, which is itself stored in a vector object. A good introduction to the priority queue and its implementation based on an array can be found in [9, 17].

We implemented an environment handling directed graphs. This consists of classes for vertices, arcs and directed graphs as well as an own class representing a distance label for Dijkstra's Algorithm. Of course, this algorithm is implemented in this environment as well as Algorithm 1 for a topological sorting. Our graph is stored in a doubly connected edge list (DCEL). This means, that every vertex stores one incoming and one outgoing arc. Every arc then stores a link to the next outgoing arc or to the next incoming arc. This structure allows easy navigation in the graph, especially for rule `R2'`.

To handle the vertex pair constraints, we built a second couple of classes extending the previously mentioned ones. These classes now support forbidden and binding pairs and the `contractGraph` algorithm presented in Chapter 3.

This setting is surrounded by a management software, which allows the easy creation, preparation, and deletion of problem instances. This part also runs the `contractGraph` method and measures its result as well as some parameters describing the performance. This will be explained further in section 5.5.

Every vertex gets a unique id of type unsigned short integer needing only two byte rather than four compared to normal integers. Unsigned short integers have a range of $\{0, \dots, 2^{16} = 65536\}$. As we will see, we need to treat around 48.000 vertices and hence this range is perfect for vertex ids. The basic `contractGraph` algorithm only determines whether there is a feasible path respecting all pair constraints or not. To actually *find* such a path, we need a slight modification of the graph. Every arc gets a label consisting of a list of vertex ids (unsigned shorts). By default, every label of arc (x, y) begins with the id of x followed by the id of y. The new arc (x, z), which arises from the contraction of y by the arcs (x, y) and (y, z) gets the concatenation of their two labels as new label[1]. By this way, we keep the information, which arcs and vertices this new arc represents. By iterating this, we always keep track of the shortest path in the original graph D connecting the two vertices x and z. Thus, the final label of the potential arc (s, t) explains the shortest trajectory respecting $\mathcal{C}$.

Summarized, our data structure for an arc (x, y) contains the following information:

- Pointers to the two vertices.
- A pointer to the next outgoing arc of x.
- A pointer to the next incoming arc of y.
- The arc weight (its length).
- The arc label.

The four pointers use 32 bit each as well as the weight, which is stored as an integer. The label contains all the vertices this arc shall connect stored as a list of ids. This leads to a bare minimum of 56 byte per arc, which is found by the `sizeof` command of C++. This is aggravated by the ids contained in the label of (x, y). The label sizes of course increase during the process of the algorithm. Typical arc label sizes are around 200 near the end of a run of `contractGraph`.

5.2 Extension to Obligatory Pairs

To get the method `contractGraph` to handle also obligatory pairs, the rule `B2` has to be altered to handle the new case of an obligatory pair. To force a path through at least one of two contiguous vertices in a topological sorted graph, we need to remove all arcs jumping over both a and b, see Figure 5.1. To achieve that in a DCEL, we needed to modify our connection structure in a manner, that not only incoming and outgoing arcs are saved at each vertex, but also *jumping*

[1] Acually, in this case, every vertex id is stored twice in the list, because the first arc label ends with the id of y and the second label begins with it. But this can clearly be avoided by simply removing this repeated id

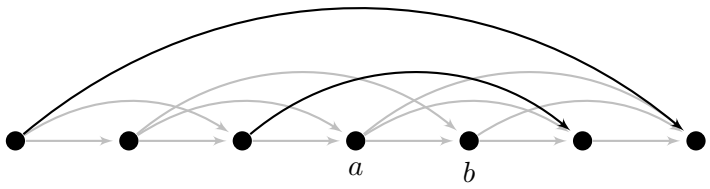

Figure 5.1: The two black arcs need to be deleted to enforce a feasible path to use at least one of the two vertices of the obligatory pair (a, b).

arcs. Since this jumping is only defined using a topological sorting in contrast to outgoing and incoming arcs, this caused problems to us. This is why we omitted obligatory pairs in this thesis.

5.3 A Translation Heuristic for Traffic Flow Restrictions

As already mentioned in the beginning, the RAD document contains tons of rules to be respected by passenger airplanes flying through the airway network graph. Table 5.1 on the next page shows a little excerpt of the RAD document. There are some columns dropped for the example and some of the arount 10.000 rows chosen to give an example. Within this context, vertices are called *waypoints* and arcs are called *segments*.

Waypoints in the airway network graph always have a name consisting of five uppercase letters. Airports have possibly shorter names. Every line in the RAD document table stands for one specific rule and every rule always has to be applied for every single flight itself. Every rule consists of three parts, namely the *scope*, the *condition* and an *order*. The first column "from – to" contains the scope, in this case a segment. There are also other scopes. The rule dictates something concerning this scope. The second column contains the condition and the order. The condition makes clear, whether this rule is active for the current flight or not. Usually, it is beginning in the second line. The first line contains the order. It tells us, what actually holds for the scope if the condition is satisfied. For the order, there are three types used in the document: *Compulsory*, *not available* and *only available*. Compulsory rules force us to use the scope if the condition is satisfied. Contrarily, not available rules forbid the scope in this case and in only available rules, we are not allowed to use the scope if the condition is not fulfilled. The scope mostly contains a single waypoint or a single segment. Sometimes, there is a height condition given to the scope as well or a complete airway consisting of multiple contiguous segmentss.

The difficult part is the condition. As seen in the examples, a condition is a boolean expression containing various types of properties a trajectory can have. Let us call these types *literals* for now. The literals are in general:

From – to	Restriction	Id No.
RANUX – VALEK	Not available for traffic ARR EDDF/FE Via BETEX Above FL245	LF2935
EPOLO – DIVKO	Only available and compulsory for traffic DEP LFMN Via MAMES/NILDU/VATIR With RFL above FL195	LF2756
NINTU – MEBAK	Only available for traffic 1. Via OLRAK 2. Via TIS a) DEP Geneva Area b) DEP EDDM, LFSB With ARR Bordeaux Group c) DEP ED**, LS**, LO** With ARR NAT 3. ARR Clermont Ferrand Group, Limoges Group, Poitiers Group	LF2313

Table 5.1: Some example rules taken from [11].

- Arrival or departure (DEP, ARR)
- Using a waypoint (VIA)
- Using an airspace (VIA)
- Using a waypoint starting with a certain prefix
- Using an airway containing given contiguous segments
- Using an airplane of a certain type
- Equipment properties
- Flight in a certain range of altitudes
- Flying in a certain time interval of the day

Of course, some of these types are constant once we chose an instance to compute. This holds e.g. for arrival, departure, equipment or the airplane type. The airplane type distincts for example between propellor maschines or jet planes. Other types are dynamicly depending on the chosen trajectory, such as the VIA conditions for example. The last two, namely the height or time intervals, always point only to a certain part of the trajectory such as a waypoint or an airspace.

The literals are combined to a Boolean expression by typical operators such

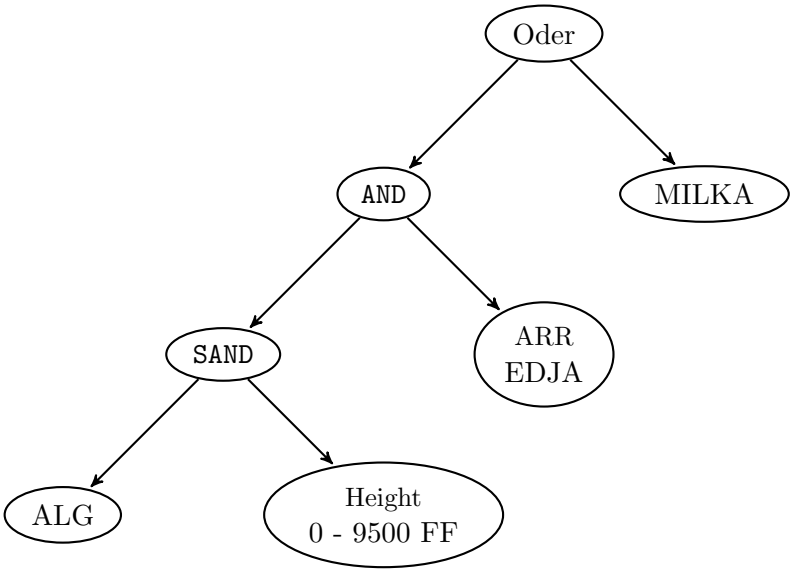

Figure 5.2: Example of a Boolean expression tree *scope* or *condition*.

as `and`, `or` and `not`. For the two intervals, there was a new operator invented: `sand`. `sand` connects a height or time interval with one of the other literals and this shall evaluate to `true` if the interval is respected at the specified literal.

The same operators are allowed for the scope as well. Thus, a rule consists of two Boolean expressions scope and condition and an order. Currently, they are implemented as a binary tree, see Figure 5.2 for an example tree. If a rule in the RAD document consists of two orders, it can be split into two rules with one order each. This is for example the case in the third rule in Table 5.1. This model is the state of the art of the development at Zuse-Institute Berlin for the optimization software for flight trajectories.

The three order types are directly related to forbidden and binding pairs. A compulsory rule is clearly a binding pair, a not available rule is a forbidden pair. And an only available rule can also be seen as a binding pair. If we want to use the scope, we need to satisfy the condition. Thus, to reduce these rules to our model of forbidden and binding pairs, the crucial point is the extraction of concrete waypoints from a condition or scope tree.

We solved this problem just by a heuristic. Thus, we have no guarantee, that all forbidden and binding pairs represent the rules in a good manner. It turns out that about 75% of the rules depend on the departure or arrival. This is the first step our heuristic checks. Then, our heuristic simply collects all waypoint ids of via nodes or airway nodes, which are contained in the Boolean expressions. Often, they contain multiple waypoints (e.g. the second rule in Table 5.1) and hence, the reduction to a vertex pair constraint is abiguous. To keep the heuristic easy, it searches for the very first pair of distinct ids from scope and condition.

The following numbers belong to the instance Sydney to Peking, but the numbers differ only very slightly between the remaining instances. From 8404 restrictions currently loaded in the software, there are 809 binding pairs and 646 forbidden pairs parsed. 6162 restrictions depend on an origin different from Sydney or on a destination different from Peking and hence are not active for this instance. 755 rules are not concerned with explicit graph elements, but with huge airspaces or other conditions. Hence, they can not be modelled as dynamic rules on the graph on the level of segments and waypoints. The last 32 restrictions are concerned with explicit waypoints, but combining it with a height condition, e.g. "If you use waypoint x, use it above flight level 265". These are modelled as compulsory restrictions. Their condition only contains waypoint x, but the scope has "x `sand` above flight level 265".

This heuristic lets one point open. Of course, these restrictions have a priori none of the structure properties presented in section 2.2.3. To achieve this, we formulated an auxiliary integer program. Once the graph is topologically sorted, we can construct a conflict graph G, where the vertices are the vertex pair constraints and an edge is inserted if two pairs halve each other. Consider the following integer program:

$$\begin{aligned} \text{maximize} \quad & z = \sum_{c \in \mathcal{C}} x_c \\ \text{subject to} \quad & x_{c_1} + x_{c_2} \leq 1 && \forall (c_1, c_2) \text{ halving} \\ & x_c \in \{0, 1\} && \forall c \in \mathcal{C} \end{aligned}$$

Our software solves this integer program using the integer program solver SCIP. The solution corresponds to a maximal independent set in G, which itself is a well-paranthesized subset of $\mathcal{C}$. Now, our instance is ready to be solved as a shortest path problem with vertex pair constraints.

5.4 Test Instances

For our test purposes, we picked eight airports worldwide and considered all 56 possible flights between them. The airports are:

1. Atlanta (ATL)
2. Paris (CDG)
3. Dubai (DXB)
4. Frankfurt (FRA)
5. Los Angeles (LAX)
6. London (LHR)
7. Peking (PEK)
8. Sydney (SYD)

We wanted to spread them over the whole world and that we have various different distances between them. The restrictions in the RAD document are

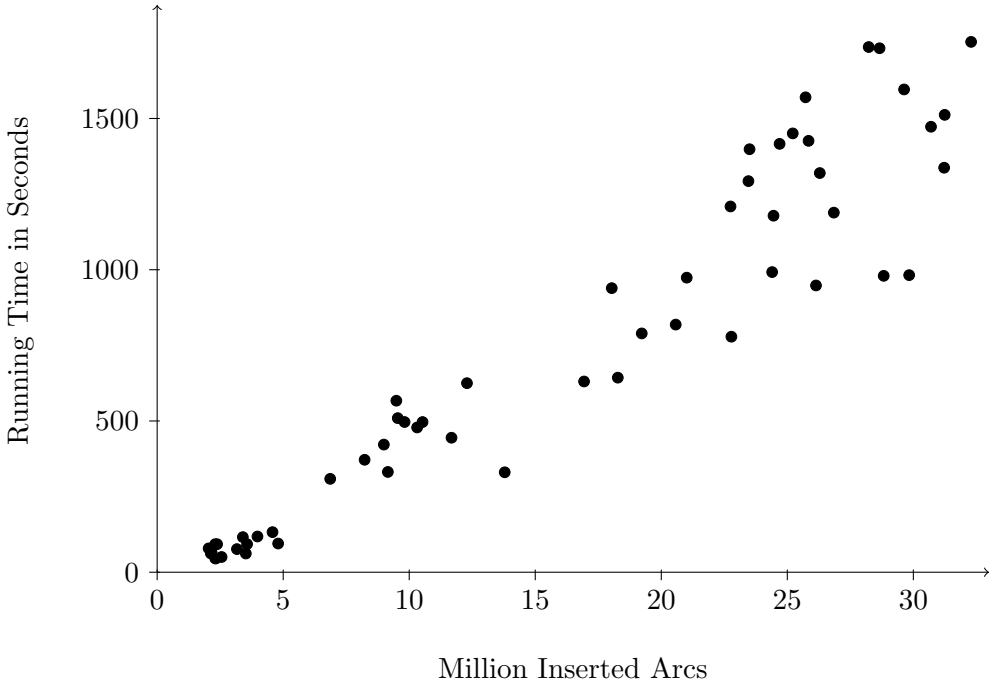

Figure 5.3: The running times depending on the number of inserted arcs.

mostly based in Europe. This is why we tried to have several distances including Europe and others avoiding Europe.

5.5 Results

We started the computations on a DELL Precision 650 machine with 32 kernels having 2.7 GHz each. The machine has 64 Gigabyte RAM internal memory running Ubuntu Linux 14.04. For each of the 56 instances, we started the `contractGraph` tool. For comparison, we also started *SCIP* with the initial IP formulation of the shortest path problem with pair constraints. The result tables can be found in Section 7.1. Table 7.1 lists all running times as well as the size and length of the result trajectories and the instance sizes. In all 56 instances, the trajectories found by `contractGraph` and `SCIP` for the same problem instance were identical.

Unfortunately, the scip computations are essentially faster than the ones of our method `contractGraph`. An interesting result is the column of inserted arcs. This value counts the number of arcs inserted by the application of rule $B2$. Figure 5.3 shows the dependency of the running time to the number of inserted arcs. The chart suggests that there is a more than linear dependency of the running time by the number of inserted arcs. The instances are of almost equal

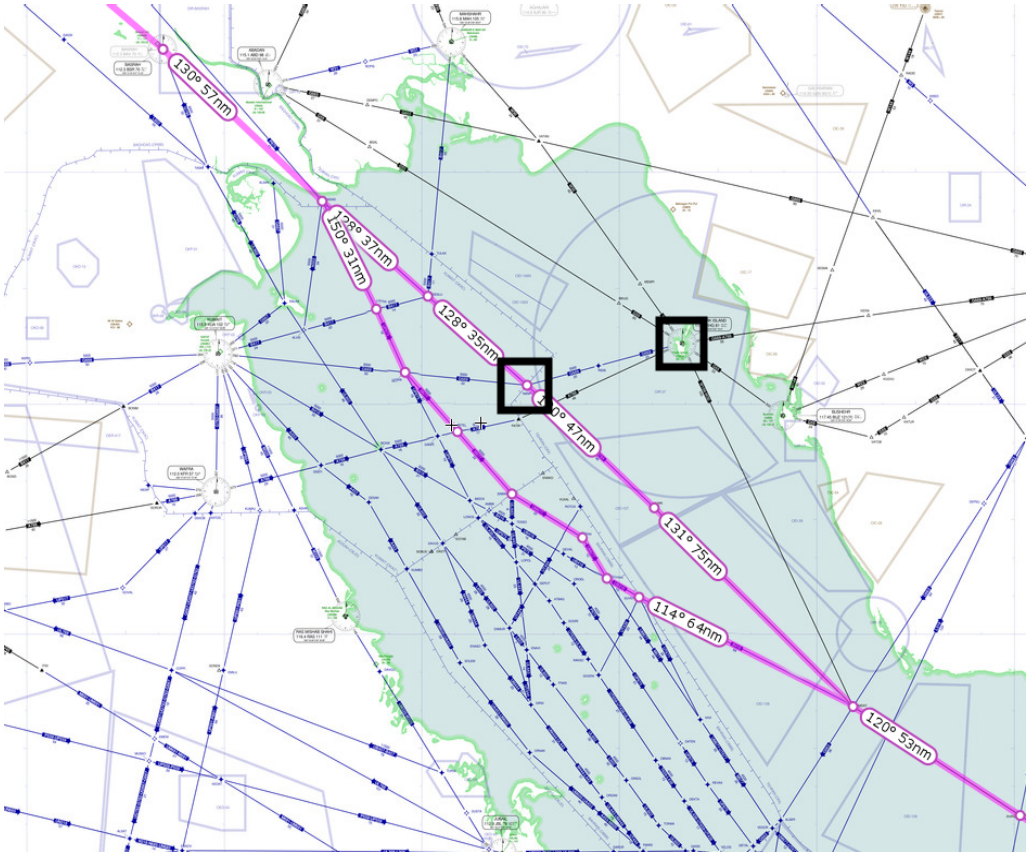

Figure 5.4: Snippet of the trajectories From Paris to Dubai. The right one is unconstrained, the left one respects $\mathcal{C}$. The black boxes mark the responsible binding pair. The picture is made using SkyVector.

size in all of the three data types vertices V, arcs A and constraints $\mathcal{C}$. However, the newly inserted arcs range from 2 millions to over 30 millions. As seen, an arc costs at the very least 56 byte and hence we have more than a Gigabyte main memory usage here. The final trajectories contain up to over hundred vertices and hence all the arc labels have sizes in this scale. This justifies the exorbitant memory usage of `contractGraph` of around 10 Gigabytes or more.

An interesting fact concerns the instance Atlanta to Los Angeles and its backwards direction. The result trajectory here uses only 4 or 5 vertices, but covers a distance of over 3.000 kilometers. The reason for this is, that there are two types of arcs used in the graph. The first one are the normal arcs between the waypoints, ususally in a certain height. The second type are so-called terminal procedures, which connect the airports to several vertices in the graph. These arcs shall represent diffenrent landing or starting routes. Some of these arcs are really long and in this concrete case, the found trajectory only consists of terminal

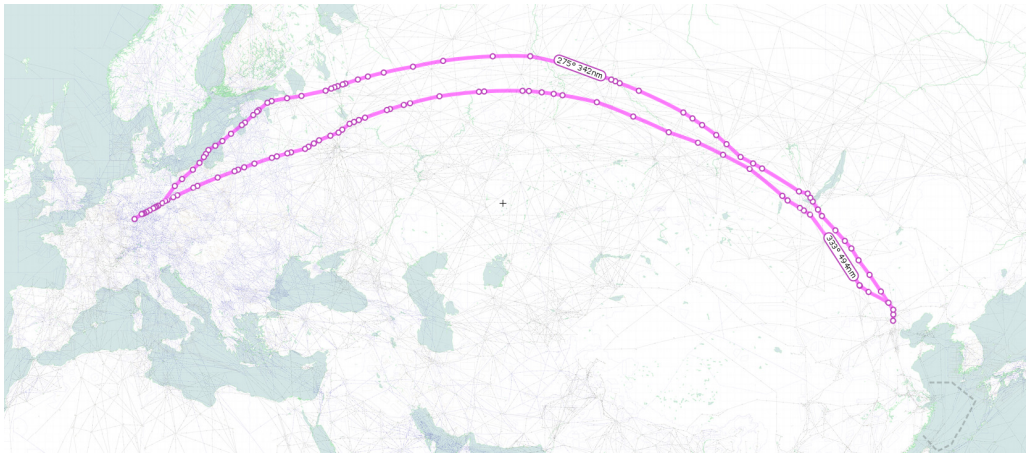

Figure 5.5: The trajectories From Peking to Frankfurt. The lower one is unconstrained, the upper one respects $\mathcal{C}$. The picture is made using SkyVector.

procedures.

We compared the result trajectories to the unconstrained shortest path through the airway network. Figure 5.4 shows a typical result. We see here the local nature of the pair constraints. The snippet is taken from the result trajectories from Paris to Dubai and shows the Persian Gulf. The right trajectory is just the shortest path not respecting any pair constraint. The black squares mark a binding pair. At the black square, the trajectory needs to turn left to Khark Island transforming it into a detour. For this reason, the feasible shortest trajectory avoids the both squares. Sometimes, the influence of the pair constraints is a bit more global, see for example Figure 5.5. In this case, the constrained trajectory (the upper one) takes a completely different route than the unconstrained one. Reason for this is just one forbidden pair shortly before they meet above Leipzig. A surprising fact is that these two trajectories differ by only 10 kilometers which is less than 0.2% of the trajectory length. The second table in Section 7.1 compares all result trajectories with the shortest path through the airway network. For most instances, the length difference is negligible, but sometimes, the values differ up to over 100 kilometers for example from Atlanta to Peking.

6 Concluding Remarks

In this thesis, we tried to approach the shortest path problem with pair constraints. We collected literature to the problem and found a formalism, which allows to formulate both the problems already considered in the literature and the concrete problem, we were concerned in our research. This includes behaviours of the problem under particular restrictions and the connections between these restrictions as well as hardness results for each of them. Furthermore, we presented and observed the combinatorial and recursive algorithms to solve the problem. We extended one of them to our new problem.

To gain further results, we implemented this algorithm and tried to find the advantages of this combinatorial approach against general problem solvers. Unfortunately, the results are not as good as expected. This has two reasons. The first and of course main one is missing expertise in `C++` especially concerning memory management and data structures. The algorithm inserts millions of arcs, which lets the memory needs explode. This leads to a second conclusion. Although this algorithm is of great theoretical interest, since it shows polynomiality to certain instances, its practicability is rather low in comparison to integer program solving methods.

But this even motivates the last aspect of this thesis. We investigated the polytope associated to the integer program formulation we chose. Here we found several complete linear descriptions for the polytopes associated to very simply restricted problem instances. This may even boost the solvability of the shortest path problem with pair constraints using integer programming. Hopefully, our results lay the foundation for new and more advanced observations and properties of polytopes of the shortest path problem with vertex pair constraints.

Finally, we want to spend some words on possibilities for further improvement of the presented results. Our first idea is about the transformation of the real-world data to instances of the pair constraint problem. The current parsing of the restrictions from the RAD document, namely just picking two numbers from scope and condition, is a very rough approach. This can be strongly improved by allowing alterations of the graph. For Considered Airways for example, there could be added an artifical vertex representing the airway. This vertex is connected to the beginning and ending of the airway. The arc labels then contain the vertex ids of the airway. With forbidden or binding pairs, we are now able to access this airway by taking the inserted vertex instead. Once this is contracted, there is no difference between this vertex or the original airway, because the labels would contract to the same sequence of ids.

Another point is the huge loss of constraints by choosing the well-paranthesized

subset. This may be improved by involving the topological sorting into this choice. Currently, we just sort the graph topologically. But since topological sortings are highly not unique, there are maybe much better sortings, which allow larger constraint sets to be well-paranthesized. Maybe, there is a possibility to combine these two steps of the conversion for better constraint sets.

As seen especially in the results section, the number of arcs inserted in the graph during the contraction seems to have no dependence to the key data of an instance. Since this number has a large influence on our computational results, we may be interested in getting this number low. This insertion number seems to purely depend on the structure of the graph or the pair constraints. So perhaps it is possible to find the causes of those high numbers.

7 Appendix

7.1 Running Time Tables

The following table prints all calculation results. To save one column, we organised the data in a hierarchical way: The instances are grouped by there origin. The column **C++** contains the running time of our program `contractGraph`. In **New Arcs** is counted, how many arcs were inserted by the program. The trajectory length $w(\mathcal{P})$ is given in km.

	Time in s			Trajectory		Instance		
To	**C++**	**Scip**	**New Arcs**	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert\mathcal{C}\rvert$
From: **Atlanta** (ATL)								
CDG	92,62	44,15	2.301.727	23	7.138	48.097	174.166	350
DXB	116,08	75,31	3.399.631	80	12.366	48.074	174.065	352
FRA	65,61	26,99	2.160.461	32	7.531	48.080	174.042	366
LAX	939,13	29,49	18.034.559	5	3.126	48.040	174.562	342
LHR	92,68	50,34	2.377.814	22	6.873	48.119	174.092	356
PEK	331,49	47,07	9.152.943	62	11.917	48.156	174.182	350
SYD	444,60	62,41	11.678.129	42	15.151	48.128	174.073	362
From: **Paris** (CDG)								
ATL	1753,24	46,93	32.291.826	31	7.158	48.081	174.908	378
DXB	1570,04	23,17	25.723.711	44	5.282	48.089	174.958	370
FRA	330.39	21,18	13.789.088	12	475	48.108	175.277	346
LAX	1337.22	49,66	31.222.097	38	9.154	48.011	175.158	380
LHR	1732,23	19,34	28.659.428	6	357	48.071	173.968	388
PEK	1736,23	52,98	28.228.764	57	8.303	48.099	175.382	372
SYD	1319.61	77,57	26.289.263	88	17.066	48.082	174.929	380
From: **Dubai** (DXB)								
ATL	1209,19	53,63	22.746.296	64	12.325	48.129	174.691	356
CDG	496,56	22,87	9.815.480	35	5.309	48.083	173.595	352
FRA	422,00	17,20	8.993.525	35	4.885	48.074	173.465	374
LAX	1292,93	90,00	23.457.378	89	13.625	48.142	175.013	360
LHR	509,42	26,50	9.545.688	35	5.540	48.099	173.616	348
PEK	1512,01	19,24	31.243.638	38	5.970	48.189	174.129	360

Continuation ...

Table 7.1: Computational Results.

... Continuation

	Time in s			Trajectory		Instance		
To	**C++**	**Scip**	**New Arcs**	$\|\mathcal{P}\|$	$w(\mathcal{P})$	$\|V\|$	$\|A\|$	$\|\mathcal{C}\|$
SYD	1416, 13	49, 82	24.693.857	48	12.075	48.146	174.600	350
From: **Frankfurt** (FRA)								
ATL	1472, 67	39, 47	30.700.840	39	7.560	48.129	174.925	366
CDG	973, 78	24, 57	21.009.877	11	459	48.117	173.487	374
DXB	1178, 60	43, 51	24.452.745	43	4.888	48.128	174.981	356
LAX	1596, 00	45, 59	29.632.553	28	9.370	48.070	175.252	366
LHR	1426, 12	18, 72	25.842.350	17	683	48.149	173.852	374
PEK	1188, 78	31, 64	26.845.858	55	7.906	48.145	175.435	360
SYD	1450, 69	132, 32	25.215.980	98	16.621	48.128	174.972	364
From: **Los Angeles** (LAX)								
ATL	625, 02	23, 15	12.291.208	4	3.127	48.205	174.639	360
CDG	78, 75	54, 03	2.041.704	35	9.137	48.148	173.840	362
DXB	76, 44	53, 22	3.165.033	77	13.659	48.066	173.667	368
FRA	77, 90	56, 91	2.086.444	31	9.365	48.127	173.639	370
LHR	62, 18	33, 50	2.129.078	38	8.794	48.170	173.858	352
PEK	308, 66	65, 38	6.865.417	51	10.172	48.173	173.867	354
SYD	496, 50	103, 14	10.528.883	25	12.072	48.102	173.697	350
From: **London** (LHR)								
ATL	982, 13	22, 78	29.834.410	26	6.878	48.061	174.744	328
CDG	371, 70	12, 95	8.230.523	9	354	48.158	174.665	338
DXB	1398, 40	43, 30	23.501.647	49	5.546	48.066	174.699	330
FRA	478, 55	15, 76	10.313.985	14	666	48.064	174.674	352
LAX	979, 84	28, 42	28.824.565	35	8.804	47.988	174.902	324
PEK	948, 00	38, 83	26.138.014	58	8.287	48.072	175.086	332
SYD	992, 24	77, 22	24.398.355	79	17.148	48.059	174.655	346
From: **Peking** (PEK)								
ATL	643, 20	61, 73	18.274.964	57	11.802	48.111	174.651	388
CDG	118, 09	25, 41	3.979.062	61	8.280	48.050	173.248	366
DXB	94, 83	33, 08	4.800.026	30	5.995	48.023	174.183	376
FRA	92, 78	45, 40	3.567.470	71	7.910	48.032	173.088	346
LAX	778, 59	45, 20	22.781.757	44	10.265	48.091	174.890	342
LHR	132, 51	27, 26	4.577.116	71	8.265	48.076	173.306	336
SYD	818, 61	33, 87	20.572.401	55	9.395	48.066	174.344	346
From: **Sydney** (SYD)								
ATL	630, 56	41, 39	16.937.011	45	15.161	48.131	174.690	370
CDG	45, 55	86, 42	2.326.094	111	17.095	48.094	173.631	354

Continuation ...

Table 7.1: Computational Results.

... Continuation

	Time in s			Trajectory		Instance		
To	**C++**	**Scip**	**New Arcs**	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	$\lvert V\rvert$	$\lvert A\rvert$	$\lvert\mathcal{C}\rvert$
DXG	61, 82	22, 75	3.517.430	49	12.067	48.058	173.642	366
FRA	45, 84	117, 64	2.304.678	108	16.658	48.080	173.481	340
LAX	789, 43	49, 56	19.225.051	24	12.072	48.141	174.990	374
LHR	50, 33	51, 78	2.555.777	95	17.156	48.110	173.667	362
PEK	567, 02	28, 77	9.488.945	43	9.475	48.210	173.975	360

Table 7.1: Computational Results.

The next table compares two trajectories: The first one respects all pair constraints in $\mathcal{C}$, the second one is just the shortest (unconstrained) connection between them. The length $w(\mathcal{P})$ is again given in km.

	$\mathcal{C}$		$\mathcal{C}=\emptyset$			$\mathcal{C}$		$\mathcal{C}=\emptyset$	
To	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	**To**	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$
From: **Atlanta** (ATL)					From: **Los Angeles** (LAX)				
CDG	23	7.138	23	7.138	ATL	4	3.127	4	3.127
DXB	80	12.366	68	12.338	CDG	35	9.137	35	9.137
FRA	32	7.531	32	7.531	DXB	77	13.659	76	13.633
LAX	5	3.126	5	3.126	FRA	31	9.365	31	9.365
LHR	22	6.873	22	6.873	LHR	38	8.794	39	8.794
PEK	62	11.917	63	11.805	PEK	51	10.172	51	10.172
SYD	42	15.151	42	15.151	SYD	25	12.072	25	12.072
From: **Paris** (CDG)					From: **London** (LHR)				
ATL	31	7.158	33	7.147	ATL	26	6.878	26	6.878
DXB	44	5.282	40	5.268	CDG	9	354	10	354
FRA	12	475	10	461	DXB	49	5.546	42	5.526
LAX	38	9.154	37	9.149	FRA	14	666	13	660
LHR	6	357	8	353	LAX	35	8.804	35	8.804
PEK	57	8.303	57	8.303	PEK	58	8.287	58	8.287
SYD	88	17.066	115	17.058	SYD	79	17.148	99	17.147
From: **Dubai** (DXB)					From: **Peking** (PEK)				
ATL	64	12.325	64	12.325	ATL	57	11.802	57	11.802
CDG	35	5.309	35	5.309	CDG	61	8.280	61	8.280
FRA	35	4.885	35	4.885	DXB	30	5.995	29	5.970
LAX	89	13.625	90	13.625	FRA	71	7.910	68	7.900
LHR	35	5.540	35	5.540	LAX	44	10.265	44	10.213
PEK	38	5.970	39	5.970	LHR	71	8.265	71	8.265

Continuation ...

Table 7.2: Comparison of the trajectories respecting $\mathcal{C}$ and the shortest path.

... Continuation

	$\mathcal{C}$		$\mathcal{C} = \emptyset$			$\mathcal{C}$		$\mathcal{C} = \emptyset$	
To	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	**To**	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$	$\lvert\mathcal{P}\rvert$	$w(\mathcal{P})$
SYD	48	12.075	48	12.075	SYD	55	9.395	55	9.395
From: **Frankfurt** (FRA)					From: **Sydney** (SYD)				
ATL	39	7.560	39	7.560	ATL	45	15.161	45	15.161
CDG	11	459	11	459	CDG	111	17.095	121	17.092
DXB	43	4.888	37	4.874	DXB	49	12.067	49	12.067
LAX	28	9.370	28	9.362	FRA	108	16.658	110	16.647
LHR	17	683	17	683	LAX	24	12.072	24	12.072
PEK	55	7.906	55	7.906	LHR	95	17.156	94	17.156
SYD	98	16.621	113	16.606	PEK	43	9.475	48	9.430

Table 7.2: Comparison of the trajectories respecting $\mathcal{C}$ and the shortest path.

7.2 List of Figures

7.3 List of Tables

7.4 List of Algorithms

7.5 References

[1] European Environment Agency. "EEA Briefing 3 / 2004 – Verkehr und Umwelt in Europa". In: *Kopenhagen* (2004).

[2] G. Ausiello. *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*. Springer-electronic-Media. U.S. Government Printing Office, 1999. ISBN: 9783540654315. URL: https://books.google.de/books?id=Yxxw90d9AuMC.

[3] Egon Balas. "Disjunctive programming". In: *Annals of Discrete Mathematics* 5 (1979), pp. 3–51.

[4] Egon Balas. "Disjunctive programming: Properties of the convex hull of feasible points". In: *Discrete Applied Mathematics* 89.1 (1998), pp. 3–44.

[5] Claude Berge. "Two theorems in graph theory". In: *Proceedings of the National Academy of Sciences of the United States of America* 43.9 (1957), p. 842.

[6] D. Bertsimas and R. Weismantel. *Optimization Over Integers*. Dynamic Ideas, 2005. ISBN: 9780975914625. URL: https://books.google.de/books?id=De2nQAAACAAJ.

[7] Statistisches Bundesamt. "Luftverkehr – Fachserie 8 Reihe 6 – Dezember 2014". In: (2014).

[8] Ting Chen et al. "A dynamic programming approach to de novo peptide sequencing via tandem mass spectrometry". In: *Journal of Computational Biology* 8.3 (2001), pp. 325–337.

[9] T.H. Cormen et al. *Introduction To Algorithms*. MIT Press, 2001. URL: https://books.google.de/books?id=NLngYyWFl_YC.

[10] C. Demetrescu, A.V. Goldberg, and D.S. Johnson. *The Shortest Path Problem: Ninth DIMACS Implementation Challenge*. DIMACS series in discrete mathematics and theoretical computer science. American Mathematical Soc., 2009. ISBN: 9780821885864. URL: https://books.google.de/books?id=Vz0Syt3VqAcC.

[11] Eurocontrol. *Route Availability Document*. Eurocontrol. URL: http://www.nm.eurocontrol.int/RAD/index.html.

[12] Harold N. Gabow, Shachindra N Maheshwari, and Leon J. Osterweil. "On Two Problems in the Generation of Program Test Paths". In: *Software Engineering, IEEE Transactions on* 2.3 (1976), pp. 227–231. DOI: http://doi.ieeecomputersociety.org/10.1109/TSE.1976.233819.

[13] MohammadTaghi Hajiaghayi et al. “The checkpoint problem”. In: *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 2010, pp. 219–231.

[14] Johan Håstad. “Some optimal inapproximability results”. In: *Journal of the ACM (JACM)* 48.4 (2001), pp. 798–859.

[15] R. W. Smith K. A. Krause and M. A. Goodwin. “Optimal Software test planning through automated network analysis”. In: *Proc. 1973 IEEE Symp. Computer Software Reliability* (1973), pp. 18–22.

[16] D.E. Knuth. *The art of computer programming: Fundamental algorithms. Vol. 1*. Addison-Wesley, 2008. ISBN: 9780201485417. URL: https://books.google.de/books?id=0XFitQAACAAJ.

[17] D.E. Knuth. *The Art of Computer Programming: Volume 3, The: Sorting and Searching*. Pearson Education, 1998. ISBN: 9780321635785. URL: https://books.google.de/books?id=cYULBAAAQBAJ.

[18] Petr Kolman and Ondřej Pangrác. “On the complexity of paths avoiding forbidden pairs”. In: *Discrete Applied Mathematics* 157.13 (2009), pp. 2871–2876.

[19] Guy Kortsarz. “On the hardness of approximating spanners”. In: *Algorithmica* 30.3 (2001), pp. 432–450.

[20] Jakub Kováč. “Complexity of the path avoiding forbidden pairs problem revisited”. In: *Discrete Applied Mathematics* 161.10 (2013), pp. 1506–1512.

[21] Jakub Kováč, Tomáš Vinař, and Broňa Brejová. “Predicting Gene Structures from Multiple RT-PCR Tests”. English. In: *Algorithms in Bioinformatics*. Vol. 5724. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2009, pp. 181–193. ISBN: 978-3-642-04240-9. DOI: 10.1007/978-3-642-04241-6_16.

[22] G.L. Nemhauser and L.A. Wolsey. *Integer and combinatorial optimization*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 1988. ISBN: 9780471828198. URL: https://books.google.de/books?id=uG4PAQAAMAAJ.

[23] S.C. Ntafos and S.L. Hakimi. “On Path Cover Problems in Digraphs and Applications to Program Testing”. In: *Software Engineering, IEEE Transactions on* SE-5.5 (Sept. 1979), pp. 520–529. ISSN: 0098-5589. DOI: 10.1109/TSE.1979.234213.

[24] T. Ottmann and P. Widmayer. *Algorithmen und Datenstrukturen*. Spektrum Akademischer Verlag GmbH, 1996. ISBN: 3-8274-0110-0. URL: https://books.google.de/books?id=wLkkBAAAQBAJ.

[25] Christoph Spiegel. “Approximating Primitive Integrals and Aircraft Performance”. MA thesis. Freie Universität Berlin, 2015.

[26] Xing Tan. "The Application of Ontologies to Reasoning with Process Modeling Formalisms". PhD thesis. University of Toronto, 2012.

[27] Hananya Yinnone. "On paths avoiding forbidden pairs of vertices in a graph". In: *Discrete applied mathematics* 74.1 (1997), pp. 85–92.

[28] G.M. Ziegler. *Lectures on Polytopes*. Graduate Texts in Mathematics. Springer New York, 1995. ISBN: 9780387943657.