Stefan Lendl, DI

Generalizations of Classic Combinatorial Optimization Problems on Graphs and Matroidal Structures: Algorithms and Complexity

DOCTORAL THESIS

to achieve the university degree of

Doktor der technischen Wissenschaften

submitted to

Graz University of Technology

Supervisor

Ao.Univ.-Prof. Dipl.-Ing. Dr.techn. Bettina Klinz

Institute of Discrete Mathematics

Graz, June 2019

AFFIDAVIT

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present doctoral thesis.

Date

Signature

Contents

Acknowledgements

I would like to thank my advisor Bettina Klinz for her guidance, support and being available to help in any kind of situation throughout my doctoral studies. Being always able to just knock on her door whenever necessary was just one of many things that made doing a PhD under her supervision a great experience. Specifically, I also want to thank her for her trust, allowing me to freely work on problems I liked, while always giving guidance when needed. I am also very grateful for the research ideas she shared with me and the opportunity for joint work. Beyond my doctoral studies, I thank Bettina Klinz for the exciting advanced lectures she taught during my master studies, which were an important reason and motivation for my wish to pursue a further eduction and research in combinatorial optimization.

Special thanks go to Abraham Punnen for hosting me at SFU Vancouver and Ante Ćustić for initiating the contact and taking care of everything I needed during my stay in Vancouver. I am thankful to both of them for sharing their research ideas and inviting me to do joint work. Furthermore, I am also thankful to all the other colleagues at SFU with whom I had many fruitful discussions during regular research seminars and joint coffees at SFU.

Special thanks also go to Gerhard Woeginger for his hospitality during my two visits at RWTH Aachen and for sharing his time and ideas with me. I am grateful that I have had the opportunity to work with him on multiple projects, on which part of this thesis is based on. His inputs during my stays in Aachen and his visits in Graz not only led to joint work, but also sparked new research collaborations and motivated me to study new tools and techniques. I am very thankful to Britta Peis who also invited me to her research group at RWTH Aachen during my first stay and for hosting me during my second stay in Aachen. I am grateful for all the research ideas she has shared with me, and for being able to do joint work with her. At this point, I also want to thank the groups of Gerhard Woeginger and Britta Peis for making my stays in Aachen a great experience. Special thanks go to Tim Hartmann and Veerle Tan-Timmermans for being great co-authors, with whom collaborations started because of my visits to Aachen.

I also thank Vladimir Deineko for sharing many ideas whenever visiting TU Graz during the summer. His enthusiasm when talking about new research ideas is always a joy and very motivating. I am grateful for the opportunity to do joint work with him.

I am lucky and thankful that Eranda Dragoti-Çela is part of our Combinatorial Optimization Group at TU Graz and also one of my mentors in the DK project. I am also grateful for the opportunity to do joint work with her. Her door was always open to me for any kind of questions and our chats made my PhD studies more enjoyable. Beyond my doctoral studies, I want to thank Eranda Dragoti-Çela for the inspiring optimization lectures she taught during my bachelor and master studies, which sparked and further

kindled my interest in combinatorial optimization.

Special thanks also go to Thomas Lachmann for many inspiring discussions about diverse topics in mathematics and being a great co-author.

I would also like to thank all other colleagues with whom I had the opportunity to do joint work going beyond the scope of this thesis. It is always a joy and I am grateful to do research with all of you, on a diverse range of subjects.

Beyond my doctoral studies, I would also like to thank Johannes Hatzl, my master thesis supervisor, for his motivation to pursue doctoral studies and for his continuing support and advise, also during this period.

I also want to thank the other members of our institute. In addition to many interesting scientific discussions we shared a lot of good time during lunches, joint coffees and other activities. Thanks also goes to our secretaries, especially Sandra Wissler, for knowing everything I ever needed to know on administrative matters and being always helpful and a joy to talk to.

I thank all the members of the DK Discrete Mathematics for making my time as part of the project a joyful experience. In particular, I thank my mentors Rainer Burkard and Oswin Aichholzer for their support throughout my doctoral studies and our speaker Wolfgang Woess.

At this point I want to acknowledge the support of the Austrian Science Fund (FWF): W1230, which made this thesis possible.

Special thanks go to Franz Rendl and Frits C.R. Spieksma for the time they invested into refereeing this thesis.

And last but not least, I thank my family and friends for their constant support and help during the whole period of writing this thesis and beyond.

1 Introduction and Preliminaries

All problems studied in this PhD thesis fit into the general framework of combinatorial optimization problems and combinatorial decision problems. Let E be the ground set and $\mathcal{F} \subseteq 2^E$ be a set of *feasible solutions*. The problem to decide about the existence of

$$S \in \mathcal{F}$$

is referred to as a *combinatorial decision* problem. If, in addition, there is given an *objective function* (or *cost function*) $f\colon 2^E \to \mathbb{R}$ the problem

$$\begin{aligned} \min \quad & f(S) \\ \text{s.t.} \quad & S \in \mathcal{F} \end{aligned}$$

is referred to as *combinatorial optimization* problem. In case of linearity of the cost function f, i.e. there exist $c(e) \in \mathbb{R}$ for each $e \in E$ and $f(S) = \sum_{e \in S} c(e)$, the given problem is a *linear* combinatorial optimization problem (LCOP). We denote an instance of LCOP by $(\mathcal{F}, c)$. A central aspect in combinatorial optimization is to find efficient solution algorithms or to understand why such algorithms most likely cannot exist. For a general introduction into the subject we refer the reader to the books by Korte and Vygen [88] and Schrijver [111].

In the remaining part of the introduction we will briefly introduce some general concepts used in the upcoming chapters of the thesis. These concepts are also the main focus points of this thesis:

- combinatorial structures based on graphs and submodular functions,
- different ways to generalize classic combinatorial optimization problems,
- understanding the computational complexity of combinatorial optimization problems depending on diverse influencing factors.

Moreover, we will give an overview of the problems studied in this thesis together with the main results obtained.

1.1 Preliminaries

1.1.1 Sets, Matrices, Vectors and Polyhedra

Let E be a set, the *ground set*, and $A, B \subseteq E$ subsets, then we denote by $|A|$ the *cardinality* of A, by $A \cap B, A \cup B, A \setminus B$ the *intersection*, *union* and *difference* of A and

B. We write $A^C = \bar{A} = E \setminus A$ for the *complement* of A in E. The *symmetric difference* of A and B is denoted by $A \Delta B = (A \setminus B) \cup (B \setminus A)$. The *power set* $2^E = \{S \colon S \subseteq E\}$ of E is the set of all subsets of E.

Let $\mathbb{K}$ be an arbitrary set (for example $\mathbb{K} = \mathbb{R}$ or $\mathbb{K} = \{0, 1\}$) and $m, n \in \mathbb{N}$, then we write $\mathbb{K}^n$ for the set of all vectors with n entries and $\mathbb{K}^{m \times n}$ for the set of all $(m \times n)$-matrices with entries from $\mathbb{K}$. If M, N are sets with $|M| = m$ and $|N| = n$, we also write $\mathbb{K}^N$ and $\mathbb{K}^{M \times N}$ to denote vectors and matrices where the entries are identified with elements and pairs of elements of the given sets. In case of $\mathbb{K}$ being a field and $A \in \mathbb{K}^{m \times n}$ a given matrix over $\mathbb{K}$ we say that a set of columns $c_1, \dots c_l$ of A is *linearly independent*, if the only solution to the equation $\sum_{i=1}^{l} \lambda_i c_i = 0$ is $\lambda = 0$.

Given a matrix $A \in \mathbb{R}^{m \times n}$ and a vector $b \in \mathbb{R}^m$ then the set $P = \{x \in \mathbb{R}^n \colon Ax \leq b\}$ is called a *polyhedron*. For $c \in \mathbb{R}^n$ being a given *cost vector*, the optimization problem

$$\begin{aligned} \min \quad & \sum_{i=1}^{n} c_i x_i \\ \text{s.t.} \quad & x \in P \end{aligned}$$

is referred to as LINEAR PROGRAMMING problem. If in addition one adds the constraint that $x \in \mathbb{Z}^n$ one arrives at the INTEGER (LINEAR) PROGRAMMING problem, and if instead $x \in \{0, 1\}$ the BINARY PROGRAMMING problem results. Lastly, we call the problem MIXED INTEGER/BINARY PROGRAMMING problem, if the binary or integer constraints only appear for a subset of the variables.

1.1.2 Graphs

Let V be an arbitrary set, the set of *vertices*, and E a set of pairs $\{u, v\}$ of vertices $u, v \in V$, the set of *edges*. Then we call $G = (V, E)$ an (undirected) graph. If not otherwise specified, we always denote by G a graph, by V the vertex set of G and by E the edge set of G. For $v \in V$ we denote by $d(v)$ the *degree* of v. We call $G = (V, A)$ a *directed graph* (*digraph*) if the set A is a set of tuples (u, v) of vertices $u, v \in V$. We call (u, v) an *arc* of G from u to v. In some cases we use more general definitions of graphs/digraphs, where multiple edges/arcs connecting the same vertices (so called *multiedges*/*multiarcs*) or edges/arcs from one vertex to itself (*loops*) are allowed. If this distinction is of relevance for the arguments and results we clarify this in the text.

In (algorithmic) graph theory the study of structures and properties of graphs plays a central role. In the following we will only list a few basic concepts which play a role in this thesis. We will concentrate on the case of undirected graphs. For many concepts directed variants exist as well.

A list of vertices $P = (v_1, v_2, \dots, v_l)$ is a (simple) *path* if those vertices are pairwise distinct and for each $j = 1, 2, \dots, l-1$ it holds that $\{v_j, v_{j+1}\} \in E$. The start of P is denoted by $\text{start}(P) = v_1$ and the end of P is denoted by $\text{end}(P) = v_l$. We define $\text{rev}(P)$ as the reverse path $(v_l, v_{l-1}, \dots, v_1)$ of P. If in addition $\{v_l, v_1\} \in E$ we call P a (simple) *cycle*. We also call a graph consisting only of a unique path/cycle a path/cycle. For two vertices $s, t \in V$ we say that s and t are *connected* if there exists a path P in G such

that $s = \text{start}(P)$ and $t = \text{end}(P)$. The set of all paths from s to t (s-t-paths) in G is denoted by $\mathcal{P}_{s,t}(G)$. A graph G is *connected* if every pair $s, t \in V$ is connected. A graph is called a *tree* if it is connected and does not contain a cycle.

A subgraph $T = (V_T, E_T)$ of a connected graph G is a *spanning tree* of G if $V_T = V$, and T is a tree. For a general graph G a *spanning forest* is a subgraph that consists of a union of spanning trees, one for every connected component of G.

Furthermore, we refer to a subgraph $C = (V_C, E_C)$ with $V_C = V$ that is a cycle/path is as a *Hamiltonian cycle/path*. A graph containing a Hamiltonian cycle is called *Hamiltonian*.

A set of vertices $I \subseteq V$ for which for all $u, v \in I$ it holds that $\{u, v\} \notin E$, is called an *independent set* of G

A subset $M \subseteq E$ of edges of a graph is called a *matching* if the edges in M are pairwise non-adjacent, meaning they have no common vertices. A matching M of maximum cardinality is called a *maximum matching* and we denote by $\nu(G)$ the cardinality of a maximum matching in G, which is called the *matching number*. If $\nu(G) = |V|/2$ we say that G contains a *perfect matching*. The set of all perfect matchings in G is denoted by $\mathcal{PM}(G)$.

Given a subset $S \subseteq V$ the set $\delta(S) = \{\{u, v\} \in E \colon u \in S, v \notin S\}$ is called the cut of G induced by S.

Directed variants of those structures exist and are introduced when needed in the following chapters.

The question if structures of the types mentioned above exist in a graph gives rise to many classic combinatorial decision problems. Given a linear cost function on the vertices or edges of the graph, finding such structures of minimum cost or maximum cardinality are classic combinatorial optimization problems.

In many cases, it is important to study (these problems on) graphs/digraphs with special properties. In the following we introduce some special graph classes which are relevant in this thesis. Given a partition of the set of vertices $V = U \dot\cup W$ we call a graph $G = (V, E)$ *bipartite*, if for all edges $\{u, w\} \in E$ it holds that $u \in U$ and $w \in W$. We also denote such a bipartite graph by $G = (U, W, E)$. Let $I = (i_1, i_2, \ldots, i_n)$ be a list or set of intervals on the real line. We denote by $G(I)$ the *interval graph* induced by I. The vertices of this graph correspond to the intervals in I. Two intervals $i, i' \in I$ are connected by an edge in $G(I)$ if $i \cap i' \neq \emptyset$.

1.1.3 Matroids and Polymatroids

Given a *ground set* E and $\mathcal{I} \subseteq 2^E$ we call $(E, \mathcal{I})$ an *independence system*, if

(M1) $\emptyset \in \mathcal{I}$,

(M2) $A \subseteq B$ and $B \in \mathcal{I}$ then $A \in \mathcal{I}$

hold. If in addition the *exchange property*,

(M3) $A, B \in \mathcal{I}$ and $|A| > |B|$, then there is an $a \in A \setminus B$ such that $B \cup \{a\} \in \mathcal{I}$

holds, we call $(E, \mathcal{I})$ a *matroid*. Note that matroids are a common generalization of sets of linearly independent columns of matrices and subforests in graphs as the following examples show.

- Let $G = (V, E)$ be a graph and $\mathcal{I}$ be the set of all subsets of edges $F \subseteq E$ such that $G[F]$ is acyclic. Then $\mathcal{M}(G) := (E, \mathcal{I})$ fulfills (M1)–(M3) and is called *graphic matroid*.
- Let E be the set of columns of the matrix A over a field $\mathbb{F}$ and $\mathcal{I}$ be the set of all subsets $F \subseteq E$ such that the columns in F are linearly independent over the field $\mathbb{F}$. Then $(E, \mathcal{I})$ is called a *linear matroid* or *vector matroid*, and we say that $(E, \mathcal{I})$ is *representable* over $\mathbb{F}$.
- Let E be an arbitrary set of elements, $k \in \mathbb{N}$ and $\mathcal{I} := \{S \subseteq E \colon |S| \leq k\}$. Then we call $(E, \mathcal{I})$ a *uniform matroid*.

Given a matroid $\mathcal{M} = (E, \mathcal{I})$ the function $\mathrm{rk} \colon 2^E \to \mathbb{R}$ defined by

$$\mathrm{rk}(X) := \max\{|S| \colon S \subseteq X, S \in \mathcal{I}\},$$

for each $X \subseteq E$, is called the *rank function* of $\mathcal{M}$ and $\mathrm{rk}(X)$ is called the *rank* of X. The maximal independent sets of a matroid, i.e. the sets $B \subseteq E$ such that $\mathrm{rk}(B) = \mathrm{rk}(E)$, are called *bases* of $\mathcal{M}$. Usually, we denote the set of all bases of a matroid by $\mathcal{B}$. It is important to note that a matroid $\mathcal{M} = (E, \mathcal{I})$ can be equivalently defined via its set of bases or its rank function, hence we also write $\mathcal{M} = (E, \mathcal{I}) = (E, \mathcal{B}) = (E, \mathrm{rk})$.

A set function $f \colon 2^E \to \mathbb{R}$ is called *submodular*, if for all $X, Y \subseteq E$

$$f(X \cup Y) + f(X \cap Y) \leq f(X) + f(Y).$$

A set function $f \colon 2^E \to \mathbb{R}$ is called monotone, if

$$f(X) \leq f(Y)$$

for all $X \subseteq Y \subseteq E$. An important example of monotone and submodular functions are the rank functions of a matroid.

We call (E, f) a *polymatroid* if f is a monotone, submodular function and $f(\emptyset) = 0$. The *base polytope* of the polymatroid is defined as

$$\mathcal{B}(f) = \{x \in \mathbb{R}_+^E \colon x(S) := \sum_{e \in S} x_e \leq f(S), S \subseteq E\}.$$

If f is a rank function of a matroid, the vertices of $\mathcal{B}(f)$ are exactly the set of incidence vectors of the bases of the matroid.

Matroids are exactly characterized as the independence systems for which the greedy algorithm solves the minimum cost basis problem. This result was proven by Korte and Monma [87] and can be directly generalized to polymatroids.

For an in-depth treatment of matroids we refer the reader to the book of Oxley [103]. For an introduction to submodular functions and polymatroids see the book of Fujishige [55].

1.1.4 Computational Complexity and Algorithms

The key tools for understanding the computational complexity of problems are algorithms and hardness proofs. An algorithm with a certain running time gives an upper bound on the time complexity of the problem it solves. A commonly used terminology in the theoretical computer science and optimization literature is that problems which admit algorithms with a running time bounded by a polynomial are called "efficiently solvable". This also matches with the definition of the complexity class P (for decision problems). However, in some cases we are interested in a more detailed analysis of the running time and aim for fast algorithms (e.g. linear). Chapter 3 and Section 6.1 are examples where we follow this line of research.

On the negative side we mainly lack a set of tools to show good lower bounds for the running time of algorithms that solve a given problem. What we can do is showing that an efficient algorithm for one problem implies also the existence of an efficient algorithm for other problems. The complexity class NP contains many natural problems for which no polynomial time algorithms are known, although numerous attempts were made over the last decades. We call a problem NP-hard if a polynomial time algorithm that solves this problem would imply polynomial time algorithms for all problems in NP. Under the usual assumption that P $\neq$ NP, proving NP-hardness is our main tool for showing that no polynomial time algorithm exists for a problem.

For a formal and more detailed introduction to the concepts of computational complexity theory, we refer the reader to the books of Papadimitriou [104] and Arora and Barak [7].

1.1.5 Robust Optimization

The central idea of robust optimization is introducing so called *uncertainty sets*. Instead of a fixed cost function $c\colon 2^E \to \mathbb{R}$ we are given a scenario set $\mathcal{U}$. The elements of this set are cost functions $c^s\colon 2^E \to \mathbb{R} \in \mathcal{U}$. Usually, we assume that the cost functions are linear, i.e. $c^s(S) = \sum_{e\in S} c^s(e)$, where $c^s(e) \in \mathbb{R}$ for each $e \in E$. Common examples for uncertainty sets from the literature are:

Discrete Uncertainties

$$\mathcal{U}^D = \{c^{s_1}, c^{s_2}, \dots, c^{s_K}\}$$

for given $c^{s_1}, \dots, c^{s_K}\colon E \to \mathbb{R}$.

Interval Uncertainties

$$\mathcal{U}^I = \{c^s\colon c^s(e) \in [\underline{c}(e), \bar{c}(e)], e \in E\}$$

for given $\underline{c}, \bar{c}\colon E \to \mathbb{R}$.

Budgeted Interval Uncertainties

$$\mathcal{U}_1^I(\Gamma) = \left\{ c^s\colon c^s(e) \in [c(e), c(e) + \delta_e d(e)], \delta_e \in \{0,1\}, e \in E, \sum_{e\in E} \delta_e \leq \Gamma \right\},$$

$$\mathcal{U}_2^I(\Gamma) = \left\{ c^s \colon c^s(e) = c(e) + \delta_e, \delta_e \in [0, d(e)], e \in E, \sum_{e \in E} \delta_e \leq \Gamma \right\}$$

for given $c, d \colon E \to \mathbb{R}$ and $\Gamma \in \mathbb{R}_+$ a given budget.

The classic assumption in robust optimization is as follows. One does not know which of the possible scenarios occur and one tries to find a good solution by hedging for the worst case. This leads to the following two classic robust optimization models.

Min-Max Robust Optimization
Input: Uncertainty set $\mathcal{U}$, feasible solutions $\mathcal{F} \subseteq 2^E$
Question: Find an optimal solution for

$$\min_{S \in \mathcal{F}} \max_{c^s \in \mathcal{U}} c^s(S).$$

Min-Max Regret Robust Optimization
Input: Uncertainty set $\mathcal{U}$, feasible solutions $\mathcal{F} \subseteq 2^E$
Question: Find an optimal solution for

$$\min_{S \in \mathcal{F}} \max_{c^s \in \mathcal{U}} (c^s(S) - c^{s,*}),$$

where $c^{s,*} = \min_{S \in \mathcal{F}} c^s(S)$.

For a more detailed introduction to the field of discrete robust optimization we refer the reader to the book of Kouvelis [89].

1.2 Key Problems in this Thesis

In the following we briefly introduce the key problems studied in this thesis. For more details we refer to the respective chapters.

The main topic of Chapter 2 is the δ-dispersion problem on graphs. Formally, let $G = (V, E)$ be an undirected connected graph, where every edge is rectifiable and has unit length. Let $P(G)$ denote the continuum set of points on all the edges in E together with all the vertices in V. For two points $p, q \in P(G)$, we denote by $d(p, q)$ the length of a shortest path connecting p and q in the graph. A subset $S \subseteq P(G)$ is said to be *δ-dispersed* for some positive real number δ, if any two points $p, q \in S$ with $p \neq q$ are at distance $d(p, q) \geq \delta$ from each other.

δ-Dispersion
Input: Graph $G = (V, E)$, $\delta > 0$
Question: Find a maximum cardinality subset $S \subseteq P(G)$ that is δ-dispersed.

In Chapter 3 we focus on obtaining fast algorithms for the Steiner cycle problem restricted to interval graphs. Given a graph $G = (V, E)$ and a Steiner set $S \subseteq V$ a cycle C is a Steiner cycle, if $S \subseteq C$.

Steiner Cycle
Input: Graph $G = (V, E)$, Steiner points $S \subseteq V$
Question: Does there exist a Steiner cycle with respect to S in G?

In Chapter 4 we study combinatorial optimization problems in the presence of so-called *interaction costs*, which are defined for two solution sets S_1, S_2 over ground sets E_1, E_2 by a matrix $Q = (q_{i,j}) \in \mathbb{R}^{E_1 \times E_2}$ as

$$c(S_1, S_2) := \sum_{i \in S_1} \sum_{j \in S_2} q_{i,j}.$$

Combinatorial Optimization with Interaction Costs (COPIC)
Input: Element sets E_1, E_2, feasible sets $\mathcal{F}_1 \subseteq 2^{E_1}, \mathcal{F}_2 \subseteq 2^{E_2}$
interaction costs $Q = (q_{i,j}) \in \mathbb{R}^{E_1 \times E_2}$,
linear costs $c \in \mathbb{R}^{E_1}, d \in \mathbb{R}^{E_2}$
Question: Find $S_1 \in \mathcal{F}_1, S_2 \in \mathcal{F}_2$ such that

$$c(S_1, S_2) := \sum_{i \in S_1} \sum_{j \in S_2} q_{i,j} + \sum_{i \in S_1} c_i + \sum_{j \in S_2} d_j$$

is minimized.
We denote such an instance of COPIC by $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$.

We study COPIC for feasible sets $\mathcal{F}$ which are unconstrained ($\mathcal{F} = 2^{[n]}$), bases of a matroid ($\mathcal{F} = \mathcal{B}(\mathcal{M})$), maximum matchings in graphs ($\mathcal{F} = \mathcal{PM}(G)$) and s-t-paths in graphs ($\mathcal{F} = \mathcal{P}_{s,t}(G)$). For the interaction costs we focus on the two special cases where Q is a matrix with fixed rank and Q is a diagonal matrix.

In Chapter 5 we study different types of matrix completion problems.

MATRIX COMPLETION
Input: Matrix class $\mathcal{F}$, partially filled matrix $A = (a_{i,j}) \in (\mathbb{K} \cup \{*\})^{m \times n}$
Question: Exist values $\tilde{a}_{i,j} \in \mathbb{K}$ for all i, j with $a_{i,j} = *$ such that the matrix $\tilde{A} = (\tilde{a}_{i,j}) \in \mathcal{F}$, where $\tilde{a}_{i,j} = a_{i,j}$, if $a_{i,j} \neq *$?

Our main focus is on the following matrix classes:

- Large matrices $\mathcal{F}_L$,
- Monge matrices $\mathcal{F}_M$,
- bottleneck Monge matrices $\mathcal{F}_{BM}$,
- matrices with monotonicity properties in their rows and columns $\mathcal{F}_{\geq,\geq}$,
- permuted variants of the classes mentioned above (denoted by $\mathcal{F}^{\pi,\sigma}$).

The aim of Chapter 6 is to obtain new results for recoverable robust discrete optimization. To that end, recoverable optimization problems are studied.

RECOVERABLE OPTIMIZATION
Input: Costs $c_1, c_2 \colon 2^E \to \mathbb{R}$, parameter $k \in \mathbb{N}$,
feasible solutions $\mathcal{F}_1, \mathcal{F}_2 \subseteq 2^E$
Question: Find an optimal solution for

$$\min_{X \in \mathcal{F}_1, Y \in \mathcal{F}_2} c_1(X) + c_2(Y),$$

such that

$$|X \cap Y| \geq k.$$

More specifically, in Section 6.1 we focus on the recoverable selection problem, where $\mathcal{F}_1 = \mathcal{F}_2 = \{S \subseteq E \colon |S| = q\}$ and in Section 6.2 we study the recoverable matroid basis problem, i.e. $\mathcal{F}_1, \mathcal{F}_2$ are the sets of bases of two matroids.

1.3 Connections and the Common Thread Among the Studied Problems

All of the problems studied in this thesis have in common that they generalize some aspect of classic problems in combinatorial optimization. These generalizations can be categorized into three types: the structure of feasible solutions, cost structures and robustness.

In this section we will categorize the problems tackled in this thesis according to their type, while giving some examples where these types of generalizations have been dealt with before in the literature.

1.3.1 Generalized Classes of Feasible Solutions

For many types of feasible solutions which are treated in combinatorial optimization there are various ways of generalizations. Several such generalizations play a role in this thesis.

Our first example are matroids which are a common generalization of spanning trees and linear independence, and polymatroids, which are a generalization of matroids. By now, the study of matroids and polymatroids has gained so much importance that they have become classic combinatorial objects on their own. They play a major role as sets of feasible solutions both in Chapter 4 and Chapter 6. Using polymatroids allows the selection of integer multiples (or if the submodular function is non-integral also real multiples) of elements instead of just 0-1 vectors of elements. Hence, in the integral case, the motivation is to look at multisets instead of sets of elements.

Furthermore, in Chapter 2 we study a common generalization of independent sets and matchings in graphs. In the model called *obnoxious facility location* or *δ-dispersion* on a graph, one is not only allowed to select vertices or edges, but also arbitrary points selected on edges of the graph.

Also, our work on matrix completion problems in Chapter 5 fits into the framework of generalized sets of feasible solutions. Our results in this regard can be seen as a generalization of some results from the graph completion literature [66]. More precisely, we allow general real or integer entries in the matrix, instead of just 0/1 entries, which are in one-to-one correspondence to edges in a graph via its adjacency matrix.

Finally, also the notion of Steiner cycles (see Chapter 3) can be seen as a generalization of the concept of a Hamiltonian cycle in a graph.

1.3.2 Generalized Cost Structures

A well studied generalization of the classic linear cost functions

$$c(S) := \sum_{e \in S} c_e$$

are quadratic cost functions. There, instead of a vector c we are given a cost matrix $Q = (q_{i,j}) \in \mathbb{R}^{E \times E}$ and we set

$$c(S) := \sum_{i,j \in S} q_{i,j}.$$

There is a vast literature about combinatorial optimization problems with quadratic cost functions of this type (see for instance [8, 9, 43, 75]).

In Chapter 4 we study a generalization of quadratic cost functions, namely *interaction costs*, which are defined for two solution sets S_1, S_2 over ground sets E_1, E_2 by a matrix $Q = (q_{i,j}) \in \mathbb{R}^{E_1 \times E_2}$ as

$$c(S_1, S_2) := \sum_{i \in S_1} \sum_{j \in S_2} q_{i,j}.$$

The study of this type of cost function for different sets of feasible solutions was motivated by existing work for the unconstrained case (*bipartite unconstrained quadratic programming problem* [44, 63, 77, 107]) and the assignment problem (*bilinear assignment problem* [40]).

1.3.3 Robust Optimization

Robust optimization can be seen as a generalization of the feasible sets or cost structures of any (combinatorial) optimization problems. Because of its huge impact in terms of practical applications and developed theory, we highlight the robust optimization models studied in this thesis separately.

Min-max and min-max regret robust optimization are already well studied, also from a theoretical point of view (see Kouvelis [89] and Kasperski and Zielinski [79]). One of their major downsides is that they model a very pessimistic decision maker. This is why robust optimization models that allow for less risk-averse decisions were introduced in the applied optimization literature. One of those models is the *recoverable robust* optimization model introduced by Liebchen et al. [94].

Recoverable Robust Optimization
Input: Preparation costs $C\colon 2^E \to \mathbb{R}$, uncertainty set $\mathcal{U}$,
feasible solutions $\mathcal{F} \subseteq 2^E$, recoverability parameter $k \in \mathbb{N}$
Question: Find an optimal solution for

$$\min_{X \in \mathcal{F}} \left(C(X) + \max_{c^s \in \mathcal{U}} \min_{Y \in \mathcal{F}\colon |Y \setminus X| \leq k} (c^s(S) \right).$$

The main idea of this model is giving the decision maker some limited power to intervene after the scenario (cost function) is revealed.

In Chapter 6 we obtain results for recoverable optimization problems (optimization problems with intersection constraints), to which the Recoverable Robust Optimization problem with interval uncertainties can be reduced to.

1.3.4 Influencing Factors on the Computational Complexity

Another common characteristic of the problems studied in this thesis is that they are computationally hard (NP-hard) in their general version. We want to obtain a better theoretical understanding of which properties of these problems lead to their hardness. To that end our goal is to identify conditions which turn hard problems into easier ones. In this manner we obtain a better understanding of the frontier between efficiently solvable cases and cases that remain hard. Imposing additional conditions on the problem structure gives rise to special cases. We mainly deal with the following two types of special cases:

- special cases which result by restricting the combinatorial structure,

- special cases which result by restricting to special cost structures.

One very common line of study for problems which are hard on general graphs is restricting the problem to special types of graph classes. The ISGCI [41] is an online encyclopaedia of graph classes summarizing a vast amount of results along these lines. Chapter 3 is an example of this kind of results where we show that the Steiner Cycle problem can be solved in linear time on interval graphs. Also, in Chapter 2 several results of this type are obtained for the δ-Dispersion problem.

Additionally, a very classic approach of this kind is the study of matroidal structures, which by definition are special cases of independence systems. We follow this approach both in Chapter 4 and Chapter 6.

Another way to restrict the combinatorial structure is restricting different parameters that define the set of feasible solutions. The special cases studied in Chapter 5 fall into this category. A natural restriction for problems where matrices show up is to restrict the feasible matrix entries to the set $[k]$, where k is a restricted parameter. Also, the main results in Chapter 2 are about the complexity of δ-Dispersion for different fixed values of δ.

Specializing the cost structure to obtain efficiently solvable special cases is another common approach in the combinatorial optimization literature (see [23] for examples). In Chapter 4 this is our main tool to gain a better understanding of the complexity of COPIC. Our focus there lies on restricting the interaction cost matrix to the special cases of matrices with bounded rank, and diagonal matrices.

1.4 Outlook on Main Results

In Chapter 2 we study the computational complexity of the δ-Dispersion problem. The main results obtained can be summarized as follows.

Theorem.

(a) If $\delta = 1/b$ for some integer b, then the δ-dispersion number of G can be determined in the following way: If G is a tree then $\delta\text{-Disp}(G) = b|E| + 1$, and if G is not a tree then $\delta\text{-Disp}(G) = b|E|$.

(b) If $\delta = 2/b$ for some integer b, then $\delta\text{-Disp}(G)$ can be computed in polynomial time.

(c) If $\delta = a/b$ for integers a and b with $a \geq 3$ and $\gcd(a, b) = 1$, then the computation of $\delta\text{-Disp}(G)$ is an NP-hard problem.

Point (b) of the theorem is based on proving a deep connection between 2-dispersion and the Edmonds-Gallai decomposition of G and reformulating the problem as a submodular optimization problem. In addition, we also obtain results for a generalization to integer edge lengths and special graph classes.

In Chapter 3 we study the Steiner Cycle and Steiner Path Cover problem on interval graphs. The main results of this chapter are summarized in the following theorem.

Theorem. *The* Steiner Path Cover *and* Steiner Cycle *problem on interval graphs given in endpoint sorted order can be solved in $O(n)$ time, where n is the number of intervals.*

Chapter 4 introduces COPIC. In the first part of the chapter we study interaction costs given by low rank matrices. For a formal definition of the parametric problem MPLCOP and parametric complexity, used in part (b) of the following theorem, we refer the reader to Section 4.3.2

Theorem.

(a) If $\mathrm{rk}(Q) = r$ *and there is a* $\mathrm{T}(\mathcal{F}_2)$*-time algorithm for* LCOP *instances* $(\mathcal{F}_2, f)$ *for every* $f \in \mathbb{R}^n$*, then the* COPIC *instance* $(2^{[m]}, \mathcal{F}_2, Q, c, d)$ *can be solved in* $O(\binom{m}{r} 2^r \max\{rm, rn, \mathrm{T}(\mathcal{F}_2)\})$ *time.*

(b) Let l_1, l_2 *be the parametric complexity of* MPLCOP *instances* $(\mathcal{F}_1', a, c)$ *and* $(\mathcal{F}_2', b, d)$ *respectively, and* $\mathrm{rk}(Q) = r$ *is a constant. If both* LCOP *instances* $(\mathcal{F}_1, h)$ *and* $(\mathcal{F}_2, h)$ *can be solved in polynomial time for arbitrary linear cost vectors* h*, then* COPIC *instances* $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ *can be solved in* $O(\mathrm{poly}(n, m, l_1^r, l_2^r))$ *time.*

For the special case of diagonal interaction cost matrices, studied in a second part of Chapter 4. Table 1.1 gives an overview of the obtained results.

$\mathcal{F}_1 \setminus \mathcal{F}_2$	$2^{[n]}$	$\mathcal{B}(\mathcal{U}_n^{k_2})$	$\mathcal{B}(\mathcal{M}_2)$	$\mathcal{PM}(G)$	$\mathcal{P}_{s_2,t_2}(G)$
$2^{[n]}$	$O(n)$	**P**	**P**	**P**	**P**
$\mathcal{B}(\mathcal{U}_n^{k_1})$		**P**	**P** $_{(c=d=0)}$	open	open
$\mathcal{B}(\mathcal{M}_1)$			**P** $_{(c=d=0)}$	open	NP-hard
$\mathcal{PM}(G)$				NP-hard	open
$\mathcal{P}_{s_1,t_1}(G)$					**Table 4.3**

Table 1.1: Summary of complexity results for COPIC with a diagonal matrix. Bold entries correspond to new results obtained in Section 4.4.

Moreover, we are also able to characterize when COPIC instances are linearizable in the final part of Chapter 4.

In Chapter 5 we obtain new results about the computational complexity of matrix completion problems. We focus mainly on matrix classes which are studied in connection with the special cases literature for combinatorial optimization problems, like Large matrices $\mathcal{F}_L$, Monge matrices $\mathcal{F}_M$, bottleneck Monge matrices $\mathcal{F}_{BM}$, matrices with monotonicity properties in their rows and columns $\mathcal{F}_{\geq,\geq}$, and permuted variants of the aforementioned classes (denoted by $\mathcal{F}^{\pi,\sigma}$). The following theorem summarizes the results from Chapter 5.

Theorem.

(a) Matrix Completion *is NP-complete for* $\mathcal{F}_M^{\pi,\sigma}, \mathcal{F}_{BM}^{\pi,\sigma}, \mathcal{F}_{\geq,\geq}^{\pi,\sigma}$.

(b) Matrix Completion *can be solved in polynomial time for* $\mathcal{F}_L^{\pi,\sigma}$.

(c) Matrix Completion *can be solved in polynomial time for* $\mathcal{F}_M^{\pi,\sigma}$, *restricted to* $\{0,1\}$-*matrices.*

In Chapter 6 we study the Recoverable Robust Optimization problem for different sets of feasible solutions. This leads to the study of combinatorial optimization problems with intersection constraints, for which we suggest several new algorithms. As a consequence the following new results about recoverable robustness are obtained.

Theorem.

(a) The Recoverable Selection *problem* ($\mathcal{F}_1 = \mathcal{F}_2$ *is the set of all sets of cardinality* q*) can be solved by a greedy algorithm (easy to implement) in* $O(n \log n)$ *time.*

(b) The Recoverable Selection *problem is solvable in* $O(n)$ *time.*

(c) The Recoverable Matroid Basis *problem* ($\mathcal{F}_1, \mathcal{F}_2$ *are sets of bases of two, possibly different, matroids) can be solved in strongly polynomial time.*

(d) The Recoverable Polymatroid Basis *problem (for a definition of the generalization to polymatroids see Section 6.2.4) can be solved in strongly polynomial time.*

From the point of view of matroid theory, the following generalization of the Recoverable Matroid Basis problem, which we denote by $(P_{=k})$ is of special interest:

$$\begin{aligned} \min\ & c_1(X) + c_2(Y) \\ \text{s.t. } & X \in \mathcal{B}_1 \\ & Y \in \mathcal{B}_2 \\ & |X \cap Y| = k. \end{aligned}$$

The following result is our main result in Section 6.2 and is obtained via a primal-dual algorithm.

Theorem. $(P_{=k})$ *can be solved using at most* $k \times |E|$ *primal or dual augmentations in strongly polynomial time.*

To finish the introductory part, we provide some information on the collaborations and papers that form a major backbone of this thesis.

Chapter 2: The results in this chapter are based on joint work with Alexander Grigoriev, Tim A. Hartmann and Gerhard J. Woeginger; see [68].

Chapter 3: The results in this chapter are based on joint work with Ante Ćustić; see [37].

Chapter 4: The results in this chapter are based on joint work with Ante Ćustić and Abraham Punnen; see [92].

Chapter 5: The results in this chapter are based on joint work with Vladimir Deineko, Eranda Dragoti-Çela, Bettina Klinz and Gerhard J. Woeginger. This is ongoing work; see [42].

Chapter 6: Parts of the investigations reported about in this chapter are ongoing work.

The results in Section 6.1 are based on joint work with Thomas Lachmann and Gerhard J. Woeginger; see [91]. A partial account of the work can be found in the extended abstract [90].

The results in Section 6.2 are based on joint work with Andras Frank, Britta Peis and Veerle Timmermans; see [52]. A partial account of the work can be found in the extended abstract [93].

2 Dispersing Obnoxious Facilities on a Graph

2.1 Introduction

A large part of the facility location literature deals with *desirable* facilities that people like to have nearby, such as service centers, police departments, fire stations, and warehouses. However, there also do exist facilities that are *undesirable* and *obnoxious*, such as nuclear reactors, garbage dumps, chemical plants, military installations, and high security penal institutions. A standard goal in location theory is to spread out such obnoxious facilities and to avoid their accumulation and concentration in a small region; see for instance Erkut & Neuman [47] and Cappanera [28] for comprehensive surveys on this topic.

In this chapter, we investigate the location of obnoxious facilities in a metric space whose topology is determined by a graph. Formally, let $G = (V, E)$ be an undirected connected graph, where every edge is rectifiable and has unit length. Let $P(G)$ denote the continuum set of points on all the edges in E together with all the vertices in V. For two points $p, q \in P(G)$, we denote by $d(p, q)$ the length of a shortest path connecting p and q in the graph. A subset $S \subseteq P(G)$ is said to be *δ-dispersed* for some positive real number δ, if any two points $p, q \in S$ with $p \neq q$ are at distance $d(p, q) \geq \delta$ from each other. Our goal is to compute for a given graph $G = (V, E)$ and a given positive real number δ a maximum cardinality subset $S \subseteq P(G)$ that is δ-dispersed. Such a set S is called an *optimal* δ-dispersed set, and $|S|$ is called the *δ-dispersion number* δ-Disp(G) of the graph G.

Known and related results.

Obnoxious facility location goes back to the seminal articles of Goldman & Dearing [64] from 1975 and Church & Garfinkel [32] from 1978. The area actually covers a wide variety of problem variants and models; some models specify a geometric setting, while other models use a graph-theoretic setting.

For example, Abravaya & Segal [1] consider a purely geometric variant of obnoxious facility location, where a maximum cardinality set of obnoxious facilities has to be placed in a rectangular region, such that their pairwise distance as well as the distance to a fixed set of demand sites is above a given threshold. As another example we mention the graph-theoretic model of Tamir [115], where every edge $e \in E$ of the underlying graph $G = (V, E)$ is rectifiable and has a given edge-dependent length $\ell(e)$. Tamir discusses the complexity and approximability of various optimization problems with various objective functions. One consequence of [115] is that if the graph G is a tree, then the value

δ-Disp(G) can be computed in polynomial time. Segal [112] locates a single obnoxious facility on a network under various objective functions, such as maximizing the smallest distance from the facility to the clients on the network or maximizing the total sum of the distances between facility and clients.

Megiddo & Tamir [98] consider the covering problem that is dual to the δ-dispersion packing problem: Given a graph $G = (V, E)$ with rectifiable unit-length edges, find a minimum cardinality subset $S \subseteq P(G)$ such that every point in $P(G)$ is at distance at most δ from one of the facilities in S. Among many other results [98] shows that this covering problem is NP-hard for $\delta = 2$.

Finally, we mention the work of Gawrychowski, Krasnopolsky, Mozes & Weimann [62] who study the problem variant where the points in the dispersed set S must be vertices of the graph G. They show that for a given tree G and a given integer k, one can compute in linear time the largest possible value δ for which there exists a δ-dispersed set S of size $|S| = k$.

Our results.

We provide a complete picture of the complexity of computing the δ-dispersion number for connected graphs $G = (V, E)$ and positive rational numbers δ.

- If $\delta = 1/b$ for some integer b, then the δ-dispersion number of G can be written down without really looking at the structure of the graph: If G is a tree then δ-Disp(G) $= b|E| + 1$, and if G is not a tree then δ-Disp(G) $= b|E|$.
- If $\delta = 2/b$ for some integer b, then δ-Disp(G) can be computed in polynomial time. The algorithm uses the Edmonds-Gallai decomposition of G and reformulates the problem as a submodular optimization problem.
- If $\delta = a/b$ for integers a and b with $a \geq 3$ and $\gcd(a, b) = 1$, then the computation of δ-Disp(G) is an NP-hard problem.

The rest of the chapter is organized as follows. Section 2.2 summarizes the basic notations and states several technical observations. Section 2.3 presents the NP-hardness results. The reductions are essentially based on routine methods, but need to resolve certain number-theoretic issues. Our technical main contribution is the polynomial time algorithm for the case $\delta = 2$ as developed in Section 2.4; this result is heavily based on tools from matching theory. Section 2.5 summarizes the polynomially solvable special cases and provides additional structural insights.

2.2 Notation and Technical Preliminaries

All graphs in this chapter are undirected and connected, and all edges have unit length. Throughout the chapter we use the word *vertex* in the graph-theoretic sense, and we use the word *point* to denote the elements of the geometric structure $P(G)$. For a graph $G = (V, E)$ and a subset $V' \subseteq V$, we denote by $G[V']$ the subgraph induced by V'. For

an integer $c \geq 1$, the *c-subdivision* of G is the graph that results from G by subdividing every edge in E by $c-1$ new vertices into c new edges.

For an edge $e = \{u, v\}$ and a real number λ with $0 \leq \lambda \leq 1$, we denote by $p(u, v, \lambda)$ the point on e that has distance λ from vertex u. Note that $p(u, v, 0) = u$ and $p(u, v, 1) = v$, and note that point $p(u, v, \lambda)$ coincides with point $p(v, u, 1-\lambda)$; hence we will sometimes assume without loss of generality that $\lambda \leq 1/2$.

Lemma 2.1. *Let G be a graph, let $c \geq 1$ be an integer, and let G' be the c-subdivision of G. Then for every $\delta > 0$, the δ-dispersed sets in G are in one-to-one correspondence with the $(c \cdot \delta)$-dispersed sets in G'. In particular, δ-Disp$(G) = (c \cdot \delta)$-Disp(G').*

Proof. Every point $p(u, v, \lambda)$ in $P(G)$ translates into a corresponding point in $P(G')$ that lies on the subdivided edge between u and v and is at distance $c \cdot \lambda$ from vertex u. □

Lemma 2.1 has many useful consequences, as for instance the following:

Lemma 2.2. *Let $\delta > 0$ and let $c \geq 1$ be an integer.*

- *If the problem of computing the δ-dispersion number is NP-hard, then also the problem of computing the $(c \cdot \delta)$-dispersion number is NP-hard.*
- *If the problem of computing the $(c \cdot \delta)$-dispersion number is polynomially solvable, then also the problem of computing the δ-dispersion number is polynomially solvable.*

Proof. By Lemma 2.1 the c-subdivision of a graph yields a polynomial time reduction from computing δ-dispersions to computing $(c \cdot \delta)$-dispersions. □

For integers ℓ and k, the rational number ℓ/k is called *k-simple*. A set $S \subseteq P(G)$ is k-simple, if for every point $p(u, v, \lambda)$ in S the number λ is k-simple.

Lemma 2.3. *Let $\delta = a/b$ with integers a and b, and let $G = (V, E)$ be a graph. Then there exists an optimal δ-dispersed set S^* that is $2b$-simple.*

Proof. We first handle the cases with $b = 1$, so that δ is integer. Consider an optimal δ-dispersed set S for graph G. Note that for every vertex u, at most one point $p(u, v, \lambda)$ with $v \in V$ and $0 \leq \lambda < 1/2$ is in S. For every point $p = p(u, v, \lambda)$ with $0 \leq \lambda \leq 1/2$ in S, we put a corresponding point p^* into set S^*: If $0 \leq \lambda < 1/2$ then $p^* = p(u, v, 0)$, and if $\lambda = 1/2$ then $p^* = p(u, v, 1/2)$. As all points in the resulting set S^* are either vertices or midpoints of edges, we get that S^* is 2-simple. We claim that S^* is still δ-dispersed: Consider two distinct points p^* and q^* in S^*. Note that $d(p, p^*) < 1/2$ and $d(q, q^*) < 1/2$ by construction.

- If p^* and q^* both are vertices in V, then the distance $d(p^*, q^*)$ is integer. By the triangle inequality $d(p, q) \leq d(p, p^*) + d(p^*, q^*) + d(q^*, q)$. As the left hand side in this inequality is at least the integer δ and as its right hand side is strictly smaller than the integer $d(p^*, q^*) + 1$, we conclude $d(p^*, q^*) \geq \delta$.

- If p^* and q^* both are midpoints of edges, then $p = p^*$ and $q = q^*$ yields $d(p^*, q^*) \geq \delta$.
- If p^* is a vertex and q^* is the midpoint of some edge, then $d(p^*, q^*) = D + 1/2$ for some integer D. The triangle inequality together with $p = p^*$ yields that $\delta \leq d(p, q) = d(p^*, q) \leq d(p^*, q^*) + d(q^*, q) < D + 1$. This implies $D \geq \delta$, so that $d(p^*, q^*) \geq \delta + 1/2$.

Since S and S^* have the same cardinality, we conclude that S^* is an optimal δ-dispersed set that is 2-simple, exactly as desired.

In the cases where $\delta = a/b$ for some integer $b \geq 2$, we consider the b-subdivision G' of G. By the above discussion, G' possesses an optimal a-dispersed set S' that is 2-simple. Then Lemma 2.1 translates S' into an optimal δ-dispersed set S for G that is $2b$-simple. □

2.3 NP-Completeness Results

In this section we present our NP-hardness proofs for computing the δ-dispersion number. All proofs are done through polynomial time reductions from the following NP-hard variant of the independent set problem; see Garey & Johnson [61].

> Problem: Independent Set in Cubic Graphs (Cubic-Ind-Set)
>
> Instance: An undirected, connected graph $H = (V_H, E_H)$ in which every vertex is adjacent to exactly three other vertices; an integer bound k.
>
> Question: Does H contain an independent set I with $|I| \geq k$ vertices?

Throughout this section we consider a fixed rational number $\delta = a/b$, where a and b are positive integers that satisfy $\gcd(a, b) = 1$ and $a \geq 3$. Section 2.3.1 the cases with odd numerators $a \geq 3$, and Section 2.3.2 the cases with even numerators $a \geq 4$. It is instructive to verify that our arguments do not work for the cases with $a = 1$ and $a = 2$, as our gadgets and our arguments break down at various places.

2.3.1 NP-Hard Cases with Odd Numerator

Throughout this section we consider a fixed rational number $\delta = a/b$ where $\gcd(a, b) = 1$ and where $a \geq 3$ is an odd integer. For the NP-hardness proof, we first determine four positive integers x_1, y_1, x_2, y_2 that satisfy the following equations (2.1) and (2.2).

$$2b \cdot x_1 - 2a \cdot y_1 = a - 1 \tag{2.1}$$

$$b \cdot x_2 - a \cdot y_2 = 1 \tag{2.2}$$

Note that the value $a - 1$ on the right hand side of equation (2.1) is even, and hence is divisible by the greatest common divisor $\gcd(2b, 2a) = 2$ of the coefficients in the left hand side. With this, Bézout's lemma yields the existence of positive integers x_1 and y_1 that satisfy (2.1). Bézout's lemma also yields the existence of positive integers x_2 and y_2 in equation (2.2), as the coefficients in the left hand are relatively prime.

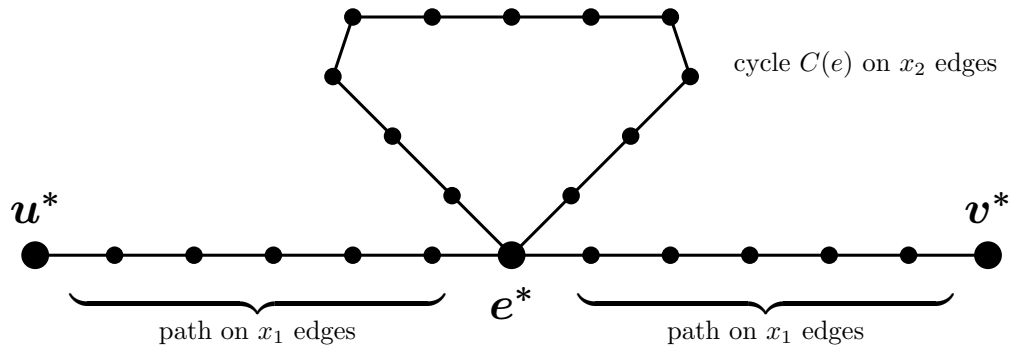


Figure 2.1: The edge $e = \{u, v\}$ in the instance of Cubic-Ind-Set translates into three vertices u^*, e^*, v^* in the dispersion instance, together with two paths and one cycle.

Our reduction now starts from an arbitrary instance $H = (V_H, E_H)$ and k of Cubic-Ind-Set, and constructs a corresponding dispersion instance $G = (V_G, E_G)$ from it.

- For every vertex $v \in V_H$, we create a corresponding vertex v^* in V_G.
- For every edge $e = \{u, v\} \in E_H$, we create a corresponding vertex e^* in V_G.
- For every edge $e = \{u, v\} \in E_H$, we create (i) a path with x_1 edges that connects vertex u^* to vertex e^*, (ii) another path with x_1 edges that connects v^* to e^*, and (iii) a cycle $C(e)$ with x_2 edges that runs through vertex e^*.

This completes the description of the graph $G = (V_G, E_G)$; see Figure 2.1 for an illustration. We claim that graph H contains an independent set of size k, if and only if (a/b)-Disp$(G) \geq k + (2y_1 + y_2)|E_H|$.

Lemma 2.4. *If graph H contains an independent set of size k, then the (a/b)-dispersion number of graph G is at least $k + (2y_1 + y_2)|E_H|$.*

Proof. Let I be an independent set of size k in graph $H = (V_H, E_H)$. We construct from I a δ-dispersed set $S \subseteq P(G)$ as follows. Let $u \in V_H$ be a vertex, and let e_1, e_2, e_3 be the three edges in E_H that are incident to u.

- If $u \in I$, then we put point u^* into S. On each of the three paths that connect vertex u^* respectively to vertex e_i^* $(i = 1, 2, 3)$, we select y_1 further points for S. The first selected point is at distance δ from u^*, and every further selected point is at distance $\delta = a/b$ from the preceding selected point. By equation (2.1), on each of the three paths the distance from the final selected point to point e_i^* $(i = 1, 2, 3)$ then equals $(a - 1)/(2b)$.

- If $u \notin I$, then on each of the three paths between u^* and e_i^* $(i = 1, 2, 3)$ we select y_1 points for S. The first selected point is at distance $\delta/2 = a/(2b)$ from u^*, and every further selected point is at distance δ from the preceding selected point. By equation (2.1), the distance from the final selected point to point e^* then equals $(2a - 1)/(2b)$.

Furthermore, for every edge $e \in E_H$ we select y_2 points from the cycle $C(e)$ for S:

- We start in point e^* and traverse $C(e)$ in clockwise direction. The first selected point is at distance $(a + 1)/(2b)$ from point e^*, and every further selected point is at distance δ from the preceding selected point. By equation (2.2), the distance from the final selected point to point e^* then equals $(a + 1)/(2b)$.

This completes the construction of set S. Now let us count the points in S. First, there are the k points $u^* \in S$ for which $u \in I$. Furthermore, for every edge $e = \{u, v\} \in E_H$ there are $2y_1$ points in S that lie on the two paths from u^* to e^* and from e^* to v^*. Finally, for every edge $e \in E_H$ there are y_2 points that lie on the cycle $C(e)$. Altogether, this yields the desired size $k + (2y_1 + y_2)|E_H|$ for S.

It remains to verify that the point set S is δ-dispersed. By construction, the points selected from each path are at distance at least δ from each other, and the same holds for the points selected from each cycle. If vertex u^* is in S, then all selected points on the three incident paths are at distance at least δ from u^*. If vertex u^* is not in S, then the first selected point on every path is at distance $\delta/2$ from u^*, so that these points are pairwise at distance at least δ from each other. Hence the only potential trouble could arise in the neighborhood of point e^*, where paths and cycles are glued together. Every selected point on $C(e)$ is at distance at least $(a + 1)/(2b)$ from point e^*. Every selected point on some path from u^* to e^* is at distance at least $(a - 1)/(2b)$ from e^* if $u \in I$ and is at distance at least $(2a - 1)/(2b)$ if $u \notin I$. Since for any edge $e = \{u, v\} \in E_H$ at most one of the end vertices u and v is in I, at most one selected point can be at distance $(a - 1)/(2b)$ from e^*, and all other points are at distance at least $(a + 1)/(2b)$ from e^*. Hence S is indeed δ-dispersed. □

Lemma 2.5. *If the (a/b)-dispersion number of graph G is at least $k + (2y_1 + y_2)|E_H|$, then graph H contains an independent set of size k.*

Proof. Let S be an (a/b)-dispersed set of size $k + (2y_1 + y_2)|E_H|$. By Lemma 2.3 we assume that for every point $p(u, v, \lambda)$ in S, the denominator of the rational number λ is $2b$.

For an edge $e = \{u, v\} \in E_H$, let us consider its corresponding path π on x_1 edges that connects vertex u^* to vertex e^*. Suppose that there is some point p in $S \cap \pi$ with $d(p, e^*) \leq (a - 2)/(2b)$. Then by Equation (2.2), set S will contain at most $y_2 - 1$ points from the cycle $C(e)$. In this case we restructure S as follows: We remove point p together with the at most $y_2 - 1$ points on cycle $C(e)$ from S, and instead insert y_2 points into S that are δ-dispersed on $C(e)$ and that all are at distance at least $(a + 1)/(2b)$ from e^*. As this restructuring does not decrease the size of S, we will from now on assume without loss of generality that $d(p, e^*) \geq (a - 1)/(2b)$ holds for every point $p \in S \cap \pi$.

Now let us take a closer look at the points in $S \cap \pi$. Equation (2.1) can be rewritten into $x_1 = y_1\delta + (a-1)/(2b)$, which yields $|S \cap \pi| \leq y_1 + 1$.

- In the equality case $|S \cap \pi| = y_1 + 1$, we must have $u^* \in S$ and also the point on π at distance $(a-1)/(2b)$ from e^* must be in S.
- In case $|S \cap \pi| \leq y_1$, there is ample space for picking y_1 points from π that are δ-dispersed and that are at distance at least $\delta/2$ from u^* and at distance at least $\delta/2$ from e^*. Hence we will from now on assume $|S \cap \pi| = y_1$ in these cases.

Now let us count: Set S contains exactly y_1 interior points from every path π, and altogether there are $2|E_H|$ such paths. Set S contains exactly y_2 points from every cycle $C(e)$, and altogether there are $|E_H|$ such cycles. Since $|S| \geq k + (2y_1 + y_2)|E_H|$, this means that S must contain at least k further points on vertices u^* with $u \in V_H$. The corresponding subset of V_H is called I.

Finally, we claim that this set I with $|I| \geq k$ forms an independent set in graph H. Suppose for the sake of contradiction that there is an edge $e = \{u, v\} \in E_H$ with $u^* \in I$ and $v^* \in I$. Consider the two paths that connect u^* to e^* and v^* to E^*. By the above discussion, S then contains two points at distance $(a-1)/(2b)$ from e^*. As these two points are then at distance at most $(a-1)/b < \delta$ from each other, we arrive at the desired contradiction. □

The statements in Lemma 2.4 and in 2.5 yield the following theorem.

Theorem 2.6. *Let a and b be positive integers with $\gcd(a, b) = 1$ and odd $a \geq 3$. Then it is NP-hard to compute the (a/b)-dispersion number of a graph G.*

2.3.2 NP-Hard Cases With Even Numerator

In this section we consider a fixed rational number $\delta = a/b$ where $\gcd(a, b) = 1$ and where $a \geq 4$ is an even integer. The NP-hardness argument is essentially a minor variation of the argument in Section 2.3.1 for the cases with odd numerators. Therefore, we will only explain the modifications, and leave all further details to the reader.

The NP-hardness proof in Section 2.3.1 is centered around the four positive integers x_1, y_1, x_2, y_2 introduced in equations (2.1) and (2.2). We perform the same reduction from Cubic-Ind-Set as in Section 2.3.1 but with positive integers x_1, y_1, x_2, y_2 that satisfy the following equations (2.3) and (2.4).

$$2b \cdot x_1 - 2a \cdot y_1 = a - 2 \tag{2.3}$$

$$b \cdot x_2 - a \cdot y_2 = 2 \tag{2.4}$$

In (2.3), the right hand side $a - 2$ is even and divisible by the greatest common divisor of the coefficients in the left hand side. In (2.4), the coefficients in the left hand are relatively prime. Therefore Bézout's lemma can be applied to both equations.

The graph $G = (V_G, E_G)$ is defined as before, with a vertex v^* for every $v \in V_H$ and a vertex e^* for every $e \in E_H$, with paths on x_1 edges and cycles $C(e)$ on x_2 edges. The arguments in Lemma 2.4 and 2.5 can easily be adapted and yield the following theorem.

Theorem 2.7. *Let a and b be positive integers with $\gcd(a,b) = 1$ and even $a \geq 4$. Then it is NP-hard to compute the (a/b)-dispersion number of a graph G.*

2.3.3 Containment in NP

In this section we consider the decision version of δ-dispersion: *"For a given graph $G = (V, E)$, a positive real δ, and a bound k, decide whether δ-Disp$(G) \leq k$."* Our NP-certificate specifies the following partial information on a δ-dispersed set S in a graph $G = (V, E)$:

- The certificate specifies the set $W := V \cap S^*$.
- For every edge $e \in E$, the certificate specifies the number n_e of facilities that are located in the interior of e.

As every edge accommodates at most $1/\delta$ points from S, the encoding length of our certificate is polynomially bounded in the instance size. For verifying the certificate, we introduce for every vertex u and for every incident edge $e = \{u, v\} \in E$ with $n_e > 0$ a corresponding real variable $x(u, e)$, which models the distance between vertex u and the closest point from S in the interior of edge e. Finally, we introduce the following linear constraints:

- The non-negativity constraints $x(u, e) \geq 0$.
- For every edge $e = \{u, v\} \in E$, the inequality
$$x(u, e) + (n_e - 1)\delta + x(v, e) \leq 1.$$
- For all $u, v \in W$ with $u \neq v$, the inequality $d(u, v) \geq \delta$.
- For all $w \in W$ and $e = \{u, v\} \in E$, the inequality $x(u, e) + d(u, w) \geq \delta$.
- For all $e = \{u, v\} \in E$ and $e' = \{u', v'\} \in E$, the inequality
$$x(u, e) + d(u, u') + x(u', e') \geq \delta.$$

These inequalities enforce that on every edge the variables properly work together, and that the underlying point set indeed is δ-dispersed. For verifying the certificate, we simply check in polynomial time whether the resulting linear program has a feasible solution, and whether $|W| + \sum_{e \in E} n_e \geq k$ holds.

Theorem 2.8. *The decision version of δ-dispersion lies in NP, even if the value δ is given as part of the input.*

2.4 The Polynomial Time Result for $\delta = 2$

This section derives a polynomial time algorithm for computing the 2-dispersion number of a graph. This algorithm is heavily based on tools from matching theory, as for instance developed in the book by Lovász & Plummer [95]. As usual, the size of a maximum cardinality matching in graph G is denoted by $\nu(G)$.

Lemma 2.9. *Every graph $G = (V, E)$ satisfies $2\text{-Disp}(G) \geq \nu(G)$.*

Proof. The midpoints of the edges in every matching form a 2-dispersed set. □

A 2-dispersed set is in *canonical* form, if it entirely consists of vertices and of midpoints of edges. Recall that by Lemma 2.3 every graph $G = (V, E)$ possesses an optimal 2-dispersed set in canonical form. Throughout this section, we will consider 2-dispersed (but not necessarily optimal) sets S^* in canonical form; we always let V^* denote the set of vertices in S^*, and we let E^* denote the set of edges whose midpoints are in S^*. Finally, $N^* \subseteq V$ denotes the set of vertices in $V - V^*$ that have a neighbor in V^*. As S^* is 2-dispersed, the vertex set V^* forms an independent set in G, and the edge set E^* forms a matching in G. Furthermore, the vertex set N^* separates the vertices in V^* from the edges in E^*; in particular, no edge in E^* covers any vertex in N^*. We start with two technical lemmas that will be useful in later arguments.

Lemma 2.10. *Let $G = (V, E)$ be a graph with a perfect matching, and let S^* be some 2-dispersed set in canonical form in G. Then $|S^*| \leq \nu(G)$.*

Proof. Let $M \subseteq E$ denote a perfect matching in G, and for every vertex $v \in V$ let $e(v)$ denote its incident edge in matching M. Consider the vertex set V^* and the edge set E^* that correspond to set S^*. Then E^* together with the edges $e(v)$ with $v \in V^*$ forms another matching M' of cardinality $|E^*| + |V^*| = |S^*|$ in G. Now $|S^*| = |M'| \leq \nu(G)$ yields the desired inequality. □

A graph G is *factor-critical* [95], if for every vertex $x \in V$ there exists a matching that covers all vertices except x. A *near-perfect* matching in a graph covers all vertices in V except one. Note that the statement in the following lemma cannot be extended to graphs that consist of a single vertex.

Lemma 2.11. *Every 2-dispersed set S^* in a graph $G = (V, E)$ with $|V| \geq 3$ that is factor-critical satisfies $|S^*| \leq \nu(G)$.*

Proof. Without loss of generality we assume that S^* is in canonical form, and we let V^* and E^* denote the underlying vertex set and edge set, respectively. If V^* is empty, we have $|S^*| = |E^*| \leq \nu(G)$ since E^* is a matching. If V^* is non-empty, then also N^* is non-empty (here we use the condition $|V| \geq 3$) and we pick some vertex $x \in N^*$. We consider a near-perfect matching M that covers all vertices except x, and we let $e(v)$ denote the edge incident to $v \in V$ in matching M. Then E^* together with the edges $e(v)$ with $v \in V^*$ forms another matching M' of cardinality $|E^*| + |V^*| = |S^*|$ in G. The claim follows from $|S^*| = |M'| \leq \nu(G)$. □

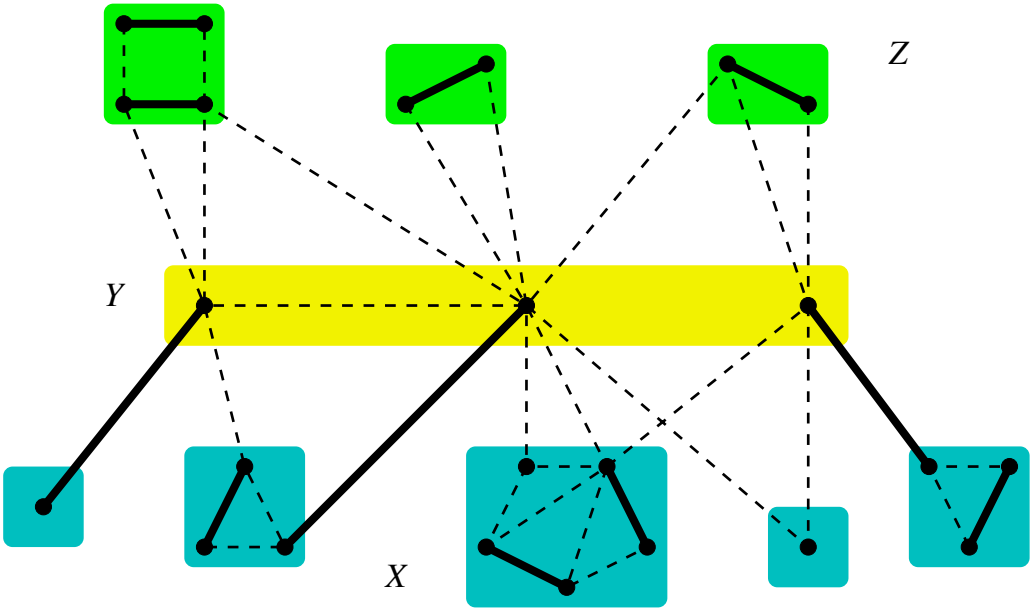


Figure 2.2: An illustration for the Edmonds-Gallai structure theorem. A maximum matching is shown with fat edges, and the non-matching edges are dashed.

The following theorem goes back to Edmonds [45] and Gallai [58, 59]; see also Lovász & Plummer [95]. Figure 2.2 gives an illustration.

Theorem 2.12. *(Edmonds-Gallai structure theorem) Let $G = (V, E)$ be a graph. The following decomposition of V into three sets X, Y, Z can be computed in polynomial time.*

$$\begin{aligned} X &= \{v \in V \mid \text{there exists a maximum matching that misses } v\} \\ Y &= \{v \in V \mid v \notin X \text{ and } v \text{ is adjacent to some vertex in } X\} \\ Z &= V - (X \cup Y) \end{aligned}$$

The Edmonds-Gallai decomposition has the following properties:

- *Set X is the union of the odd-sized components of $G - Y$; every such odd-sized component is factor-critical. Set Z is the union of the even-sized components of $G - Y$.*
- *Every maximum matching in G induces a perfect matching on every (even-sized) component of Z and a near-perfect matching on every (odd-sized) component of X. Furthermore, the matching matches the vertices in Y to vertices that belong to $|Y|$ different components of X.*

We further subdivide the set X in the Edmonds-Gallai decomposition into two parts: Set X_1 contains the vertices of X that belong to components of size 1, and set $X_{\geq 3}$ contains the vertices that belong to (odd-sized) components of size at least 3. The

vicinity $\mathrm{vic}(v)$ of a vertex $v \in V$ consists of vertex v itself and of the midpoints of all edges incident to v.

Lemma 2.13. *There exists an optimal 2-dispersed set S^* in canonical form (with underlying edge set E^*) that additionally satisfies the following three properties.*

- *P1. In every component of $X_{\geq 3}$, the set E^* induces a near-perfect matching.*
- *P2. For every vertex $y \in Y$, the set $vic(y) \cap S^*$ is either empty or consists of the midpoint of some edge between X and Y.*
- *P3. In every component of Z, the set E^* induces a perfect matching.*

Proof. We start from an arbitrary optimal 2-dispersed set S^* (in canonical form, with corresponding sets V^* and E^*) and transform it in two steps into an optimal 2-dispersed set of the desired form.

In the first transformation step, we exploit a matching M between sets Y and X that matches every vertex $y \in Y$ to some vertex $M(y)$, so that for $y_1 \neq y_2$ the vertices $M(y_1)$ and $M(y_2)$ belong to different components of X; see Theorem 2.12. A vertex $y \in Y$ is called *blocked*, if it is adjacent to some $x \in X_1 \cap S^*$. As for a blocked vertex the set $\mathrm{vic}(y) \cap S^*$ is already empty (and hence already satisfies property P2), we will not touch it at the moment. We transform S^* in the following way.

- For every non-blocked vertex $y \in Y$, the set $\mathrm{vic}(y) \cap S^*$ contains at most one point. We remove this point from S^*, and we insert instead the midpoint of the edge between y and $M(y)$ into S^*. These operations cannot decrease the size of S^*.
- Every (odd-sized) component C of $X_{\geq 3}$ contains at most one point $M(y)$ with $y \in Y$. We compute a near-perfect matching M_C for C that misses this vertex $M(y)$ (and if no such vertex is in C, matching M_C misses an arbitrary vertex of C). We remove all points in C from S^*, and we insert instead the midpoints of the edges in M_C. As by Lemma 2.11 we remove at most $\nu(C)$ points and as we insert exactly $\nu(C)$ points, these operations will not decrease the size of S^*.

The resulting set S^* is of course again in canonical form, and it is also easy to see that S^* is still 2-dispersed. Furthermore, S^* now satisfies properties P1 and P2.

In the second transformation step, we note that the current S^* does neither contain vertices from Y nor midpoints of edges between Y and Z. For every (even-sized) component C of Z, we compute a perfect matching M_C. We remove all points in C from S^*, and we insert instead the midpoints of the edges in M_C. As by Lemma 2.11 we remove at most $\nu(C)$ points and as we insert exactly $\nu(C)$ points, these operations will not decrease the size of S^*. The resulting set S^* is 2-dispersed and satisfies properties P1, P2, and P3. □

The optimal 2-dispersed sets in Lemma 2.13 are strongly structured and fairly easy to understand: The perfect matchings in set Z contribute exactly $|Z|/2$ points to S^*. Every (odd-sized) component C in $X_{\geq 3}$ contributes exactly $(|C|-1)/2$ points to S^*. The

only remaining open decisions concern the points in X_1 and the midpoints of the edges $\{y, M(y)\}$ for $y \in Y$. So let us consider the set $T := S^* \cap X_1$, and let $\Gamma(T) \subseteq Y$ denote the vertices in Y that are adjacent to some vertex in T. Then every vertex y in $Y - \Gamma(T)$ contributes the midpoint of $\{y, M(y)\}$ to S^*, and every vertex $x \in T$ contributes itself to S^*.

Hence the remaining optimization problem boils down to finding a subset $T \subseteq X_1$ that maximizes the function value $f(T) := |Y - \Gamma(T)| + |T|$, which is equivalent to minimizing the function value

$$g(T) := |\Gamma(T)| - |T|. \tag{2.5}$$

The set function $g(T)$ in (2.5) is a *submodular* function, as it satisfies

$$g(A) + g(B) \geq g(A \cup B) + g(A \cap B)$$

for all $A, B \subseteq X_1$; see for instance Grötschel, Lovász & Schrijver [69]. Therefore, the minimum value of $g(T)$ can be determined in polynomial time by the ellipsoid method [69], or by Cunningham's combinatorial algorithm [35].

We also describe another way of minimizing the function $g(T)$ in polynomial time, that avoids the heavy machinery of submodular optimization and that formulates the problem as a minimum s-t-cut computation in a weighted directed auxiliary graph. The auxiliary graph is defined as follows.

- Its vertex set contains a source s and a sink t, together with all the vertices in X_1 and all the vertices in Y.
- For every $x \in X_1$, there is an arc (s, x) of weight $w(s, x) = 1$ from the source to x. For every $y \in Y$, there is an arc (y, t) of weight $w(y, t) = 1$ from y to the sink. Whenever the vertices $x \in X_1$ and $y \in Y$ are adjacent in the original graph G, the auxiliary graph contains the arc (x, y) of weight $w(x, y) = +\infty$.

Now let us consider some s-t-cut of finite weight, which is induced by some vertex set U in the auxiliary graph with $s \in U$ and $t \notin U$. As all arcs from set X_1 to set Y have infinite weights, whenever U contains some vertex $x \in X_1$ then U must also contain all the neighbors of x in Y. By setting $T := X_1 \cap U$, we get that the value of the cut equals $|X_1 - T| + |\Gamma(T)|$; hence the minimizer for (2.5) can be read off the minimizing cut in the auxiliary graph.

We finally summarize all our insights and formulate the main result of this section.

Theorem 2.14. *The 2-dispersion number of a graph G can be computed in polynomial time.*

2.5 The Polynomially Solvable Cases

Theorem 2.14 and Lemma 2.2 together imply that for every rational number $\delta = a/b$ with numerator $a \leq 2$, the δ-dispersion number of a graph can be computed in polynomial time. We now present some results that provide additional structural insights into these

cases. The cases where the numerator is $a = 1$ are structurally trivial, and the value of the corresponding δ-dispersion number can be written down with the sole knowledge of $|V|$ and $|E|$.

Lemma 2.15. *Let $\delta = 1/b$ for some integer b, and let $G = (V, E)$ be a connected graph.*

- *If G is a tree then δ-Disp$(G) = b|E| + 1$.*
- *If G is not a tree then δ-Disp$(G) = b|E|$.*

Proof. If G is a tree, we use a δ-dispersed set S that contains all vertices in V and that for every edge $e = \{u, v\}$ contains all points $p(u, v, i/b)$ with $i = 1, \ldots, b-1$. Clearly $|S| = b|E| + 1$. If G is not a tree, set S contains for every edge $e = \{u, v\}$ all the points $p(u, v, (2i-1)/(2b))$ with $i = 1, \ldots, b$. Clearly $|S| = b|E|$.

It remains to show that there are no δ-dispersed sets of larger cardinality. If G is a tree, we root it at an arbitrary vertex so that it becomes an out-tree. We partition $P(G)$ into $|E| + 1$ regions: One region consists of the root, and all other regions consist of the interior points on some edge together with the source vertex of that edge. A δ-dispersed set contains at most b points from every edge-region and at most one point from the root region. If G is not a tree, we similarly partition $P(G)$ into $|E|$ regions: Every region either consists of the interior points of some edge, or of the interior points of an edge together with one of its incident vertices. A δ-dispersed set contains at most b points from every such region. □

The following lemma derives an explicit (and very simple) connection between the 2-dispersion number and the $(2/b)$-dispersion number (with odd denominator b) of a graph. The lemma also implies directly that for every odd b, the computation of $(2/b)$-dispersion numbers is polynomial time equivalent to the computation of 2-dispersion numbers.

Lemma 2.16. *Let $G = (V, E)$ be a graph, let $z \geq 1$ be an integer, and let $\delta = 2/(2z+1)$. Then the dispersion numbers satisfy δ-Disp$(G) = 2$-Disp$(G) + z|E|$.*

Proof. We first show that δ-Disp$(G) \geq 2$-Disp$(G) + z|E|$. Indeed, let S_2 denote an optimal 2-dispersed set for G. By Lemma 2.3 we assume that S_2 is in canonical form and hence entirely consists of vertices and of midpoints of edges. We partition the edge set E into three parts: Part E_1 contains the edges, for which one end vertex is in S_2. Part $E_{1/2}$ contains the edges whose midpoint lies in S_2. Part E_0 contains the remaining edges (which hence are disjoint from S_2). We construct a point set $S_\delta \subseteq P(G)$ as follows:

- For every edge $\{u, v\} \in E_1$ with $u \in S_2$, we put point u together with the z points $p(u, v, i\delta)$ with $i = 1, \ldots, z$ into S_δ.
- For every edge $\{u, v\} \in E_{1/2}$, we put the $z + 1$ points $p(u, v, (4i-3)\delta/4)$ with $i = 1, \ldots, z+1$ into S_δ.

- For every $\{u, v\} \in E_0$, we put the z points $p(u, v, (4i-1)\delta/4)$ with $i = 1, \ldots, z$ into S_δ.

It is easily verified that the resulting set S_δ is δ-dispersed and contains $|S_2|+z|E|$ points.

Next, we show that δ-Disp$(G) \leq 2$-Disp$(G) + z|E|$. Let S_δ denote an optimal δ-dispersed set for G. By Lemma 2.3 we assume that for every point $p(u, v, \lambda)$ in S_δ, the denominator of the rational number λ is $2(2z+1)$. Our first goal is to bring the points in S_δ into a particularly simple constellation.

- As long as there exist edges $e = \{u, v\} \in E$ with $u, v \in S_\delta$, we remove all points on e from S_δ and replace them by the $z+1$ points $p(u, v, (4i-3)\delta/4)$ with $i = 1, \ldots, z+1$.

- Next, for every edge $e = \{u, v\} \in E$ with $u \in S_\delta$ and $v \notin S_\delta$, we remove all points on e from S_δ and replace them by the $z+1$ points $p(u, v, i\delta)$ with $i = 1, \ldots, z$.

- Finally, for every edge $e = \{u, v\} \in E$ with $u, v \notin S_\delta$ we remove all points on e from S_δ and replace them by the z points $p(u, v, (4i-1)\delta/4)$ with $i = 1, \ldots, z$.

It can be seen that these transformations do not decrease the cardinality of S_δ, and that the resulting set is still δ-dispersed. Finally, we construct the following set S_2 from S_δ: First, S_2 contains all points in $V \cap S_\delta$, Secondly, whenever S_δ contains $z+1$ points from the interior of some edge $e \in E$, then we put the midpoint of e into S_2. It can be shown that the resulting set S_2 is 2-dispersed and has the desired cardinality. □

2.6 Integer Edge Lengths

In this section we show that the results of Lemma 2.15 and Lemma 2.16 can be generalized to graphs with integer edge lengths l_e for $e \in E$. The main observation is that an instance of δ-dispersion in a graph $G = (V, E)$ with edge lengths l is equivalent to an instance of δ-dispersion in the graph sub(G, l) with unit-length edges, where sub(G, l) is obtained from G by subdividing each edge e of G into a path with exactly l_e edges. This directly proves that if l_e is polynomially bounded our problem can be solved in polynomial time for δ with numerator 1 and 2. We will now show this holds also with no restriction on l_e, except for being integer.

Lemma 2.17. *Let $\delta = 1/b$ for some integer b, and let $G = (V, E)$ be a connected graph with edge lengths l_e.*

- *If G is a tree then δ-Disp$((G, l)) = b\sum_{e \in E} l_e + 1$.*

- *If G is not a tree then δ-Disp$((G, l)) = b\sum_{e \in E} l_e$.*

Proof. This result follows directly from the observation about sub(G, l) and Lemma 2.15, since subdivision of edges has no influence on the property whether G is a tree. □

To obtain a polynomial time result for $\delta = \frac{2}{b}$ we need to show that a maximum cardinality matching in $\text{sub}(G, l)$ can be obtained in polynomial time and analyze the structure of the Edmonds-Gallai decomposition of $\text{sub}(G, l)$.

We first transform the instance (G, l) into an equivalent instance $(G' = (V', E'), l')$ such that for each $e \in E'$ it holds that l'_e is odd. This can be easily achieved by subdividing every edge $e \in E$ with l_e even exactly once into edges e' and e''. We then set $l'_{e'} := l'_e - 1$ and $l'_{e''} := 1$. For all other edges e we set $l'_e := l_e$. By the same arguments as used for the equivalence of $\text{sub}(G, l)$ to the instance (G, l) the new instance (G', l') is equivalent to solving the instance (G, l) for arbitrary δ.

Lemma 2.18. *If l_e is odd for each $e \in E$ a maximum cardinality matching M' of $\text{sub}(G, l)$ is characterized by a maximum cardinality matching of G and it holds that $\nu(\text{sub}(G, l)) = \nu(G) + \sum_{e \in E} \frac{l_e - 1}{2}$.*

We have that M' contains exactly $\nu(G)$ subdivided edges e of G in which there are exactly $\frac{l_e+1}{2}$ edges of M' and these edges are a maximum matching in G. The subdivision of every other edge e of G contains exactly $\frac{l_e-1}{2}$ edges of M'.

Proof. We make the following two observations about the induced paths of odd length in $\text{sub}(G, l)$.

- A path of odd length l_e has exactly one perfect matching of cardinality $\frac{l_e+1}{2}$. In this matching obviously both end-vertices are matched.
- A path of odd length l_e has a matching of cardinality $\frac{l_e-1}{2}$ that leaves both end-vertices unmatched.

These two observations imply that a maximum matching in $\text{sub}(G, l)$ always contains at least $\frac{l_e-1}{2}$ edges in each path. Without loss of generality one can assume that the matching of this kind leaving both end-vertices unmatched is chosen, since this is the matching of this cardinality interfering the least with the rest of the graph. One additional matching-edge in a path can only be chosen by matching both end-vertices with matching-edges of the path. Choosing for which paths to do this corresponds to calculating a maximum cardinality matching in G. □

Lemma 2.19. *If l_e is odd for each $e \in E$ the Edmonds-Gallai decomposition (X', Y', Z') of $\text{sub}(G, l)$ has the following properties, based on the Edmonds-Gallai decomposition (X, Y, Z) of G:*

- *The vertices of subdivisions of a component C of X (both the original vertices and the subdivision ones) are in X'.*
- *The vertices of subdivisions of a component C of Z (both the original vertices and the subdivision ones) are in Z'.*
- *Let $\{y, x\}$ be an edge between $y \in Y$ and $x \in X$ in G. It holds that $y \in Y', x \in X'$ and the subdivision vertices of the edge alternate to be in X'_1 and Y'.*

- *The subdivision vertices of edges $\{y, z\}$ for $y \in Y$ and $z \in Z$ are in Z' and become part of the components C of Z they are connected to.*

Proof. The statement about vertices of subdivisions of a component C of X follows easily since to make a subdivision vertex unmatched one considers a matching in G that does not match one of the endpoints of the edge corresponding to the subdivision vertex. Then one can use a matching of cardinality $\frac{l_e-1}{2}$ inside the subdivision that matches the endpoint but does not match the designated subdivision vertex.

For the subdivision vertices corresponding to edges in a component C of Z, we know that all the original vertices are matched according to some perfect matching of C in G, and hence one can complete each other edge by the unique perfect matching of the other subdivision vertices. In a maximum matching one must do this, hence all such vertices are also in Z'.

Since for edges $\{y, z\}$ for $y \in Y, z \in Z$ it also holds after subdivision that both endpoints y and z are matched in every maximum matching we have that the unique perfect matching of the subdivision points of the edge is part of every maximum matching. Hence those subdivision points merge with the component of z in Z'.

The most relevant change in X', Y', Z' happens for the subdivision of edges $\{y, x\}$ for $y \in Y, x \in X$. By the fact that $x \in X$ there is a maximum matching in G that does not contain the edge $e = \{y, x\}$. Since in the factor-critical component connected to x we can select a maximum matching that keeps x free, we can easily check that all ways to select the $\frac{l_e-1}{2}$ edges along the subdivision of e is such that y and every second subdivision vertex after y is always matched and for every other subdivision vertex and also x there exists exactly one such a matching that keeps it unmatched. Hence the claim follows. □

Based on this it is easy to see that actually also minimizing $g(T)$ for the graph G gives the optimal solution for $\mathrm{sub}(G, l)$, assuming l is odd. The set of singletons $X'_1 = X_1 \cup S_1$, where S_1 are the subdivision points of edges $\{y, x\}$ for $y \in Y, x \in X$ with odd distance from y. Observe that the only way in which taking the subdivision points S_1 in a solution can increase the size (compared to taking a matching) is when the end-vertex x of the path is also a singleton in X_1 and is taken. In this case we can and have to take all the singletons and this has the same effect on y as as in G, since then along no subdivided edge e' incident to y the matching of size $\frac{l_{e'}+1}{2}$ can be taken. Hence by solving the minimization of $g(T)$ for the Edmonds-Gallai decomposition of G one obtains also the optimal solution for $\mathrm{sub}(G, l)$. In summary we obtain the following theorem.

Theorem 2.20. *The 2-dispersion number of a graph G with integer edge lengths l can be computed in polynomial time.*

2.7 Special Graph Classes

In the following we summarize some results about the δ-Disp(G) for graphs G with special structure.

Theorem 2.21. *If G has treewidth t we can compute δ-Disp(G) in time $f(t)n$ for fixed δ, where f is some computable function.*

Proof. This follows directly from the optimization variant of Courcelle's theorem [6].

We first reduce the given instance to $\delta \in \mathbb{N}$ using Lemma 2.2. Then based on Lemma 2.3 we can formulate the fact that a set $S \subseteq V$ and a set $M \subseteq E$ in combination form a δ-dispersed set using the following MSO2 formula:

$$
\begin{aligned}
&(\forall s_1, s_2 \in S \colon s_1 \neq s_2 \wedge \mathbf{distv}^{(>)}(s_1, s_2, 1) \wedge \cdots \wedge \mathbf{distv}^{(>)}(s_1, s_2, \delta - 1)) \wedge \\
&(\forall e_1, e_2 \in M \colon e_1 \neq e_2 \wedge \mathbf{diste}^{(>)}(e_1, e_2, 1) \wedge \cdots \wedge \mathbf{diste}^{(>)}(e_1, e_2, \delta - 1)) \wedge \\
&(\forall s \in S, e \in M \colon \mathbf{distve}^{(>)}(s, e, 1) \wedge \cdots \wedge \mathbf{distve}^{(>)}(s, e, \delta - 1)),
\end{aligned}
$$

where for all $k \in \mathbb{N}$

$$
\begin{aligned}
&\mathbf{distv}^{(>)}(s_1, s_2, k) := \\
&\quad \nexists v_1, \ldots, v_{k+1} \in V \colon s_1 = v_1 \wedge s_2 = v_{k+1} \wedge \mathbf{adj}(v_1, v_2) \wedge \cdots \wedge \mathbf{adj}(v_k, v_{k+1})), \\
&\mathbf{diste}^{(>)}(e_1, e_2, k) := \\
&\quad \nexists v_1, \ldots, v_k \in V \colon \mathbf{inc}(v_1, e_1) \wedge \mathbf{inc}(v_k, e_2) \wedge \mathbf{adj}(v_1, v_2) \wedge \cdots \wedge \mathbf{adj}(v_{k-1}, v_k)), \\
&\mathbf{distve}^{(>)}(s, e, k) := \\
&\quad \nexists v_1, \ldots, v_{k+1} \in V \colon v_1 = s \wedge \mathbf{inc}(v_{k+1}, e) \wedge \mathbf{adj}(v_1, v_2) \wedge \cdots \wedge \mathbf{adj}(v_k, v_{k+1})).
\end{aligned}
$$

□

It is an intersting open problem whether this problem is W[1]-hard on graphs with bounded clique-width. Also, the computational complexity of δ-dispersion on graphs with bounded treewith for arbitrary δ is an open problem.

3 Steiner Problems on Interval Graphs

3.1 Introduction

In this chapter we investigate the *Steiner cycle* and *Steiner path* problem in interval graphs. Our investigation was motivated by 2D platform games which is a common game mechanism to include platforms that fall or break after the player visits them once. Additionally, it is often the case that the player has to collect certain items (coins, stars, ...) that are placed on some of these platforms and afterwards get back to the start or reach the exit of the level. Popular examples of video games that are (partially) based on these principles include Super Mario Bros.[1] (see Figure 3.1), Donkey Kong Country and Super Mario Land. We study solvability of levels based on these principles by introducing a toy model of such video games, in which all platforms (except for the target/starting point) have this falling property. The reachability between two platforms is modeled via an interval graph, which in many cases is a reasonable simplification. Then, the solvability of a level boils down to either finding a Steiner cycle or a Steiner path in the corresponding interval graph. To our knowledge, these problems have not been studied for this specific graph class. The *Hamiltonian cycle* and *Hamiltonian path* problem, which are special cases of the Steiner variants, are extensively studied for interval graphs and can be solved in linear time, if the intervals are given as a right endpoint sorted list $[5, 73, 82, 97]$.

Figure 3.1: Super Mario Bros. (1985, NES)

In this work we generalize the algorithms of Manacher et al. [97] to the *Steiner setting* and obtain first linear time algorithms for the *Steiner path cover* and *Steiner cycle* problem on interval graphs. A second important aspect when considering 2D game

[1] Super Mario Bros. is a trademark of Nintendo. Sprites are used here under Fair Use for educational purposes.

levels is the fact that the screen size is limited, so the whole level is not visible to the player at once. By studying our algorithms as *single pass streaming algorithms* we state precisely which parts of a level have to be visible to the player to deterministically decide how to play at each time. Alternatively, this can be interpreted as a memory bound for the streaming algorithms in terms of a natural graph parameter for interval graphs. To obtain our results we extend the tools introduced in [73].

These problems in a more general model for platform game levels based on intersection graphs of two dimensional boxes are hard. Such graphs are generalizations of grid graphs for which already the Hamiltonian path problem is known to be NP-hard [74].

3.2 Definitions and Preliminary Results

Given an interval $i = [x, y]$ we denote the left endpoint x by $\mathrm{l}(i) = x$ and the right endpoint y by $\mathrm{r}(i) = y$. Let $I = (i_1, i_2, \ldots, i_n)$ be a list or set of intervals. We denote by $G(I)$ the interval graph of I. The vertices of $G(I)$ correspond to the intervals of I. Two intervals $i, i' \in I$ are connected by an edge in $G(I)$ if $i \cap i' \neq \emptyset$.

For an arbitrary graph $G = (V, E)$ a list of vertices $P = (v_1, v_2, \ldots, v_l)$ is a (simple) path if those vertices are pairwise distinct and for each $j = 1, 2, \ldots, l-1$ it holds that $\{v_j, v_{j+1}\} \in E$. The start of P is denoted by $\mathrm{start}(P) = v_1$ and the end of P is denoted by $\mathrm{end}(P) = v_l$. We define $\mathrm{rev}(P)$ as the reverse path $(v_l, v_{l-1}, \ldots, v_1)$ of P. If in addition $\{v_l, v_1\} \in E$ then we call P a (simple) cycle. For ease of writing we sometimes abuse notation and consider P as a set instead of a list, to allow for the use of set operations. Given two paths P and Q and a vertex v we also write (P, Q) for the concatenation of P and Q, and (P, v) for the concatenation of P and v.

Given a set $S \subseteq V$, a *Steiner cycle* is a cycle C in G such that $S \subseteq C$. A *Steiner path cover* of G is a set $\{P_1, P_2, \ldots, P_k\}$ of pairwise disjoint paths in G such that $S \subseteq \bigcup_{j=1}^{k} P_j$. The *Steiner path cover number* $\pi_S(G)$ is the minimum cardinality of a Steiner path cover of G. If $\pi_S(G) = 1$ we say that G has a *Steiner path*.

A set $C \subseteq V$ is called a *cutset* of G if $G - C$ is disconnected. A set of vertices $T \subseteq V$ is called an island with respect to C, if T is not adjacent to any vertex in $V \setminus (C \cup T)$. T is called an S-island with respect to C, if T is an island with respect to C and $S \cap T \neq \emptyset$.

The following two results for general simple graphs are generalizations of two observations by Hung and Chung [73].

Proposition 3.1. *Let C be a cutset of G and g_S the number of connected components K in $G - C$ such that $K \cap S \neq \emptyset$. Then, $\pi_S(G) \geq g_S - |C|$.*

Proof. Let $(P_1, P_2, \ldots, P_k)$ be a Steiner path cover of G. For every P_j let g_j be the number of components K of $G - C$ with $K \cap S \neq \emptyset$ and $K \cap P_j \neq \emptyset$. The P_j must use at least $g_j - 1$ distinct vertices from C to reconnect itself from those different components of $G - C$, i.e., $|P_j \cap C| \geq g_j - 1$. Now, since paths of a path cover are vertex disjoint, we have that $|C| \geq \sum_{j=1}^{k} (g_j - 1)$. Finally, from the fact that $\sum_{j=1}^{k} g_j \geq g_S$, we get $k \geq g_S - |C|$, which proves our claim. □

Proposition 3.2. *Let C be a cutset of G and g_S the number of connected components K in $G - C$ such that $K \cap S \neq \emptyset$. If $g_S > |C|$, then G has no Steiner cycle.*

Proof. A Steiner cycle needs to connect all components K of $G - C$ with $K \cap S \neq \emptyset$. For each such connection a distinct vertex from C has to be used. Since it is a cycle, it has to be closed, hence g_S such connections are necessary. This implies that if $g_S > |C|$ no Steiner cycle can exist. □

We use these results to solve the *Steiner path cover* problem (see Section 3.3) and the *Steiner cycle* problem (see Section 3.4) on interval graphs efficiently. Throughout the chapter we assume that $|S|$ is known to the algorithms and queries $i \in S$ can be performed in $O(1)$ time.

3.3 The Steiner Path Cover Problem

We show that the basic greedy principle, that is at the core of efficient algorithms for the *path cover* problem on interval graphs, can be generalized by the introduction of *neglectable intervals*. But first we explain the basic greedy principle to find paths in interval graphs that was introduced independently by Manacher et al. [97] and Arikati et al. [5].

Given a endpoint sorted list of intervals we number those intervals as $i_1, i_2, \ldots, i_n$ in increasing order with respect to their right endpoint, hence $r(i_j) < r(i_{j+1})$ for all $j = 1, 2, \ldots, n-1$. (Such a numbering can be easily obtained in $O(n)$ time using the endpoint sorted list and is assumed for the rest of this chapter. W.l.o.g. we can assume that $r(i_j) \neq r(i_k)$ for $i \neq k$.) The algorithm iteratively constructs a path P. It starts with the path $P := (i_1)$ containing only the first interval. Then it repetitively extends P by the neighbor of end(P) not in P with minimum right endpoint. If no such extension is possible the algorithm terminates with the current path P as an output. We denote this algorithm by **GP** and the path P obtained by this algorithm by $\mathbf{GP}(I)$.

For a path $P = \mathbf{GP}(I) = (v_1, v_2, \ldots, v_l)$ obtained by executing the algorithm on an interval graph $G(I)$, we define $L(P)$ as the set of intervals of P that exceed beyond the right endpoint of end(P), i.e. $L(P) = \{v \in P\colon \mathrm{r}(v) > \mathrm{r}(\mathrm{end}(P))\}$. Now we can recursively define $C(P)$, the set of *covers* of the path P, as follows. If $L(P) = \emptyset$, we set $C(P) = \emptyset$. Otherwise, let j be the maximum index such that $v_j \in L(P)$. We set $C(P) = \{v_j\} \cup C(P')$ for $P' = (v_1, v_2, \ldots, v_{j-1})$.

For $C(P) = \{c_1, c_2, \ldots, c_k\}$ and $P = (P_0, c_1, P_1, c_2, \ldots, c_k, P_k)$, Manacher et al. [97] proved that for each $j = 0, 1, \ldots, k$ it holds that P_j is an island with respect to $C(P)$ and if $I \setminus P \neq \emptyset$ also $I \setminus P$ is an island with respect to $C(P)$. Such a representation of P we call a *decomposition into covers and islands*.

Manacher et al. [97] also observed the following important properties of a decomposition into covers and islands.

Proposition 3.3. *Let $P = \mathbf{GP}(I) = (v_1, v_2, \ldots, v_l)$.*

1. *If $v_j \in C(P)$ it holds that $\mathrm{r}(v_j) > \mathrm{r}(v_{j+1})$.*

2. *If $P = (P_0, c_1, P_1, c_2, \ldots, c_k, P_k)$ is a decomposition into covers and islands, then it holds that $L(P_j) = \emptyset$ for each $j = 0, 1, \ldots, k$.*

To illustrate the notions introduced above, consider the intervals in Figure 3.2 given as a right endpoint-sorted list $I = (i_1, i_2, \ldots, i_{12})$. Algorithm **GP** starts by setting

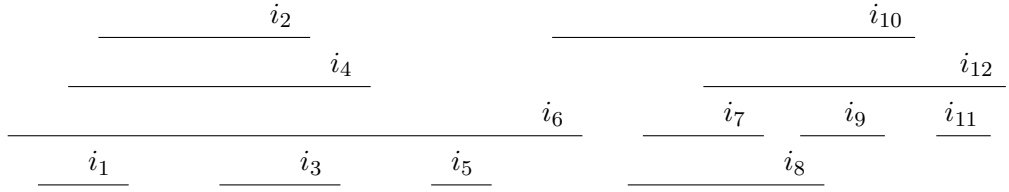


Figure 3.2: An interval model I of twelve endpoint-sorted intervals [73].

$P = (i_1)$. Neighbors of i_1 are $\{i_2, i_4, i_6\}$, and since $\mathrm{r}(i_2) < \min\{\mathrm{r}(i_4), \mathrm{r}(i_6)\}$ we extend P by i_2, i.e. $P = (i_1, i_2)$. Among neighbors of i_2 that are not already in P, i_3 has the smallest right endpoint, so P is extended to $P = (i_1, i_2, i_3)$. Next candidates for the extension are $\{i_4, i_6\}$ among which we chose i_4, i.e. $P = (i_1, i_2, i_3, i_4)$. Next, the only possible extension is by i_6, hence $P = (i_1, i_2, i_3, i_4, i_6)$. Among the next candidates for extension $\{i_5, i_{10}\}$, interval i_5 is chosen. At this point the algorithm terminates and outputs $P = (i_1, i_2, i_3, i_4, i_6, i_5)$, since there is no neighbor of i_5 that is not already in P.

Now we find a decomposition into covers and islands of P. Since $\mathrm{r}(i_6) > \mathrm{r}(\mathrm{end}(P) = i_5)$, we have that $L(P) = \{i_6\}$, and $C(P) = \{i_6\} \cup C(P' = (i_1, i_2, i_3, i_4))$. $L(P')$ is the empty set, so the decomposition process is over and we have that the decomposition into covers and islands of P is given by $C(P) = \{i_6\}$ and $P = (P_0, i_6, P_1)$, where $P_0 = (i_1, i_2, i_3, i_4)$ and $P_1 = (i_5)$. Note that P_0, P_1 and $I \setminus P$ are islands with respect to $C(P) = \{i_6\}$. Furthermore, note that our decomposition satisfies the properties in Proposition 3.3.

Given the fact that in the Steiner variant of the problem only the intervals in S have to be visited, we introduce *neglectable intervals*. Let P be the current path at any point of the greedy algorithm and v' be the next extension. We call v' *neglectable* with respect to $\mathrm{end}(P)$, if $v' \notin S$ and $\mathrm{r}(v') < \mathrm{r}(\mathrm{end}(P))$, i.e. $\mathrm{end}(P) \in L((P, v'))$. We modify the algorithm **GP**, such that it skips neglectable intervals with respect to the end of the current path. Analogously to **GP** this modification is denoted by $\mathbf{GP}_S$. Additional two distinctions of $\mathbf{GP}_S$ are that it starts with the interval (with the smallest $\mathrm{r}(v)$) that is in S, and it terminates as soon as there are no more uncovered intervals in S. We denote by N_v the set of intervals that are not contained in $\mathbf{GP}_S(I)$ because they are neglectable with respect to v for some path P during the execution of $\mathbf{GP}_S$, where $v = \mathrm{end}(P)$. Let N be the set of all such neglectable intervals obtained during the entire run of $\mathbf{GP}_S$.

Now we present a lemma which is our main tool for elegantly proving our main results.

Lemma 3.4. *Let $P = \mathbf{GP}_S(I)$ be the path obtained by $\mathbf{GP}_S$ for a given list of intervals I, let $P = (P_0, c_1, P_1, c_2, \ldots, P_{k-1}, c_k, P_k)$ be its decomposition into covers and islands in $G(I \setminus N)$, and let $C(P) = \{c_1, c_2, \ldots, c_k\}$. Then it holds for all $j = 0, 1, \ldots, k$ that*

$P_j \cap S \neq \emptyset$, i.e. P_j is an S-island with respect to $C(P)$ in $G(I \setminus N)$. It even holds that $P_j \cup N_{c_j}$ contains at least one S-island with respect to $C(P)$ in $G(I)$.

Proof. It is easy to see that this decomposition into covers and islands exists, since if $P = \mathbf{GP}_S(I)$ it follows by construction that $P = \mathbf{GP}(I \setminus N)$.

The fact that P_j is an S-island with respect to $C(P)$ in $G(I \setminus N)$ is a trivial consequence of the decomposition into covers and islands and the definition of $\mathbf{GP}_S$. Since c_j is used before every interval in N_{c_j} we have that the left endpoint of every interval in N_{c_j} is larger than the left endpoint of c_j. The right endpoints of each of the intervals in N_{c_j} is smaller than the right endpoint of c_j by definition of neglected intervals. But this directly implies that $C(P)$ separates also N_{c_j} from the rest of $G(I)$, except for possibly P_j. □

Now we can obtain an easy procedure to solve the Steiner path cover problem on interval graphs. We start with $\mathcal{P} = \emptyset$ and apply the algorithm $\mathbf{GP}_S$. After termination let $P = \mathbf{GP}_S(I)$. We add P to our partial solution $\mathcal{P}$ and find the smallest index j such that $i_j \in S$ and i_j is not in any path currently contained in $\mathcal{P}$. Then we apply $\mathbf{GP}_S$ again to the list of intervals $i_j, i_{j+1}, \ldots, i_n$. We iterate like this until all intervals in S are covered by one of the paths in $\mathcal{P}$. The algorithm terminates with the Steiner path cover $\mathcal{P}$ as its output.

Theorem 3.5. *The Steiner path cover obtained by iterated application of $\mathbf{GP}_S$ is optimal.*

Proof. Let $P_1, P_2, \ldots, P_l$ be the paths obtained by the iterated application of $\mathbf{GP}_S$ and $C' = \bigcup_{j=1}^{l} C(P_j)$ be the union of all the covers in the decomposition into covers and islands of each path. Then, by repeated application of Lemma 3.4 we obtain that there are $l + |C'|$ S-islands with respect to C' in $G(I)$. By Proposition 3.1 we then know that $\pi_S(G(I)) \geq l$, so our solution is an optimal Steiner path cover. □

To illustrate our algorithm for the Steiner path cover problem we again consider the example in Figure 3.2. In the case when $S = I$, i.e., all intervals need to be covered, our algorithm runs $\mathbf{GP}_S(I)$ which outputs $P' = (i_1, i_2, i_3, i_6, i_5)$, and then it runs $\mathbf{GP}_S(I \setminus P')$ which outputs $P'' = (i_7, i_8, i_9, i_{10}, i_{12}, i_{11})$, and the algorithm terminates. Therefore, for $S = I$ we have that $\pi_S(I) = 2$. Now lets say that $S = \{i_2, i_4, i_6, i_8, i_{10}, i_{12}\}$. $\mathbf{GP}_S(I)$ starts with the element of S with the smallest right endpoint which is i_2. Then it extends the path with i_3, i_4 and then i_6. After that, the algorithm neglects i_5 since $\mathrm{r}(i_5) < \mathrm{r}(i_6)$ and $i_5 \notin S$. Next, the path is extended by i_{10}, then i_7 is neglected, but i_8 is added to the path (since $i_8 \in S$). Then the path is extended by i_9 and finally by i_{12}. Interval i_{11} is neglected. The output of the algorithm is the path $P = (i_2, i_3, i_4, i_6, i_{10}, i_8, i_9, i_{12})$, so $\pi_S(I) = 1$. Note that the key factor that allowed us to cover the set S with only one path is the fact that we could neglect i_5.

By using the deferred-query approach by Chang et al. [31] this algorithm can be implemented in $O(n)$ time.

Theorem 3.6. *The iterated application of* $\mathbf{GP}_S$ *can be implemented in* $O(n)$ *time, using the deferred-query approach.*

Proof. Firstly, we give a high-level explainantion of how to implement $\mathbf{GP}_S$ using the deferred-query technique. Afterwards we argue how the modifictions can still be implemented in linear time.

In the deferred-query approach the algorithm handles the intervals in the given right endpoint sorted order one after another. For each j where $i_{j-1} \cap i_j \neq \emptyset$ the algorithm can be executed as stated above, since in this case we have that $\text{end}(P) = i_{j-1}$ and we extend P with i_j. The main difference is, that when $i_{j-1} \cap i_j = \emptyset$ we still have to process i_j instantly.

This is handled in the following way: The algorithm keeps at each time a collection of paths $P_1, \ldots, P_l$ and a flag that indicates whether P_l already contains an interval from S. (For all other paths it is an invariant of the algorithm that they always contain an interval from S). In the beginning we have that $l = 1$ and $P_1 = (i_1)$, where we assume that $i_1 \in S$. When handling i_j we have the following case distinction.

(a) $\text{end}(P_l) \cap i_j \neq \emptyset$ and $\text{end}(P_k) \cap i_j = \emptyset$ for all $k < l$: in this case, if P_l contains an interval from S we extend P_l by i_j, hence $P_l := (P_l, i_j)$. Otherwise we set $P_l := (i_j)$.

(b) There is a $k < l$ such that $\text{end}(P_k) \cap i_j \neq \emptyset$: let k be minimum with this property. Then also $\text{start}(P_{k+1}) \cap i_j \neq \emptyset$ and hence we can connect P_k and P_{k+1} using i_j, hence the new collection of paths is $P_1, \ldots, (P_k, i_j, P_{k+1}), \ldots, P_l$. if $k + 1 = l$ and P_l does not contain an interval from S we instead set the new collection of paths to $P_1, \ldots, (P_{l-1}, i_j)$.

(c) $\text{end}(P_l) \cap i_j = \emptyset$: In this case if P_l contains an interval from S we add a new path $P_{l+1} := (i_j)$, hence the new collection of paths is $P_1, P_2, \ldots, P_l, P_{l+1}$. Otherwise we replace $P_l := (i_j)$.

It is now easy to see that after termination this algorithm obtains exactly the set $(P_1, \ldots, P_l)$ that is the output of iterating algorithm $\mathbf{GP}_S$. The main observation is that the intervals that are removed by the procedure above are either neglectable or are intervals not in S that are are strictly between $\text{r}(\text{end}(P_k))$ and $\text{l}(\text{start}(P_{k+1}))$ for some $k = 1, \ldots, l - 1$.

It remains to show that interval i_j can be handled in $O(1)$ time. This follows directly by the implementation based on *static tree set union* shown in by Chang et al. [31], since the only difference is that in each step we have to do a case distinction for whether the current P_l contains an interval from S. The operations performed then correspond to operations already performed by the original algorithm, and removing the current last path P_l. This remove operation can obviously also be handled in constant time. □

3.4 The Steiner Cycle Problem

Next we generalize the algorithm of Manacher et al. [97] to solve the Steiner cycle problem on interval graphs. We first run our algorithm for the Steiner path cover problem (see

Section 3.3). If $\pi_S > 1$ we know that there cannot exist a Steiner cycle. Otherwise, let $P = (v_1, v_2, \ldots, v_l)$ be the obtained Steiner path in $G(I)$.

Based on P we construct two paths Q and R. We start by setting $R = (v_1)$ and $Q = (v_2)$. Then, we iteratively process the intervals v_3 to v_l. If in the step of processing interval v_j we have that $\text{end}(Q) = v_{j-1}$, we consider the following two cases. Firstly, if $v_j \cap \text{end}(R) \neq \emptyset$, we extend R by v_j, i.e. $R = (R, v_j)$. Otherwise, we extend Q by v_j, i.e. $Q = (Q, v_j)$. If on the other hand in this step we have that $\text{end}(R) = v_{j-1}$ we analogously check if $v_j \cap \text{end}(Q) \neq \emptyset$. If this is the case we extend Q by v_j and if not we extend R by v_j.

In the end of this process we try to connect R and $\text{rev}(Q)$ to a Steiner cycle. To achieve this we check if $\text{end}(Q)$ and $\text{end}(R)$ are directly connected, i.e. $\text{end}(Q) \cap \text{end}(R) \neq \emptyset$, or if there is an interval v' among the intervals $I' \subseteq I \setminus P$, whose right endpoints $\text{r}(v') > \text{r}(v_l)$ such that both $\text{end}(Q) \cap v' \neq \emptyset$ and $\text{end}(R) \cap v' \neq \emptyset$. In any of those two cases we can connect Q and $\text{rev}(R)$ to a Steiner cycle. Otherwise, the algorithm returns that no Steiner cycle exists.

Theorem 3.7. *The given algorithm correctly decides the existence of a Steiner cycle in $G(I)$ and obtains such a cycle if possible.*

Proof. If the algorithm finds a Steiner cycle this is obviously true. Also, by correctness of the algorithm for the Steiner path cover (Theorem 3.5), if no Steiner path is found we correctly determine that no Steiner cycle can exist.

Otherwise, let us assume that the algorithm did not find a Steiner cycle. Without loss of generality, let $\text{end}(R) = v_h$ with $h < l - 1$ and consider the path $P' = (v_1, v_2, \ldots, v_h)$ and its decomposition into covers and islands. Since R was not extended by any of the intervals $v_{h+2}, v_{h+3} \ldots, v_l$, we have that $C(P') \cup \{v_{h+1}\}$ separates the islands of P' from $\{v_{h+2}, v_{h+3}, \ldots, v_l\}$. In addition, since $\text{end}(R)$ and $\text{end}(Q)$ could not be connected with any interval in I', for all intervals $v' \in I'$ it holds that $\text{l}(v') > \text{r}(v_h)$. Combining this with point 2 of Proposition 3.3 we observe that $\{v_{h+2}, v_{h+3}, \ldots, v_l\} \cup I'$ is non-empty and an S-island with respect to $C(P') \cup \{v_{h+1}\}$.

By Lemma 3.4 there are at least $|C(P')| + 1$ S-islands with respect to $C \cup \{v_{h+1}\}$. So, by Proposition 3.2 there does not exist a Steiner cycle in $G(I)$. □

Given a Steiner path P, the paths Q and R can be easily constructed in $O(n)$ time. This gives a linear time algorithm for the Steiner cycle problem in interval graphs.

Now we illustrate our algorithm for the Steiner cycle problem on interval graphs with the example given in Figure 3.3. The given instance has 10 intervals $I = \{i_1, i_2, \ldots, i_{10}\}$ and $S = \{i_2, i_5, i_8\}$. Intervals in S are represented with the red color. First we run $\mathbf{GP}_S(I)$. It starts the path with i_2 and then extends it with i_3 and i_5 before neglecting i_4. Then it proceeds by extending the path with i_6, i_7, finishing with i_8. Hence it obtains the Steiner path $P = (i_2, i_3, i_5, i_6, i_7, i_8)$. In an attempt to create a Steiner cycle, we partition P into two paths R and Q. We initialize them with the first two intervals in P, that is, $R = (i_2)$ and $Q = (i_3)$. Now we consider Q to be the current path, and R to be the previous path. In each step we consider the next interval of P, and in the case

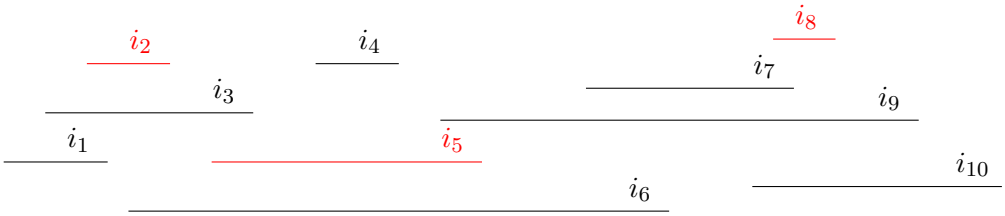


Figure 3.3: An instance of the Steiner cycle problem on an interval graph with $S = \{i_2, i_5, i_8\}$.

that it intersect the end of the previous path, we extend the previous path and make it the current path. Otherwise we add the interval to the current path. So, interval i_5 is the next interval in P, and it does not intersect $\text{end}(R) = i_2$, hence we add it to Q, making it $Q = (i_3, i_5)$. The next interval is i_6, and it intersects $\text{end}(R) = i_2$, hence we extend R and make it the current path, so $R = (i_2, i_6)$. Next interval i_7 does not intersect $\text{end}(Q) = i_5$ so we extend R again, making it $R = (i_2, i_6, i_7)$. Finally, interval i_8 does not intersect $\text{end}(Q) = i_5$ so we extend R, making it $R = (i_2, i_6, i_7, i_8)$. This ends our partition of P with the resulting subpaths $R = (i_2, i_6, i_7, i_8)$ and $Q = (i_3, i_5)$. Since $\text{end}(R) = i_8$ and $\text{end}(Q) = i_5$ do not intersect, we cannot connect them into a cycle. The only remaining chance to do so is using an interval from $I' = \{i \in I \setminus P \colon \text{r}(i) > \text{r}(\text{end}(P))\} = \{i_9, i_{10}\}$. Luckily, i_9 intersect both $\text{end}(R) = i_8$ and $\text{end}(Q) = i_5$, and can be used to connect R and Q into a cycle. The Steiner cycle is then given by $(R, i_9, \text{rev}(Q)) = (i_2, i_6, i_7, i_8, i_9, i_5, i_3)$.

Now let us consider a modified instance of Figure 3.3, where i_4 is also an element of S. Then $\mathbf{GP}_S(I)$ would output the path $P = (i_2, i_3, i_5, i_4, i_6, i_7, i_8)$, and the subsequent partition of P would give $R = (i_2, i_6, i_7, i_8)$ and $Q = (i_3, i_5, i_4)$. But now there is no interval in I' that connects $\text{end}(R) = i_8$ and $\text{end}(Q) = i_4$, so our algorithm outputs that there is no Steiner cycle. In order to verify that there is no Steiner cycle we can follow the arguments in the proof of Theorem 3.7, which gives us a cutset $C = \{i_5, i_6\}$ that separates I into three S-islands, and hence, by Proposition 3.2, guarantees that there is no Steiner cycle.

3.5 Streaming Algorithms – The Problem of Limited Screen Size

An important question when considering solvability of game levels is which parts of a level have to be visible to the user at any time for them to deterministically know how to play correctly. To answer this question for our toy model, we study the algorithms from Section 3.3 and 3.4 as streaming algorithms. We assume that the input stream is presented as a sequence of right endpoint sorted intervals which can only be examined in one pass. As its output the streaming algorithm has to write the list of intervals giving the paths or cycle.

First, consider the algorithm $\mathbf{GP}_S$. In each step this algorithm needs access to the next interval on the stream that is connected with the current path $\text{end}(P)$. If the next

interval i on the stream is not connected to end(P) there can be two reasons. This interval could either be in a new different connected component than P, or it could be connected to P via another interval i' with $\mathrm{r}(i') > \mathrm{r}(i)$. Intervals of this kind are all completely contained in i'. After processing and storing all such intervals we clearly know whether the graph is disconnected or the path P can be extended and we can further process the stored intervals. This motivates the introduction of the parameter $\kappa(I)$, the maximum number of intervals contained in another interval. Based on this parameter we observe that $\mathbf{GP}_S$ can be implemented as a single pass streaming algorithm with $O(\kappa(I))$ additional storage. Based on this we obtain the following result.

Theorem 3.8. *Given $\kappa(I)$ the Steiner path cover problem on interval graphs can be solved by a single pass streaming algorithm in $O(n)$ time with $O(\kappa(I))$ additional storage.*

Remark. *If $\kappa(I)$ is not known to the algorithm the same result only holds assuming $G(I)$ is connected. Otherwise in the case of a disconnected interval graph the algorithm can not decide after $O(\kappa(I))$ steps that the graph is disconnected. It has to continue to store the intervals from the stream till the end, because there is no way of knowing if a future interval will be connected to* end(P) *for the current path P.*

On the other hand if $\kappa(I)$ is known we can stop this process after storing $\kappa(I)$ intervals since we know that no more of them can be contained in another interval and terminate with the current path P.

To solve the Steiner cycle problem, a single pass streaming algorithm can no longer first run $\mathbf{GP}_S$ and then construct the two paths Q and R, since this would need two passes. Also the output of the cycle is only possible in a single pass, without a large amount of additional memory, if the two paths Q and R are accepted as an output instead of the list for the Steiner cycle. In the application to platform games this is not a problem since here a player actually is doing first a pass from the left to the right and then another pass from the right to the left. So correct construction of Q during the first pass is enough to guarantee the possibility of getting back to the exit later. This path for the way back can then be easily found doing a simple greedy approach (see the description in the end of the current section).

The construction of Q and R can be incorporated into the streaming variant of $\mathbf{GP}_S$ described above without the need for additional memory. In addition to end(P) we also store end(Q) and end(R). This way in each step of the algorithm we can decide whether the next interval extending P should be appended to Q or R, by the same method as explained in Section 3.4. This only needs additional memory for storing both end(Q) and end(R) compared to just executing $\mathbf{GP}_S$.

Theorem 3.9. *Given $\kappa(I)$ the Steiner cycle problem on interval graphs can be decided by a single pass streaming algorithm in $O(n)$ time with $O(\kappa(I))$ additional storage.*

It is important to note that from the view of a player the additional storage in the streaming algorithms does not correspond to storage needed to decide the next step of the game but to the range of the level that has to be visible to the player. It covers the fact that the player has to be able to see at least the next two intervals reachable from

its current position and all the intervals before that in a right endpoint sorted order. The two things a player needs to remember at each point of the game are end(Q), the platform it is currently on, and end(R). The algorithm can also be simplified in the following way.

Assume the player is currently located on the interval end(Q). There are two possible cases. In the first case the last step was jumping onto end(Q). Let i be the interval reachable from end(Q) with r(i) minimum, such that i is not neglectable with respect to end(Q). If $i \cap \text{end}(R) \neq \emptyset$ we extend R, so the player remembers end(R) $= i$. Otherwise the player jumps to i, so end(Q) $= i$. If neither is possible the current level is unsolvable. In the second case the last step was an extension of R, so end(R) was updated. Let i be the interval reachable from end(R) with r(i) minimum, such that i is not neglectable with respect to end(R). If $i \cap \text{end}(Q) \neq \emptyset$ the player jumps to i, so end(Q) $= i$. Otherwise we extend R so the player remembers that end(R) $= i$. If neither is possible the current level is also unsolvable. If the last interval in S is either visited by the player, i.e. is equal to end(Q) or reached by R, i.e. is equal to end(R) the player tries to reach end(R) from end(Q) by jumping there directly or using an interval i' with $\text{r}(i') > \max\{\text{r}(\text{end}(Q)), \text{r}(\text{end}(R)\}$. If this is not possible the player determines that the current level is unsolvable. Otherwise it can easily get back to the exit visiting all the unvisited intervals in S by reconstructing a maybe permuted version of the path rev(R). Let i be the interval the player is currently on. In each step it can greedily jump to the reachable interval i' with maximum left endpoint l(i'), such that i' is not neglectable with respect to i in the reverse sense. This means we can neglect jumping to i' if $\text{l}(i) < \text{l}(i')$ and $i' \notin S$. This is just an application of $\mathbf{GP}_S$ in reverse direction. Since the path R exists, by the optimality of $\mathbf{GP}_S$ for the path cover problem, using this strategy the player finds a path R' covering all intervals in S and returning to the start of the level.

3.6 Conclusion

We obtained linear time algorithms for both the Steiner path cover problem and the Steiner cycle problem, assuming the intervals are given as a right endpoint sorted list. We also analyzed those algorithms as single pass streaming algorithms to study solvability of a simplified model for platform game levels.

Our simplification reduced those levels to a one-dimensional interval graph model. The hamiltonian cycle and path problems for two-dimensional generalizations of interval graphs are known to be NP-hard. It would be of interest to study special cases of these problems inspired from game levels. Furthermore the analysis of streaming algorithms for interval graphs is a natural extension to classic algorithms for interval graphs. Understanding other efficient algorithms for different problems on interval graphs in this model is a very interesting area for further research.

4 Combinatorial Optimization with Interaction Costs

4.1 Introduction

Let a family $\mathcal{F}_1$ of subset of $[m] = \{1, 2, \ldots, m\}$, and a family $\mathcal{F}_2$ of subsets of $[n] = \{1, 2, \ldots, n\}$ represent feasible solutions. For each element $i \in [m]$ a linear cost c_i is given. Also, for each element $j \in [n]$ a linear cost d_j is given. In addition, for any $(i, j) \in [m] \times [n]$ their *interaction cost* q_{ij} is given. Then the *combinatorial optimization problem with interaction costs* (COPIC) is the problem of finding $S_1 \in \mathcal{F}_1$ and $S_2 \in \mathcal{F}_2$ such that

$$f(S_1, S_2) = \sum_{i \in S_1} \sum_{j \in S_2} q_{ij} + \sum_{i \in S_1} c_i + \sum_{j \in S_2} d_j \tag{4.1}$$

is minimized. We denote an instance of COPIC by $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$, where $Q = (q_{ij})$ is the interaction cost matrix and $c = (c_i)$, $d = (d_j)$ are linear cost vectors of the instance. This generalizes the classical *linear cost combinatorial optimization problem*, where for a given family $\mathcal{F}$ of subsets of $[n]$, and cost vector $w \in \mathbb{R}^n$ one tries to find a set $S \in \mathcal{F}$ minimizing $\sum_{i \in S} w_i$. We denote an instance of LCOP by $(\mathcal{F}, w)$.

COPIC generalizes many well studied combinatorial optimization problems. For example, when $\mathcal{F}_1$ and $\mathcal{F}_2$ are respectively the family of perfect matchings in bipartite graphs G_1 and G_2 with respective edge sets $[m]$ and $[n]$, then COPIC reduces to the *bilinear assignment problem* (BAP) [40]. BAP is a generalization of the well studied *quadratic assignment problem* [43] and the *three-dimensional assignment problem* [113] and hence COPIC generalizes these problems as well. When $\mathcal{F}_1$ and $\mathcal{F}_2$ contain all subsets of $[m]$ and $[n]$ respectively, COPIC reduces to the *bipartite unconstrained quadratic programming problem* [44, 63, 77, 107] studied in the literature by various authors and under different names. Also, when $\mathcal{F}_1$ and $\mathcal{F}_2$ are feasible solutions of assignment constraints, where $[m]$ and $[n]$ correspond to edges of the complete bipartite graph $K_{n,n}$ and $K_{m,m}$, respectively, COPIC reduces to the *bipartite quadratic assignment problem* and its variations [38, 108]. Most quadratic combinatorial optimization problems can also be viewed as special cases of COPIC, including the *quadratic minimum spanning tree problem* [8], *quadratic set covering problem* [9], *quadratic travelling salesman problem* [75], etc. Thus all the applications studied in the context of these special cases are applications of COPIC as well. COPIC is a special case of bilinear integer programs [3, 53, 86] when $\mathcal{F}_1$ and $\mathcal{F}_2$ can be represented by polyhedral sets.

In this chapter we investigate various theoretical properties of COPIC. To understand the impact of interaction costs in combinatorial optimization we will analyze special cases of the interaction cost matrix Q for representative well-studied sets of feasible

solutions. Among others, the classes of interaction cost matrices Q that we will be focused on in this chapter include matrices of fixed rank, and diagonal matrices. In the literature many quadratic-like optimization problems have been investigated in the context of fixed rank or low rank cost matrices, for example see [4,17,107,118]. Further, the importance of investigating COPIC with diagonal matrices is illustrated by its direct connections to problems of disjointness of combinatorial structures [50, 56, 110, 116], packing, covering and partitioning problems [11], as well as to problems of congestion games [2,117]. In this chapter we also pose the problem of identifying cost structures of COPIC instances that can be reduced to an instance with no interaction costs. These instances are called *linearizable instances* [30, 39, 40, 76, 106]. We suggest an approach of identifying such instances for COPIC with specific feasible solution structures along with a characterization of linearizable instances.

The aforementioned topics are investigated on COPIC's with representative well-studied sets of feasible solutions $\mathcal{F}_1$, $\mathcal{F}_2$. To make easy future references to different sets of feasible solutions we introduce shorthand notations. We denote by $2^{[n]} = \{S\colon S \subseteq [n]\}$ the unconstrained solution set. Given a matroid $\mathcal{M}$ we denote by $\mathcal{B}(\mathcal{M})$ the set of bases of $\mathcal{M}$. We denote by $\mathcal{U}_n^k$ the uniform matroid, whose base set $\mathcal{B}(\mathcal{U}_n^k)$ is the set of all k-sets of $[n]$. Given a graph G, $\mathcal{M}(G)$ is the graphic matroid of G, whose base set $\mathcal{B}(\mathcal{M}(G))$ is the set of all spanning trees of G (or spanning forests if G is not connected). The set of all maximum matchings of G is denoted by $\mathcal{PM}(G)$. Given two terminals $s, t \in V(G)$ the set of all s-t-paths in G is denoted by $\mathcal{P}_{s,t}(G)$. If G is a directed graph $\mathcal{P}_{s,t}(G)$ is the set of all directed s-t-paths in G. The set of all cuts in G is denoted by $\mathcal{CUT}(G)$ and $\mathcal{CUT}_{s,t}(G)$ is the set of all s-t-cuts in G.

Using these definitions, the bipartite unconstrained quadratic programming problem [107] is equivalent to the instance $(2^{[m]}, 2^{[n]}, Q, c, d)$ of COPIC.

The structure of this chapter is as follows. We begin by discussing the complexity of COPIC with no significant constraints on the cost structure in Section 4.2. Section 4.3 investigates the case when the interaction cost matrix Q is of fixed rank. Using methods from parametric optimization we show that in the case when one of the solution sets is unconstrained, i.e. $\mathcal{F}_1 = 2^{[n]}$ or $\mathcal{F}_2 = 2^{[m]}$, and linear cost optimization over the other solution set can be done in polynomial time, the problem becomes polynomially solvable. Further, we show that approximability may be achieved in the case of Q with fixed rank. We also show that if the number of breakpoints of multi-parametric linear optimization over both sets of feasible solutions is polynomially bounded and if Q has fixed rank, then COPIC can be solved in polynomial time. Section 4.4 investigates COPIC's where interaction cost matrix Q is diagonal. That is, there is a one-to-one relation between ground elements of $\mathcal{F}_1$ and $\mathcal{F}_2$ and the interaction costs appear only between the pairs of the relation. The complexity of COPIC with various well-knows feasible structures (matroids, paths, matchings, cuts, etc.) in the context of diagonal matrix Q are considered, and their relationship to some existing results in the literature is presented. Characterization of linearizable instances is investigated in Section 4.5. The chapter is concluded with Section 4.6, where we summarize the results and suggest some problems for future work.

4.2 General Complexity

Being a generalization of many hard combinatorial optimization problems, the general COPIC is NP-hard. Moreover, even for the "simple" case with no constraints on the feasible solutions it results in the bipartite unconstrained quadratic programming problem which is NP-hard [107]. A COPIC instance $(2^{[m]}, 2^{[n]}, Q, c, d)$ can easily reduced to a COPIC instance for most other sets of feasible solutions $\mathcal{F}_1$ and $\mathcal{F}_2$, which implies again NP-hardness for COPIC with those feasible solutions. However, COPIC instances $(2^{[m]}, 2^{[n]}, Q, c, d)$ are known to be solvable in polynomial time if $Q \leq 0$ and if $Q, c, d \geq 0$ (see Punnen et al. [107]). This is not true anymore if $\mathcal{F}_1, \mathcal{F}_2$ are bases of a uniform matroid, for which we obtain the following hardness result.

Theorem 4.1. COPIC *is strongly NP-hard, even if instances are restricted to the form* $(\mathcal{B}(\mathcal{U}_m^{k_1}), \mathcal{B}(\mathcal{U}_n^{k_2}), Q, 0, 0)$ *and* $Q \geq 0$.

Proof. We give a reduction from a strongly NP-hard version of the cardinality constrained directed minimum cut problem.

Let $\vec{K}_{m,n}$ be a digraph with vertex sets $[m]$ and $[n]$ and arcs (i, j) for each $i \in [m]$ and $j \in [n]$. The *k-card min directed cut problem* asks for a minimum cost directed cut $\delta^+(S) = \{(i, j) \colon i \in S, j \notin S\}$ such that $|\delta^+(S)| = k$. Using similar arguments as in [21] one can show that this directed version of the minimum cut problem is strongly NP-hard. Now we show how this problem can be solved in polynomial time, assuming a polynomial time algorithm for COPIC instances $(\mathcal{B}(\mathcal{U}_m^{k_1}), \mathcal{B}(\mathcal{U}_n^{k_2}), Q, 0, 0)$ exists.

For each $k_1 = 1, 2, \ldots, m$ check if $\frac{k}{k_1}$ is an integer. If so set $k_2 = \frac{k}{k_1}$ and solve the COPIC instance $(\mathcal{B}(\mathcal{U}_m^{k_1}), \mathcal{B}(\mathcal{U}_n^{k_2}), Q, 0, 0)$, obtaining solution sets S_1, S_2. Note that $|S_1||S_2| = k$, i.e. it corresponds to exactly k edges. We can define an equivalent directed cut $\delta^+(S)$ by setting $S = S_1 \cup ([n] \setminus S_2)$. This way the directed cuts $\delta^+(S)$ are in one to one correspondence with solutions of COPIC. Doing this for all possible pairs (k_1, k_2), we can obtain all possible k-cuts as feasible solutions of COPIC instances $(\mathcal{B}(\mathcal{U}_m^{k_1}), \mathcal{B}(\mathcal{U}_n^{k_2}), Q, 0, 0)$. Taking the minimum found via all such COPIC problems solves the k-card directed min cut problem in the given bipartite digraph. □

Theorem 4.1 can be used to show that COPIC instances restricted to the form $(\mathcal{F}_1, \mathcal{F}_2, Q, 0, 0)$ is NP-hard already for $Q \geq 0$ for most sets of feasible solutions $\mathcal{F}_1, \mathcal{F}_2$. The reason is that cardinality constraints can be often reduced to other more complicated sets of feasible solutions.

On the positive side, if we fix one of the two solutions, e.g. $S_1 \in \mathcal{F}_1$, then finding the corresponding optimal solution $S_2 \in \mathcal{F}_2$ reduces to solving the LCOP instance $(\mathcal{F}_2, h)$, where

$$h_j = \sum_{i \in S_1} q_{ij} + d_j \quad \text{for } j \in [n]. \tag{4.2}$$

This implies that if the cardinality of one set of feasible solutions, say $\mathcal{F}_1$, is polynomially bounded in the size of the input, then we can solve COPIC by solving linear LCOP instances $(\mathcal{F}_2, h)$ (where h is defined by (4.2)) for all $S_1 \in \mathcal{F}_1$.

Theorem 4.2. *If $m = O(\log n)$ and the* LCOP *instance $(\mathcal{F}_2, h)$ can be solved in polynomial time for any cost vector $h \in \mathbb{R}^n$, then the* COPIC *instance $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ can be solved in polynomial time.*

4.3 The Interaction Matrix with Fixed Rank

In this section we investigate the behavior of COPIC in terms of complexity and approximability when the rank of the interaction costs matrix Q is fixed. In the literature, many optimization problems have been investigated in the context of fixed rank or low rank cost matrices. This also includes problems with quadratic-like objective functions. For example, the Koopmans-Beckmann QAP [17], the unconstrained zero-one quadratic maximization problem [4], bilinear programming problems [118], non-convex quadratic programming [67], the bipartite unconstrained quadratic programming problem [107], among others.

Let $\mathrm{rk}(Q)$ denotes the rank of a matrix Q. Then $\mathrm{rk}(Q)$ is at most r, if and only if there exist vectors $a_p = (a_1^{(p)}, a_2^{(p)}, \ldots, a_m^{(p)}) \in \mathbb{R}^m$ and $b_p = (b_1^{(p)}, b_2^{(p)}, \ldots, b_n^{(p)}) \in \mathbb{R}^n$ for $p = 1, 2, \ldots, r$, such that

$$Q = \sum_{p=1}^{r} a_p b_p^T. \tag{4.3}$$

We say that (4.3) is a *factored form* of Q. Then the instance $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ of COPIC, where Q is of fixed rank r, becomes minimizing

$$f(S_1, S_2) = \sum_{p=1}^{r} \left(\sum_{i \in S_1} a_i^{(p)} \sum_{j \in S_2} b_j^{(p)} \right) + \sum_{i \in S_1} c_i + \sum_{j \in S_2} d_j, \tag{4.4}$$

such that $S_1 \in \mathcal{F}_1$, $S_2 \in \mathcal{F}_2$.

In the following, we show that if $\mathcal{F}_1 (= 2^{[m]})$ is unrestricted, i.e. the set of all subsets of $[m]$, then we can generalize the results of Punnen et al. [107] to solve the problem. Using methods of multi-parametric optimization we also demonstrate how to tackle more-general problems where both sets of feasible solutions are constrained, if their parametric complexity is bounded.

These results are obtained using methods from binary and linear optimization. To apply these techniques we will formulate our problem in terms of binary variables. We achieve this in a straightforward way, by introducing variables $x \in \{0,1\}^m, y \in \{0,1\}^n$ in one to one correspondence with a solution S_1, S_2, such that $x_i = 1$ iff $i \in S_1$, and $y_j = 1$ iff $j \in S_2$. The vector x and y are respectively called the incidence vectors of S_1 and S_2. Thus the family of feasible solutions can be represented in terms of the incidence vectors, i.e. $\mathcal{F}_1' = \{x \in \{0,1\}^m \colon S_1 \in \mathcal{F}_1 \text{ and } (x_j = 1 \Leftrightarrow j \in S_1)\}$ and $\mathcal{F}_2' = \{y \in \{0,1\}^n \colon S_2 \in \mathcal{F}_2 \text{ and } (y_j = 1 \Leftrightarrow j \in S_2)\}$. Now, rank r COPIC can be

formulated as the binary optimization problem:

$$\begin{aligned} \min\ & \sum_{p=1}^{r}(a_p^T x)(b_p^T y) + c^T x + d^T y \\ \text{s.t.}\ & x \in \mathcal{F}_1',\ y \in \mathcal{F}_2' \end{aligned}$$

4.3.1 One-Sided Unconstrained Fixed Rank COPIC

In this section we consider the case where $\mathcal{F}_1' = \{0,1\}^m$. Observe that COPIC is equivalent to the following linear relaxation of the binary constraint $x \in \{0,1\}^m$.

$$\begin{aligned} \min\ & \sum_{p=1}^{r}(a_p^T x)(b_p^T y) + c^T x + d^T y \\ \text{s.t.}\ & x \in [0,1]^m,\ y \in \mathcal{F}_2' \end{aligned}$$

To solve this problem, consider the multi-parametric linear program (MLP)

$$\begin{aligned} h_1(\lambda) := \min\ & c^T x \\ \text{s.t. } & a_p^T x = \lambda_p \quad \text{for } p = 1, 2, \ldots, r \\ & x \in [0,1]^m, \end{aligned}$$

where $\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_r) \in \mathbb{R}^r$. Then $h_1(\lambda)$ is a piecewise linear convex function [57]. A basis structure for MLP is a partition $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ of $[m]$, such that $|\mathcal{B}| = r$. With each basic feasible solution of MLP we associate a basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U})$, where $\mathcal{L}$ is the index set of nonbasic variables at the lower bound 0, $\mathcal{U}$ is the index set of nonbasic variables at the upper bound 1 and $\mathcal{B}$ is the index set of basic variables. Given a dual feasible basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U})$, the set of values $\lambda \in \mathbb{R}^r$ for which the corresponding basic solution is optimal is called the characteristic region of $(\mathcal{B}, \mathcal{L}, \mathcal{U})$. Since $h_1(\lambda)$ is piecewise linear convex, $h_1(\lambda)$ is linear if λ is restricted to a characteristic region associated with a dual feasible basic structure $(\mathcal{B}, \mathcal{L}, \mathcal{U})$. We call the extreme points of the characteristic regions of $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ breakpoints and denote the set of these breakpoints by B_1 and define $x(\lambda)$ as the optimal basic feasible solution of $h_1(\lambda)$ at each $\lambda \in B_1$. By the results of Punnen et al. [107, Theorem 3] we know that $x(\lambda) \in \{0,1\}^m$. Let $y(\lambda) \in \mathcal{F}_2$ be an optimal solution to our instance of COPIC when x is fixed at $x(\lambda)$. In this case COPIC reduces to

$$\min \left\{ \left(\sum_{p=1}^{r}(a_p^T x(\lambda)) b_p^T + d^T \right) y \colon y \in \mathcal{F}_2 \right\}$$

which is an LCOP instance $(\mathcal{F}_2, f)$, with $f = \sum_{p=1}^{r}(a_p^T x(\lambda)) b_p + d$. The running time needed to calculate the cost vector f is bounded by $O(\max\{rm, rn\})$. This allows us to calculate $y(\lambda)$ in $O(\max\{rm, rn, T(\mathcal{F}_2)\})$ time, using an $T(\mathcal{F}_2)$-time algorithm for the LCOP instance $(\mathcal{F}_2, f)$, for each $\lambda \in B_1$.

Theorem 4.3. *There is an optimal solution to the instance* $(2^{[m]}, \mathcal{F}_2, Q, c, d)$ *of* COPIC *with* $\mathrm{rk}(Q) = r$ *amongst the solutions* $\{(x(\lambda), y(\lambda))\colon \lambda \in B_1\}$.

Proof. Rank r COPIC is equivalent to solving the bilinear program

$$\begin{aligned} \min \ & \sum_{p=1}^{r} \lambda_p (b_p^T y) + c^T x + d^T y \\ \text{s.t.} \ & a_p^T x = \lambda_p \quad p = 1, 2, \ldots, r \\ & x \in [0,1]^m, y \in \mathcal{F}_2, \lambda \in \mathbb{R}^r. \end{aligned}$$

Let $h(\lambda)$ be the optimal value if λ is fixed, then we can decompose $h(\lambda)$ into $h(\lambda) = h_1(\lambda) + h_2(\lambda)$, where

$$h_2(\lambda) = \min \left\{ \sum_{p=1}^{r} \lambda_p (b_p^T y) + d^T y\colon y \in \mathcal{F}_2 \right\}.$$

So rank r COPIC can be reduced to solving $\min_{\lambda \in \mathbb{R}^r} h(\lambda)$.

We already argued above that $h_1(\lambda)$ is a piecewise linear convex function in λ. Using the fact that $h_2(\lambda)$ is the pointwise minimum of linear functions, we obtain that $h_2(\lambda)$ is a piecewise linear concave function in λ [18]. This implies that $h_1(\lambda)$ is linear, if λ is restricted to any characteristic region of $h_1(\lambda)$ and thus $h(\lambda)$ is concave on each of these regions. This implies that the minimum of $h(\lambda)$ is attained at a breakpoint of $h_1(\lambda)$, which implies the result since B_1 is defined as the set of these breakpoints. □

Analogously to Punnen et al. [107], we can solve rank r COPIC based on Theorem 4.3 by generating the set of breakpoints B_1 and then computing the set $\{x(\lambda), y(\lambda))\colon \lambda \in B_1\}$ and selecting the best solution from there. Note, that it is not necessary to explicitly calculate B_1, since the set $\{x(\lambda)\colon \lambda \in B_1\}$ can be obtained without computing the corresponding values of λ in advance. For each dual-feasible and dual non-degenerate basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ we can generate 2^r basic feasible solutions corresponding to its extreme points by varying $\tau \in \{0,1\}^r$, giving the values of the basic variables, and setting the non-basic variables in accordance with $\mathcal{L}$ and $\mathcal{U}$ (for details see the proof of [107, Theorem 3]). For each of these basic feasible solutions x we can compute the corresponding y by solving an instance of LCOP. Below we give a high-level summary of our algorithm.

1. Let Γ be the set of all dual feasible basis structures $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ for the corresponding MLP.

2. Compute the set $\bar{S}$ of all optimal basic feasible solutions corresponding to the extreme points of the characteristic region of a dual feasible basis structure $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ in Γ.

3. For each $x \in \bar{S}$ compute the best $y \in \mathcal{F}_2$ by solving the LCOP instance $(\mathcal{F}_2, f)$, with $f = \sum_{p=1}^{r} (a_p^T x) b_p + d$.

4. Output the best pair (x, y) with minimum total cost found in the last step.

By the arguments above it follows that this algorithm always finds an optimal solution. There are $\binom{m}{r}$ choices for $\mathcal{B}$ and each of them gives a unique allocation of non-basic variables to $\mathcal{L}$ and $\mathcal{U}$ (uniqueness following from non-degeneracy which can be achieved by appropriate perturbation of the cost vector). The basis inverse can be obtained in $O(r^3)$ time and given this inverse $\mathcal{L}$ and $\mathcal{U}$ can be identified in $O(mr^3)$ time, such that $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ is dual feasible. This implies that the set of dual feasible basis structures is bounded by $\binom{m}{r}$ and can be calculated in $O(\binom{m}{r}(r^3 + mr^2))$ time. By [107, Theorem 3], we know that the number of extreme points associated with $(\mathcal{B}, \mathcal{L}, \mathcal{U})$ is bounded by 2^r and we argued above how they can be generated. This allows us to compute $\bar{S}$ in $O(\binom{m}{r}2^r m)$ time (step 1 and 2). Fixing $x \in \bar{S}$, the best corresponding solution y can be computed in $O(\max\{rm, rn, \mathrm{T}(\mathcal{F}_2)\})$ time (step 2). Summarizing this gives the following result.

Theorem 4.4. *If* $\mathrm{rk}(Q) = r$ *and there is a* $\mathrm{T}(\mathcal{F}_2)$*-time algorithm for* LCOP *instances* $(\mathcal{F}_2, f)$ *for every* $f \in \mathbb{R}^n$*, then the* COPIC *instance* $(2^{[m]}, \mathcal{F}_2, Q, c, d)$ *can be solved in* $O(\binom{m}{r}2^r \max\{rm, rn, \mathrm{T}(\mathcal{F}_2)\})$ *time.*

Remark. *An identical approach works for sets of feasible solutions* $\mathcal{F}_1$*, for which we can solve the linear cost minimization problem, extended by a constant number of side constraints of the form* $a_p^T x = \lambda_p$ *and the number of breakpoints (in* λ*) is polynomially bounded. But, this does not help for most non-continuous problems, because already for the bases of a uniform matroid this corresponds to a partition problem.*

We can now use Theorem 4.3 to obtain approximation algorithms for rank r COPIC based on approximation algorithms for the linear problem with feasible solutions in $\mathcal{F}_2$.

Theorem 4.5. COPIC *with instances restricted to the form* $(2^{[m]}, \mathcal{F}_2, Q, c, d)$*, such that the* LCOP *instance* $(\mathcal{F}_2, f)$ *admits a* $\mathrm{T}(\mathcal{F}_2)$ *time* α*-approximation algorithm for arbitrary* $f \in \mathbb{R}^n$*, has a* $O(\binom{m}{r}2^r \max\{rm, rn, \mathrm{T}(\mathcal{F}_2)\})$ *time* α*-approximation algorithm.*

Proof. By Theorem 4.3 there exists an optimal solution

$$(x^*, y^*) = (x(\lambda^*), y(\lambda^*)) \in \{(x(\lambda), y(\lambda))\colon \lambda \in B_1\}.$$

By the method above we will in some iteration find x^* as one of the extreme points of a characteristic region of $h_1(\lambda)$. Then calculating y^* is equivalent to solving the LCOP instance $(\mathcal{F}_2, f)$ with $f = \sum_{p=1}^{r}(a_p^T x^*)b_p + d$. Instead of solving this problem to optimality we can use our α-approximation algorithm and obtain a solution $\tilde{y} \in \mathcal{F}_2$ such that

$$\tilde{h}_2 := \sum_{p=1}^{r}(a_p^T x^*)(b_p^T \tilde{y}) + d^T \tilde{y} \leq \alpha h_2(\lambda^*).$$

Altogether for our found solution $(x^*, \tilde{y})$ we obtain a bound on the objective value given by

$$h_1(\lambda^*) + \tilde{h}_2 \leq h_1(\lambda^*) + \alpha h_2(\lambda^*) \leq \alpha h(\lambda^*).$$

□

4.3.2 General Fixed Rank COPIC via Multi-Parametric Optimization

To solve fixed rank COPIC when both sets of feasible solutions $\mathcal{F}_1$ and $\mathcal{F}_2$ are constrained, we apply different methods from parametric optimization. Since in many cases additional linear constraints of the form $a_p^T x = \lambda_p$ imply NP-hardness, we cannot follow an identical approach as above. Instead, we analyze and solve multi-parametric objective versions for both sets of feasible solutions directly. Given linear cost vectors $a_1, a_2, \dots a_r \in \mathbb{R}^n$ and $c \in \mathbb{R}^n$ in addition to a set of feasible solutions $\mathcal{F} \subseteq \{0,1\}^n$, the problem of finding optimal solutions to

$$\min \left\{ \sum_{p=1}^{r} \mu_p (a_p^T x) + c^T x \colon x \in \mathcal{F} \right\}$$

for all possible values of $\mu \in \mathbb{R}^r$ is called multi-parametric linear optimization over $\mathcal{F}$. In this section the number of vectors a will always be fixed to r. For every fixed $\mu \in \mathbb{R}^r$ this is equivalent to solving an LCOP instance of $(\mathcal{F}, h)$ for $h = \sum_{p=1}^{r} \mu_p a_p + c$. We denote this problem by MPLCOP instance $(\mathcal{F}, a, c)(\mu)$.

It is well known that the optimal value of the instance $(\mathcal{F}, a, c)(\mu)$ of MPLCOP is a piecewise-linear concave function in μ on $\mathbb{R}^r$. For such a function the parameter space $\mathbb{R}^r$ can be partitioned into regions $M_1, M_2, \dots, M_l$, such that in each of these regions the optimal objective value is linear in μ and for each $i = 1, 2, \dots, l$ there exists a solution $x_i \in \mathcal{F}$ that achieves this value on the whole region M_i. The smallest needed number l of such regions is called the parametric complexity of the MPLCOP instances $(\mathcal{F}, a, c)$. Bökler and Mutzel [15] showed that there is an output-sensitive algorithm for MPLCOP instances $(\mathcal{F}, a, c)(\lambda)$ to obtain all the solutions $x_1, x_2, \dots, x_l$ with running time $O(\text{poly}(n, m, l^r))$, if the LCOP instance $(\mathcal{F}, h)$ can be solved in polynomial time.

Given an instance of fixed rank COPIC

$$\begin{aligned} \min \ & \sum_{p=1}^{r} (a_p^T x)(b_p^T y) + c^T x + d^T y \\ \text{s.t.} \ & x \in \mathcal{F}_1', \ y \in \mathcal{F}_2' \end{aligned}$$

and its optimal solution $(x^*, y^*) \in \mathcal{F}_1' \times \mathcal{F}_2'$, we observe that x^* is an optimal solution to the MPLCOP instance $(\mathcal{F}_1', a, c)(\mu^*)$ for $\mu_p^* = b_p^T y^*$ and y^* is an optimal solution to MPLCOP instance $(\mathcal{F}_2', b, d)(\lambda^*)$ for $\lambda_p^* = a_p^T x^*$. This yields the following approach for solving such instances of COPIC:

1. Obtain optimal solutions $x_1, x_2, \dots, x_{l_1}$ for all possible parameter values μ of MPLCOP instances $(\mathcal{F}_1', a, c)(\mu)$ and $y_1, y_2, \dots, y_{l_2}$ for all possible parameter values λ of MPLCOP instances $(\mathcal{F}_2', b, d)(\lambda)$.

2. Calculate the corresponding parameter values $\mu^{(1)}, \mu^{(2)}, \dots, \mu^{(l_1)}$ for $(\mathcal{F}_1', a, c)(\mu)$ and $\lambda^{(1)}, \lambda^{(2)}, \dots, \lambda^{(l_1)}$ for $(\mathcal{F}_2', b, d)(\lambda)$ as $\lambda_p^{(i)} = a_p^T x_i$ and $\mu_p^{(j)} = b_p^T y_j$.

3. For each pair (x_i, y_j) check if x_i is optimal for the LCOP instance $(\mathcal{F}_1', a, c)(\mu^{(j)})$ and y_j is optimal for LCOP instance $(\mathcal{F}_2', b, d)(\lambda^{(i)})$.

4. Among all the pairs that fulfill conditions in step 3, take the one with minimum objective value for our instance of COPIC.

To guarantee that this method finds the optimal solution (x^*, y^*) the two given instances of MPLCOP must be non-degenerate. This can be guaranteed by appropriate perturbations of the cost vectors. Based on the algorithm of Bökler and Mutzel [15] we obtain the following result.

Theorem 4.6. *Let l_1, l_2 be the parametric complexity of* MPLCOP *instances $(\mathcal{F}_1', a, c)$ and $(\mathcal{F}_2', b, d)$ respectively, and $\mathrm{rk}(Q) = r$ is a constant. If both* LCOP *instances $(\mathcal{F}_1, h)$ and $(\mathcal{F}_2, h)$ can be solved in polynomial time for arbitrary linear cost vectors h, then* COPIC *instances $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ can be solved in $O(\mathrm{poly}(n, m, l_1^r, l_2^r))$ time.*

Table 4.1 summarizes known results about the parametric complexity of MPLCOP for different sets of feasible solutions.

feasible solutions $\mathcal{F}$	paramteric compl. l	references
$\mathcal{B}(\mathcal{M}_1)$	$O(n^{2r-2})$	[60]
$\mathcal{CUT}(G)$	$O(n^{r+1})$	[78]
$\mathcal{P}_{s,t}(G)$	subexp. lower bound	[29, 70]
$\mathcal{PM}(G)$	subexp. lower bound	[29, 70]
arbitrary	$O(n^{2r}\phi^r), \phi \geq 1$ (smoothed)	[22]

Table 4.1: Summary of known results about the parametric complexity of MPLCOP for different sets of feasible solutions with instance size n and rank r.

In addition to the exact results above we can obtain a FPTAS based on the results of Mittal and Schulz [100], for a restricted class of objective functions.

Theorem 4.7 (Mittal and Schulz [100])**.** COPIC *instances $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ admit a FPTAS, if $\mathrm{rk}(Q) = r$ is fixed, the sets $\mathcal{F}_1, \mathcal{F}_2$ can be represented as polytopes of polynomial size or polytopes with a polynomial time separation oracle and $c^T x > 0, d^T y > 0$ and $a_p^T x > 0, b_p^T y > 0$ for $p = 1, 2, \ldots, r$.*

4.4 Diagonal Interaction Matrix

In this section we analyze the special case of COPIC, referred to as *diagonal COPIC*, where for a given vector $a \in \mathbb{R}^n$ the matrix $Q = (q_{ij})$ is given as the diagonal $n \times n$ matrix $q_{ii} = a_i$. This results in finding solutions $S_1 \in \mathcal{F}_1 \subseteq \{0,1\}^n$ and $S_2 \in \mathcal{F}_2 \subseteq \{0,1\}^n$ that minimize the objective function

$$f(S_1, S_2) = \sum_{i \in S_1 \cap S_2} a_i + \sum_{i \in S_1} c_i + \sum_{j \in S_2} d_j.$$

Such COPIC instances are denoted by $(\mathcal{F}_1, \mathcal{F}_2, \mathrm{diag}(a), c, d)$.

Already this very restricted version of COPIC includes many well-studied problems of combinatorial optimization. For example, problems that ask for two disjoint combinatorial structures among an element set can all be handled by solving COPIC with identity interaction matrix $Q = I$ and $c = d = 0$. This includes the disjoint spanning tree problem [110], disjoint matroid base problem [56], disjoint path problems [50, 116], disjoint matchings problem [54] and many others. Bernáth and Király [11] analyzed the computational complexity of many combinations of different packing, covering and partitioning problems on graphs and matroids. It is easy to model all of these problems as instances of diagonal COPIC. The hardness results for packing problems in this chapter directly imply NP-hardness results for diagonal COPIC with $Q = I$ and $c = d = 0$ for several classes of problems. In this section we further investigate complexity of diagonal COPIC. Some results investigated in this section are summarized in Table 4.2.

$\mathcal{F}_1 \setminus \mathcal{F}_2$	$2^{[n]}$	$\mathcal{B}(\mathcal{U}_n^{k_2})$	$\mathcal{B}(\mathcal{M}_2)$	$\mathcal{PM}(G)$	$\mathcal{P}_{s_2,t_2}(G)$
$2^{[n]}$	$O(n)$	**P**	**P**	**P**	**P**
$\mathcal{B}(\mathcal{U}_n^{k_1})$		**P**	**P** $_{(c\,=\,d\,=\,0)}$	open	open
$\mathcal{B}(\mathcal{M}_1)$			**P** $_{(c\,=\,d\,=\,0)}$	open	NP-hard
$\mathcal{PM}(G)$				NP-hard	open
$\mathcal{P}_{s_1,t_1}(G)$					**Table 4.3**

Table 4.2: Summary of complexity results for COPIC with a diagonal matrix. Bold entries correspond to new reductions obtained in Section 4.4.

4.4.1 Unconstrained Feasible Sets

We start by considering diagonal COPIC with one unconstrained feasibility set.

Theorem 4.8. *The* COPIC *instance* $(\mathcal{F}, 2^{[n]}, \mathrm{diag}(a), c, d)$ *can be solved by solving the* LCOP *instance* $(\mathcal{F}, f)$*, where* $f_i = \min\{c_i + d_i + a_i, c_i\} - \min\{d_i, 0\}$ *for each* $i \in [n]$.

Proof. For any $S_1 \in \mathcal{F}_1$ fixed we can obtain $S_2 \in \mathrm{argmin}_{S \subseteq [n]} f(S_1, S_2)$ by the following case distincition for every $i \in [n]$. If $i \in S_1$ then $i \in S_2$ if $c_i + d_i + a_i \leq c_i$. If $i \notin S_1$, then $i \in S_2$ if $d_i \leq 0$.

By setting $f_1(i) = \min\{c_i + d_i + a_i, c_i\}$ and $f_2(i) = \min\{d_i, 0\}$, we have that

$$\min_{S_2 \subseteq [n]} f(S_1, S_2) = f_1(S_1) + f_2(E \setminus S_1) = f_1(S_1) + f_2(E) - f_2(S_1).$$

By observing that $f_2(E)$ is constant, we obtain an optimal S_1 four our instance of COPIC by solving $\min_{S_1 \subseteq [m]}(f_1 - f_2)(S_1)$. The corresponding optimal S_2 can be obtained by the case distinction given above. □

This directly implies the following corollary for unconstrained diagonal COPIC.

Corollary 4.9. *The* COPIC *instance* $(2^{[n]}, 2^{[n]}, \mathrm{diag}(a), c, d)$ *can be solved in linear time.*

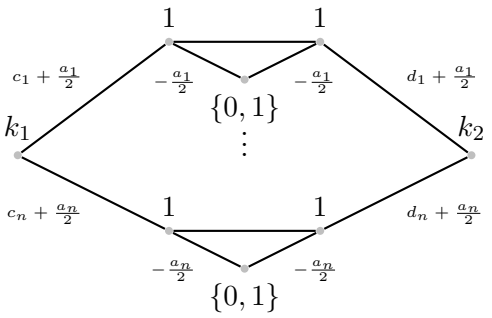


Figure 4.1: Illustration for the proof of Theorem 4.10.

In the PhD thesis of Sripratak a generalization of Corollary 4.9 to matrices of bandwidth $O(\log n)$ is shown. [114, Theorem 2.11]

4.4.2 Uniform and Partition Matroids

In the following two subsections we investigate diagonal COPIC where $\mathcal{F}_1$ and $\mathcal{F}_2$ correspond to bases of different types of matroids. For bases of uniform and partition matroids the main insight is that we can solve our problem in polynomial time using matching algorithms.

Given a graph $G = (V, E)$ and a function $b\colon V \to 2^{\mathbb{N}}$ an edge set $M \subseteq E$ is a b-factor, if $|M \cap \delta(v)| \in b(v)$ for each $v \in V$, where $\delta(v)$ is the set of all edges incident with v. If $b(v) = \{k\}$ for some integer $k \in \mathbb{N}$ we simply write $b(v) = k$. Given an additional cost function $c\colon E \to \mathbb{R}$ a minimum cost b-factor can be found in polynomial time, if all the b-values $b(v)$ are sequences of consecutive integers $[b_1; b_2] = \{b_1, b_1 + 1, \ldots, b_2\}$ (see [96, Section 10.2]).

Theorem 4.10. COPIC *can be solved in polynomial time, if instances are restricted to the form* $(\mathcal{B}(\mathcal{U}_n^{k_1}), \mathcal{B}(\mathcal{U}_n^{k_2}), \mathrm{diag}(a), c, d)$.

Proof. We reduce to an instance of the minimum cost b-factor problem on a graph G (see Figure 4.1). To achieve this, we introduce two special vertices x and y with $b(x) = k_1$ and $b(y) = k_2$ and another $3n$ vertices i_x, i_y and i_m for $i = 1, 2, \ldots, n$, i.e., for each element of the ground set of the two matroids. We set $b(i_x) = b(i_y) = 1$ and $b(i_m) = \{0, 1\}$. The k_1 vertices matched with x and k_2 vertices matched with y correspond to the sets S_1 and S_2, respectively.

We introduce edges $\{x, i_x\}$ with cost $c_i + \frac{a_i}{2}$ and $\{y, i_y\}$ with cost $d_i + \frac{a_i}{2}$. We also connect $\{i_x, i_y\}$ with edges of cost 0 and $\{i_x, i_m\}$ and $\{i_x, i_m\}$ both with cost $-\frac{a_i}{2}$.

It is easy to see that there is a one-to-one mapping between feasible solutions of the given diagonal COPIC and this instance of the b-factor problem, and moreover, the corresponding costs are the same. □

This reduction technique can be used in a straightforward way to also obtain a polynomial time algorithm also for the more general cases of partition matroids and generalized

partition matroids (see [51] for a definition).

4.4.3 Matroid Bases as Feasible Sets

Another problem of great interest is the case when $\mathcal{F}_1, \mathcal{F}_2$ are sets of spanning trees of a graph, especially if the underlying graphs are isomorphic. A generalization of this problem is the case when $\mathcal{F}_i = \mathcal{B}(\mathcal{M}_i)$ are the sets of bases of (not necessarily isomorphic) matroids $\mathcal{M}_1, \mathcal{M}_2$. In this section we assume familiarity with matroids and refer the reader to Oxley [103] for further definitions, results and notations. We assume in thit sections that there exist efficient oracles to solve the static-base circuit problem. This means that for both matroids $\mathcal{M}_i$, $i = 1, 2$, independent set S and element $e \notin S$, we can efficiently decide if $S \cup \{e\}$ is independent in $\mathcal{M}_i$, and if not, output all elements in $C(e, S)$, the unique cycle contained in $S \cup \{e\}$ of the matroid $\mathcal{M}_i$.

We focus on the case without linear costs, i.e. $c \equiv d \equiv 0$. So the problem we are interested in is, given a ground set $E = [n]$ and a cost vector $a \in \mathbb{R}^n$, to minimize the objective function

$$f(B_1, B_2) = \sum_{i \in B_1 \cap B_2} a_i$$

under the restrictions that $B_1 \in \mathcal{B}(\mathcal{M}_1), B_2 \in \mathcal{B}(\mathcal{M}_2)$ for two given matroids $\mathcal{M}_1, \mathcal{M}_2$ over the ground set E.

The case where for each element $i \in B_1 \cap B_2$ we pay a non-negative cost $a_i \geq 0$ was already studied in the algorithmic game theory literature. It is equivalent to computing the socially optimal state of a two player matroid congestion game. Ackermann et al. [2] show that this problem can be solved in polynomial time, by an reduction to the minimum cost disjoint matroid base problem, which can be solved with the algorithm of Gabow and Westermann [56].

The case of arbitrary real costs $a_e \in \mathbb{R}$ can also be handled. This is not included in the algorithmic game theory literature, since in that context a positive impact of congestion (i.e. $a_i < 0$) does not make sense. First, we find a set $B \in \mathcal{I}(\mathcal{M}_1) \cap \mathcal{I}(\mathcal{M}_2)$ of minimum cost and we contract this set. For all edges $e \in E \setminus B$ with $a_e < 0$ it holds that $B + e \notin \mathcal{I}(\mathcal{M}_1) \cap \mathcal{I}(\mathcal{M}_2)$, or we could improve the solution, so these elements can never be in the intersection of a feasible solution together with B. Hence we can run the algorithm for $a \geq 0$ on the remaining instance. The optimality of this approach follows from the following lemma.

Lemma 4.11. *Let B be an element of $\mathcal{I}(\mathcal{M}_1) \cap \mathcal{I}(\mathcal{M}_2)$ with minimum cost $a(B) := \sum_{i \in B} a_i$, and B_1, B_2 be two bases. Then B_1, B_2 can be transformed into two new bases $\tilde{B}_1, \tilde{B}_2$ such that $B \subseteq \tilde{B}_1 \cap \tilde{B}_2$ and*

$$a(\tilde{B}_1 \cap \tilde{B}_2) \leq a(B_1 \cap B_2).$$

Proof. Since B_i is a basis there exist sets $Q_i \subseteq B_i$ with $|Q_i| = |B|$ such that $\tilde{B}_i = (B_i \setminus Q_i) \cup B$ is a basis for both $i = 1, 2$. It now holds that $\tilde{B}_1 \cap \tilde{B}_2 = B \cup ((B_1 \setminus Q_1) \cap (B_2 \setminus Q_2))$ and the union above is disjoint because if $e \in B \cap B_i$ it follows that $e \in Q_i$, else $\tilde{B}_i$ could

not be a basis for both $i = 1, 2$. This implies that $a(\tilde{B}_1 \cap \tilde{B}_2) = a(B) + a(((B_1 \setminus Q_1) \cap (B_2 \setminus Q_2)))$.

Now also $B_1 \cap B_2$ can be written as the disjoint union of B_1 and B_2, $B_1 \cap B_2 = ((B_1 \setminus Q_2) \cap (B_2 \setminus Q_2)) \cup (Q_1 \cap (B_2 \setminus Q_2)) \cup (B_1 \cap Q_2)$, and we obtain $a(B_1 \cap B_2) = a(((B_1 \setminus Q_2) \cap (B_2 \setminus Q_2))) + a((Q_1 \cap (B_2 \setminus Q_2)) \cup (B_1 \cap Q_2))$. We know that $(Q_1 \cap (B_2 \setminus Q_2)) \cup (B_1 \cap Q_2) \in \mathcal{I}(\mathcal{M}_1) \cap \mathcal{I}(\mathcal{M}_2)$, since it is a subset of $B_1 \cap B_2$, which implies $a(B) \leq a((Q_1 \cap (B_2 \setminus Q_2)) \cup (B_1 \cap Q_2))$. Using this our claim follows. □

By the approach above we obtain the following result.

Theorem 4.12. *A* COPIC *instance* $(\mathcal{B}(\mathcal{M}_1), \mathcal{B}(\mathcal{M}_2), \mathrm{diag}(a), 0, 0)$ *is solvable in polynomial time, for any two matroids* $\mathcal{M}_1, \mathcal{M}_2$ *and cost vector* $a \in \mathbb{R}^n$.

4.4.4 Pairs of Paths

In this section we summarize the special case when $\mathcal{F}_1$ and $\mathcal{F}_2$ correspond to the set of s_1-t_1- and s_2-t_2-paths in a graph or digraph. We will again look at the case where the graphs corresponding to $\mathcal{F}_1$ and $\mathcal{F}_2$ are identical. One must also make sure that there do not exist negative circles in the graph, else already optimizing over a linear cost function without interaction costs is NP-hard. To simplify the exposition we will focus on $Q, c, d \geq 0$.

We observe that this special case of COPIC is a generalization of different variants of edge-disjoint path problems and the computational complexity results from the edge-disjoint paths literature can be directly transfered using standard reduction techniques. Table 4.3 summarizes those results and the methods to obtain them.

It is important to differentiate between directed and undirected graphs, which is clear in the light of Proposition 4.13 and the known complexity results for the edge-disjoint paths problem.

Proposition 4.13. *Given a graph* G *and* $a > 0$*, an* COPIC *instance restricted to the form* $(\mathcal{P}_{s_1,t_1}(G), \mathcal{P}_{s_2,t_2}(G), \mathrm{diag}(a), 0, 0)$ *has a solution with objective value* 0*, if and only if there exist two edge-disjoint paths* s_i-t_i*-paths in* G.

It is well known that the edge-disjoint paths problem is polynomial time solvable for every constant number of paths in undirected graphs [109], but NP-hard already for 2 paths in directed graphs [49]. This imediatly yields the following result.

Corollary 4.14. *Given a directed graph* G*,* COPIC *with instances restricted to the form* $(\mathcal{P}_{s_1,t_1}(G), \mathcal{P}_{s_2,t_2}(G), \mathrm{diag}(a), 0, 0)$ *is NP-hard, even for* $a \equiv 1$.

We use the following results obtained by Eilam-Tzoreff [46] to further classify the complexity of our problem.

Theorem 4.15 (Eilam-Tzoreff [46])**.** *The undirected edge-disjoint two shortest paths problem is polynomial time solvable, even in the weighted case. On the other hand, the undirected two edge-disjoint one shortest paths problem is NP-hard.*

directedness	terminals	cost restrictions	complexity	method
directed	arbitrary	$Q = I, c = d = 0$	NP-hard	dEDP
directed	common	$Q = \text{diag}(\infty)$	NP-hard	dEDP
undirected	arbitrary	$Q = \text{diag}(\infty), d = 0$	NP-hard	uED2SP
undirected	arbitrary	$c = d = 0$	open	
undirected	common	$Q = \text{diag}(\infty)$	NP-hard	uED2SP
both	common	$c = d$	P	MCF

Table 4.3: Summary of the results for diagonal COPIC with paths as feasible solutions. The column *method* contains the problem we reduce from/to: dEDP... directed edge disjoint paths, uEDP2SP... undirected edge disjoint two shortest paths [46], MCF... minimum cost flow.

It is important to note that in the results of Eilam-Tzoreff, a shortest path always means a shortest path in the original graph, not a shortest path after removing the edges of the other disjoint path. This is the reason why using Theorem 4.15, we cannot conclude that COPIC with instances restricted to the form $(\mathcal{P}_{s_1,t_1}(G), \mathcal{P}_{s_2,t_2}(G), \text{diag}(\infty), c, c)$ is polynomial time solvable, since in our model we cannot enforce two shortest paths of the original graph. If $c = d = 1$ and $Q = \text{diag}(\infty)$ Björklund and Husfeldt [13] showed in 2014 how to solve the problem using a polynomial time Monte Carlo algorithm. The existence of a deterministic polynomial time algorithm is still unknown and a long-standing open problem.

Nevertheless, it is possible to use the hardness results of Eilam-Tzoreff [46] to show that for general costs $c, d \geq 0$ the problem is NP-hard.

Corollary 4.16. *Given an undirected graph G,* COPIC *with instances restricted to the form $(\mathcal{P}_{s_1,t_1}(G), \mathcal{P}_{s_2,t_2}(G), \text{diag}(\infty), c, 0)$ is NP-hard for $c \geq 0$.*

Proof. Using a polynomial time algorithm for COPIC we can determine, if the two edge-disjoint one shortest paths problem has a solution. Just run the algorithm and check if the objective value equals the length of a shortest s_1-t_1-path in the given graph. □

This covers the case if $s_1 \neq s_2$ and $t_1 \neq t_2$. From the edge-disjoint path literature we know that the problem becomes easier, if one assumes a common source s and a common sink t for all the paths. We can classify the complexity of this case for our problem, using the following results.

Theorem 4.17. *Given a graph or digraph G, a* COPIC *instance restricted to the form $(\mathcal{P}_{s,t}(G), \mathcal{P}_{s,t}(G), \text{diag}(a), c, c)$ is solvable in polynomial time, for cost vectors $a, c \geq 0$.*

Proof. We reduce to a minimum cost flow problem. Set $b(s) = 2$ and $b(t) = -2$ and double each edge/arc $e \in E$ to two versions e_1, e_2 with $\tilde{c}_{e_1} = c_e$ and $\tilde{c}_{e_2} = a_e + c_e$. Now a minimum cost flow in this network will be integral and can be decomposed into two path flows, each sending one unit from s to t. The cost of the flow corresponds to the cost of these two paths in our problem. □

Theorem 4.18. *Given a graph or digraph G,* COPIC *with instances restricted to the form $(\mathcal{P}_{s,t}(G), \mathcal{P}_{s,t}(G), \mathrm{diag}(\infty), c, d)$ is NP-hard.*

Proof. For digraphs the statement follows from a reduction from directed two disjoint paths. Given such an instance we introduce the new terminals s and t and add arcs (s, s_1), (s, s_2), (t_1, t), (t_2, t). We use $Q = \mathrm{diag}(\infty)$ and as linear costs $c_{(s,s_1)} = c_{(t_1,t)} = d_{(s,s_2)} = d_{(t_2,t)} = 0$ and $c_{(s,s_2)} = c_{(t_2,t)} = d_{(s,s_1)} = d_{(t_1,t)} = \infty$ and $c_e = d_e = 0$ for all other edges. This enforces that paths S_i are s_i-t_i-paths and the diagonal matrix with infinite entries ensures disjointness.

In the undirected case we apply the same construction as above but using the undirected two edge-disjoint one shortest paths problem. To solve the decision problem analyzed by Eilam-Tzoreff [46], we create COPIC with $c_e = 1$ and $d_e = 0$ for all the edges in the original network to enforce that S_1 is a shortest path. After finding a finite cost solution to this problem we check if the length of S_1 is equal to the length of a shortest s_1-t_1-path in G. □

4.5 Linearizable Instances

In this section we explore for which cost matrices COPIC leads to an equivalent problem where there is essentially no interaction between two structures of COPIC.

More precisely, we say that an interaction cost matrix Q of a COPIC is *linearizable*, if there exist vectors $a = (a_i)$ and $b = (b_i)$ such that for all $S_1 \in \mathcal{F}_1$ and $S_2 \in \mathcal{F}_2$

$$\sum_{i \in S_1} \sum_{j \in S_2} q_{ij} = \sum_{i \in S_1} a_i + \sum_{j \in S_2} b_j$$

holds. In that case we say that the pair of vectors a and b together is a *linearization* of Q.

Note that for a COPIC instance $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$, $f(S_1, S_2) = \sum_{i \in S_1} \bar{a}_i + \sum_{j \in S_2} \bar{b}_j$ for some $\bar{a} = (\bar{a}_i)$, $\bar{b} = (\bar{b}_i)$ and all $S_1 \in \mathcal{F}_1$, $S_2 \in \mathcal{F}_2$, if and only if Q is linearizable. Hence, we extend our notion of linearizability and say that a COPIC instance $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ is linearizable if and only if Q is linearizable. Our aim is to characterize all linearizable instances of COPIC, with respect to given solution sets $\mathcal{F}_1$ and $\mathcal{F}_2$.

Linearizable instances have been studied by various authors for the case of quadratic assignment problem [30, 76, 106], quadratic spanning tree problem [39] and bilinear assignment problem [40]. Here we generalize the ideas from [40] and suggest an approach for finding a characterization of linearizable instances of COPIC's.

An interaction cost matrix Q of a COPIC has *constant objective property with respect to* $\mathcal{F}_1$ if for every $j \in [n]$ there exist a constant $K_j^{(1)}$, so that

$$\sum_{i \in S_1} q_{ij} = K_j^{(1)} \qquad \text{for all} \quad S_1 \in \mathcal{F}_1.$$

Similarly, Q has *constant objective property with respect to* $\mathcal{F}_2$ if for every $i \in [m]$ there

exist a constant $K_i^{(2)}$, so that

$$\sum_{j \in S_2} q_{ij} = K_i^{(2)} \qquad \text{for all } \; S_2 \in \mathcal{F}_2.$$

For $\mathcal{F}_i$, $i = 1, 2$, let $\mathrm{CVP}_i(\mathcal{F}_i)$ be the vector space of all matrices with constant objective property with respect to $\mathcal{F}_i$.

Combinatorial optimization problems with constant objective property have been studied by various authors [10, 24, 36, 81].

Let $\mathrm{CVP}_1(\mathcal{F}_1) + \mathrm{CVP}_2(\mathcal{F}_2)$ be the vector space of all interaction matrices $Q = (q_{ij})$ of COPIC, such that $q_{ij} = a_{ij} + b_{ij}$ for all i, j and for some $A = (a_{ij}) \in \mathrm{CVP}_1(\mathcal{F}_1)$ and $B = (b_{ij}) \in \mathrm{CVP}_2(\mathcal{F}_2)$.

Lemma 4.19 (Sufficient conditions)**.** *If the interaction cost matrix Q of a* COPIC *instance $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ is an element of $CVP_1(\mathcal{F}_1) + CVP_2(\mathcal{F}_2)$, then Q is linearizable.*

Proof. Let Q be of the form $Q = E + F$, where $E = (e_{ij}) \in \mathrm{CVP}_1(\mathcal{F}_1)$ and $F = (f_{ij}) \in \mathrm{CVP}_2(\mathcal{F}_2)$. Then

$$\sum_{i \in S_1} \sum_{j \in S_2} q_{ij} = \sum_{i \in S_1} \sum_{j \in S_2} (e_{ij} + f_{ij}) =$$
$$\sum_{j \in S_2} \left(\sum_{i \in S_1} e_{ij} \right) + \sum_{i \in S_1} \left(\sum_{j \in S_2} f_{ij} \right) = \sum_{j \in S_2} K_j^{(1)} + \sum_{i \in S_1} K_i^{(2)}.$$

Hence Q is linearizable, and $a = (a_i)$, $b = (b_j)$ with $a_i = K_i^{(2)}$, $b_j = K_j^{(1)}$ is a linearization of Q. □

Now we show that the opposite direction is also true, provided some additional conditions are satisfied. In fact, these additional conditions are satisfied for many well studied combinatorial optimization problems.

Lemma 4.20 (Necessary conditions)**.** *Let $\mathcal{F}_1 \subseteq 2^{[m]}$ and $\mathcal{F}_2 \subseteq 2^{[n]}$ be such that:*

(i) There exist an m vector $a = (a_i)$, an n vector $b = (b_j)$ and two non-zero constants K_a, K_b, such that

$$\sum_{i \in S_1} a_i = K_a \;\; \forall S_1 \in \mathcal{F}_1 \quad \textit{and} \quad \sum_{j \in S_2} b_j = K_b \;\; \forall S_2 \in \mathcal{F}_2.$$

(ii) If an $m \times n$ matrix $\bar{Q} = (\bar{q}_{ij})$ is such that $\sum_{i \in S_1} \sum_{j \in S_2} \bar{q}_{ij} = 0$ for all $S_1 \in \mathcal{F}_1$, $S_2 \in \mathcal{F}_2$, then $\bar{Q} \in CVP_1(\mathcal{F}_1) + CVP_2(\mathcal{F}_2)$.

If an instance $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ of COPIC *is linearizable, then it holds that $Q \in CVP_1(\mathcal{F}_1) + CVP_2(\mathcal{F}_2)$.*

Proof. Assume that the conditions (i) and (ii) of Lemma 4.20 are satisfied, and that Q is linearizable. We will show that $Q \in \mathrm{CVP}_1(\mathcal{F}_1) + \mathrm{CVP}_2(\mathcal{F}_2)$ by reconstructing the proof of Lemma 4.19 in reverse direction.

Since Q is linearizable, there exist $a = (a_i)$ and $b = (b_j)$ such that

$$\sum_{i \in S_1} \sum_{j \in S_2} q_{ij} = \sum_{i \in S_1} a_i + \sum_{j \in S_2} b_j \quad \forall S_1 \in \mathcal{F}_i, S_2 \in \mathcal{F}_2. \tag{4.5}$$

Note that from (i) it follows that there exist matrices $\hat{E} = (\hat{e}_{ij}) \in \mathrm{CVP}_1(\mathcal{F}_1)$ and $\hat{F} = (\hat{f}_{ij}) \in \mathrm{CVP}_2(\mathcal{F}_2)$ such that

$$\sum_{j \in S_2} \hat{f}_{ij} = a_i \qquad \forall S_2 \in \mathcal{F}_2,\ i \in M, \tag{4.6}$$

$$\sum_{i \in S_i} \hat{e}_{ij} = b_j \qquad \forall S_1 \in \mathcal{F}_1,\ j \in N. \tag{4.7}$$

Using (4.6) and (4.7), we can rewrite (4.5) as

$$\sum_{i \in S_1} \sum_{j \in S_2} q_{ij} = \sum_{i \in S_1} \left(\sum_{j \in S_2} \hat{f}_{ij} \right) + \sum_{j \in S_2} \left(\sum_{i \in S_1} \hat{e}_{ij} \right) = \sum_{i \in S_1} \sum_{j \in S_2} \left(\hat{e}_{ij} + \hat{f}_{ij} \right) \tag{4.8}$$

for all $S_1 \in \mathcal{F}_1$, $S_2 \in \mathcal{F}_2$. Hence it follows that

$$\sum_{i \in S_1} \sum_{j \in S_2} \left(q_{ij} - (\hat{e}_{ij} + \hat{f}_{ij}) \right) = 0 \quad \forall S_1 \in \mathcal{F}_i, S_2 \in \mathcal{F}_2. \tag{4.9}$$

Now, from (ii) it follows that $Q - (\hat{E} + \hat{F}) = E + F$ for some $E \in \mathrm{CVP}_1(\mathcal{F}_1)$, $F \in \mathrm{CVP}_2(\mathcal{F}_2)$, and hence, $Q = (E + \hat{E}) + (F + \hat{F}) \in \mathrm{CVP}_1(\mathcal{F}_1) + \mathrm{CVP}_2(\mathcal{F}_2)$. □

From Lemma 4.19 and Lemma 4.20 it follows that $\mathrm{CVP}_1(\mathcal{F}_1) + \mathrm{CVP}_2(\mathcal{F}_2)$ is the set of all linearizable matrices, provided that the corresponding COPIC satisfies properties (i) and (ii) of Lemma 4.20.

In most cases, property (i) is straightforward to check. For example, it is true for all COPIC's for which elements of $\mathcal{F}_1$ and $\mathcal{F}_2$ are of fixed cardinality. If $\mathcal{F}_1$ and $\mathcal{F}_2$ are s-t paths in a graph, then again property (i) is satisfied, although feasible solutions are of different cardinality. Condition (i) is not satisfied for unconstrained solution sets, i.e., when $\mathcal{F}_1$ ($\mathcal{F}_2$) is $2^{[m]}$ ($2^{[n]}$).

Now we show how Lemma 4.19 and Lemma 4.20 can be used to characterize linearizable instances for some specific COPIC's. Note that, to simplifying notation, in the case of $\mathcal{F} = \mathcal{PM}(K_{m,m})$ we denote an element $e \in \mathcal{F}$ corresponding to an edge $\{i, j\} \in [m] \times [m]$ by ij, and denote the interaction cost of two such elements ij and kl by q_{ijkl}.

Theorem 4.21.

(i) A COPIC *instance* $(\mathcal{PM}(K_{m,m}), \mathcal{PM}(K_{n,n}), Q, c, d)$ *is linearizable if and only if there are some arrays* A, B, C, D *such that* $q_{ijk\ell} = a_{ijk} + b_{ij\ell} + c_{ik\ell} + d_{jk\ell}$.

(ii) A COPIC *instance* $(\mathcal{B}(\mathcal{M}(K_m)), \mathcal{B}(\mathcal{M}(K_n)), Q, c, d)$ *is linearizable if and only if there are some vectors* a, b *such that* $q_{ij} = a_i + b_j$.

(iii) A COPIC *instance* $(\mathcal{B}(\mathcal{U}_m^{k_1}), \mathcal{B}(\mathcal{U}_n^{k_2}), Q, c, d)$ *is linearizable if and only if there are some vectors* a, b *such that* $q_{ij} = a_i + b_j$.

(iv) A COPIC *instance* $(\mathcal{PM}(K_{m,m}), \mathcal{B}(\mathcal{M}(K_n)), Q, c, d)$ *is linearizable if and only if there are some arrays* A, B, C *such that* $q_{ijk} = a_{ij} + b_{ik} + c_{jk}$.

(v) A COPIC *instance* $(\mathcal{B}(\mathcal{M}(K_m)), \mathcal{B}(\mathcal{U}_n^k), Q, c, d)$ *is linearizable if and only if there are some vectors* a, b *such that* $q_{ij} = a_i + b_j$.

(vi) A COPIC *instance* $(\mathcal{PM}(K_{m,m}), \mathcal{B}(\mathcal{U}_n^s), Q, c, d)$ *is linearizable if and only if there are some arrays* A, B, C *such that* $q_{ijk} = a_{ij} + b_{ik} + c_{jk}$.

Proof. We present a complete proof for (iv), and indicate how other statements can be shown analogously.

In the case of COPIC instances $(\mathcal{PM}(K_{m,m}), \mathcal{B}(\mathcal{M}(K_n)), Q, c, d)$, the interaction costs are represented in a three-dimensional array Q, since for convenience we represent the cost vector of $\mathcal{F}_1 = \mathcal{PM}(K_{m,m})$ in two indices. It is well known that a linear assignment problem instance $R = (r_{ij})$ has the constant objective property if and only if $r_{ij} = s_i + t_j$, for some vectors s and t. Hence $\mathrm{CVP}_1(\mathcal{PM}(K_{m,m})) = \{A = (a_{ijk})\colon a_{ijk} = b_{ik} + c_{jk}$ for some $B = (b_{ij}), C = (c_{ij})\}$. A spanning tree problem on a complete graph has the constant objective property if and only if the cost vector is constant, therefore $\mathrm{CVP}_2(\mathcal{B}(\mathcal{M}(K_n))) = \{A = (a_{ijk})\colon a_{ijk} = b_{ij}$ for some $B = (b_{ij})\}$. Hence, Q is an element of $\mathrm{CVP}_1(\mathcal{PM}(K_{m,m})) + \mathrm{CVP}_2(\mathcal{B}(\mathcal{M}(K_n)))$ if and only if there are some A, B and C such that

$$q_{ijk} = a_{ij} + b_{ik} + c_{jk}. \tag{4.10}$$

Lemma 4.19 tells us that (4.10) is a sufficient condition for Q to be linearizable. To show that it is also a necessary condition, we just need to show that properties (i) and (ii) of Lemma 4.20 are true for COPIC instances $(\mathcal{PM}(K_{m,m}), \mathcal{B}(\mathcal{M}(K_n)), Q, c, d)$. (i) is obviously true, hence it remains to show that if Q is such that

$$\sum_{(i,j)\in S_1} \sum_{k\in S_2} q_{ijk} = 0 \text{ for all } S_1 \in \mathcal{PM}(K_{m,m}), S_2 \in \mathcal{B}(\mathcal{M}(K_n)),$$

then $Q \in \mathrm{CVP}_1(\mathcal{PM}(K_{m,m})) + \mathrm{CVP}_2(\mathcal{B}(\mathcal{M}(K_n)))$.

Let $i, j \in \{2, 3, \ldots, m\}$ be fixed, and let $S'_{PM}, S''_{PM} \in \mathcal{PM}(K_{m,m})$ be such that $S'_{PM} \setminus S''_{PM} = \{(1,1), (i,j)\}$ and $S''_{PM} \setminus S'_{PM} = \{(1,j), (i,1)\}$. Further, let $k \in \{2, 3, \ldots, n\}$ be fixed, and let $S'_{ST}, S''_{ST} \in \mathcal{B}(\mathcal{M}(K_n))$ be such that $S'_{ST} \setminus S''_{ST} = \{1\}$ and $S''_{ST} \setminus S'_{ST} = \{k\}$. Note that such $S'_{PM}, S''_{PM}, S'_{ST}, S''_{ST}$ exist for all $i, j \in \{2, 3, \ldots, m\}$, $k \in \{2, 3, \ldots, n\}$.

Let us assume that Q is of the form such that

$$\sum_{(i,j)\in S_1}\sum_{k\in S_2} q_{ijk} = 0 \text{for all } S_1 \in \mathcal{PM}(K_{m,m}), S_2 \in \mathcal{B}(\mathcal{M}(K_n)).$$

Then, in particular, we have that

$$\sum_{(i,j)\in S'_{PM}}\sum_{k\in S'_{ST}} q_{ijk} + \sum_{(i,j)\in S''_{PM}}\sum_{k\in S''_{ST}} q_{ijk} = \sum_{(i,j)\in S'_{PM}}\sum_{k\in S''_{ST}} q_{ijk} + \sum_{(i,j)\in S''_{PM}}\sum_{k\in S'_{ST}} q_{ijk}, \tag{4.11}$$

which, after cancellations, gives us

$$q_{111} + q_{ij1} + q_{1jk} + q_{i1k} = q_{11k} + q_{ijk} + q_{1j1} + q_{i11} \tag{4.12}$$

for all $i, j \in \{2, 3, \ldots, m\}$, $k \in \{2, 3, \ldots, n\}$. Note that (4.12) holds true even if i, j or k is equal to 1, since in that case everything chancels out. Therefore, q_{ijk} can be expressed as

$$q_{ijk} = a_{ij} + b_{ik} + c_{jk} \qquad \forall i, j \in [m],\ \forall k \in [n], \tag{4.13}$$

where

$$a_{ij} = q_{ij1} - \frac{1}{2}q_{1j1} - \frac{1}{2}q_{i11} + \frac{1}{3}q_{111},\ b_{ik} = q_{i1k} - \frac{1}{2}q_{11k} - \frac{1}{2}q_{i11} + \frac{1}{3}q_{111},$$

$$c_{jk} = q_{1jk} - \frac{1}{2}q_{11k} - \frac{1}{2}q_{1j1} + \frac{1}{3}q_{111},$$

i.e., $Q \in \mathrm{CVP}_1(\mathcal{PM}(K_{m,m})) + \mathrm{CVP}_2(\mathcal{B}(\mathcal{M}(K_n)))$, hence also (ii) of Lemma 4.20 is true. That proves statement (iv) of the theorem.

Statements (i) and (ii) of the theorem can be proved by considering equation (4.11) with two pairs of S'_{PM}, S''_{PM} for COPIC instances $(\mathcal{PM}(K_{m,m}), \mathcal{PM}(K_{n,n}), Q, c, d)$, and two pairs of S'_{ST}, S''_{ST} for the case of COPIC instances $(\mathcal{B}(\mathcal{M}(K_m)), \mathcal{B}(\mathcal{M}(K_n)), Q, c, d)$. Using analogous approach, the remaining statements of the theorem can be shown. □

As we mentioned before, property (i) of Lemma 4.20 does not hold for unconstrained solution set $2^{[m]}$ ($2^{[n]}$), nevertheless, it is not hard to show that $\mathrm{CVP}_1(\mathcal{F}_1) + \mathrm{CVP}_2(\mathcal{F}_2)$ characterizes all linearizable matrices even if $\mathcal{F}_1 = 2^{[m]}$ or $\mathcal{F}_2 = 2^{[n]}$.

Theorem 4.22. *A* COPIC *instance* $(\mathcal{F}_1, \mathcal{F}_2, Q, c, d)$ *with* $\mathcal{F}_1 = 2^{[m]}$ *(*$\mathcal{F}_2 = 2^{[n]}$*) is linearizable if and only if* $Q \in CVP_2(\mathcal{F}_2)$ *(*$Q \in CVP_1(\mathcal{F}_1)$*).*

Proof. Assume that $\mathcal{F}_1 = 2^{[m]}$. Note that $\mathrm{CVP}_1(2^{[m]})$ contains only the $m \times n$ zero matrix, hence Lemma 4.19 implies that elements of $\mathrm{CVP}_2(\mathcal{F}_2)$ are linearizable.

Now let us assume that Q is linearizable and not an element of $\mathrm{CVP}_2(\mathcal{F}_2)$. Then there must exist some $i' \in [m]$ and $S_2, S'_2 \in \mathcal{F}_2$ such that $\sum_{j\in S_2} q_{i'j} \neq \sum_{j\in S'_2} q_{i'j}$. Let $a = (a_i)$ and $b = (b_i)$ be a linearization of Q. Since $\{i'\} \in 2^{[m]}$, we have that

$$\sum_{j\in S_2} q_{i'j} = \sum_{i\in\{i'\}}\sum_{j\in S_2} q_{ij} = a_{i'} + \sum_{j\in S_2} b_j$$

and

$$\sum_{j \in S_2'} q_{i'j} = \sum_{i \in \{i'\}} \sum_{j \in S_2'} q_{ij} = a_{i'} + \sum_{j \in S_2'} b_j.$$

Hence, $\sum_{j \in S_2} b_j \neq \sum_{j \in S_2'} b_j$. However, since $\emptyset \in 2^{[m]}$ we have

$$0 = \sum_{i \in \emptyset} \sum_{j \in S_2} q_{ij} = \sum_{j \in S_2} b_j \quad \text{and} \quad 0 = \sum_{i \in \emptyset} \sum_{j \in S_2'} q_{ij} = \sum_{j \in S_2'} b_j$$

which implies that $\sum_{j \in S_2} b_j = \sum_{j \in S_2'} b_j$, a contradiction. □

4.6 Conclusion

We introduced a general model to study combinatorial optimization problems with interaction costs and showed that many classical hard combinatorial optimization problems are special cases. In many cases, interaction costs can be identified as the origin of the hardness of these problems. Therefore we considered special structures of interaction costs, and their impact on the computational complexity of the underlying combinatorial optimization problems. We presented a general approach based on multi-parametric programming to solve instances parametrized with the rank of the interaction cost matrix Q. Complementary to that, we analyzed problems with diagonal interaction cost matrix Q, which can be used to enforce disjointness constraints. Even for this special type of interaction costs, we can show that for many common sets of feasible solutions, that have no matroid structure, COPIC becomes NP-hard. We also identified conditions on the interaction costs so that COPIC can be reduced to an equivalent instance with no interaction costs.

To further characterize how interaction costs impact the computational complexity of different combinatorial optimization problems, the following questions could be addressed.

- Are the polynomially solvable cases of COPIC where matrix Q has fixed rank r W[1]-hard?
- For cases of COPIC with diagonal matrix that can be efficiently solved, analyze the parameterized complexity with respect to the bandwith of Q.
- Can a COPIC instance $(\mathcal{B}(\mathcal{U}_m^k), \mathcal{P}_{s,t}(G), \mathrm{diag}(a), c, d)$ be solved in polynomial time, if $a \geq 0, c \geq 0$ and $d \geq 0$?
- Is there a polynomial time algorithm for COPIC with instances restricted to the form $(\mathcal{B}(\mathcal{M}_1), \mathcal{B}(\mathcal{M}_2), \mathrm{diag}(a), c, d)$, without any restrictions on $\mathcal{M}_1, \mathcal{M}_2, a, c$ and d?

For the case of diagonal COPIC it would be interesting to study further types of sets of feasible solutions. For example the matching-cut problem analyzed by Bonsma [16]

can be also formulated as a special case of diagonal COPIC, so analyzing graph cuts as feasible sets in diagonal COPIC is an interesting candidate for further research.

Additionally, understanding the influence of interaction costs with other special matrix structures, besides fixed rank and diagonal matrices, to the computational complexity of combinatorial optimization problems would be of interest.

5 Matrix Completion Problems

5.1 Introduction

Restricted classes of matrices are common objects of study in mathematics and combinatorics [19, 20], with strong connections to combinatorial optimization. Examples include, but are not limited to, low rank matrices, Monge matrices, bottleneck Monge matrices, matrices with restrictions on the row and column sums (magic squares), Latin squares and their generalizations (Sudoku variants) and adjacency matrices of graphs with certain properties. Since in many applications, the order of rows and columns of a matrix is not important also permuted versions of those classes are studied. Here, for square matrices, we differentiate between the case where the rows and columns have to be permuted by the same or two independent permutations. Formally, for a matrix $A = (a_{i,j})$ and permutations π, σ we denote by $A^{\pi,\sigma} = (a_{\pi(i),\sigma(j)})$ the matrix where all rows are permuted by π and all columns are permuted by σ. If A is a square matrix, we write $A^{\pi} := A^{\pi,\pi}$ for the matrix where rows and columns are permuted using the same permutation. Given a matrix class $\mathcal{F}$ we denote by $\mathcal{F}^{\pi}$ the set of all square matrices A such that there exists a permutation π with $A^{\pi} \in \mathcal{F}$. The class $\mathcal{F}^{\pi,\sigma}$ is the set of matrices such that there exist permutations π, σ such that $A^{\pi,\sigma} \in \mathcal{F}$.

For different classes of matrices, there is a vast literature on the computational complexity of the resulting recognition problem. For a given matrix class $\mathcal{F}$ the $\mathcal{F}$-Recognition problem is the problem to decide for any given matrix A, if $A \in \mathcal{F}$. Especially for permuted matrix classes $\mathcal{F}^{\pi,\sigma}$ this problem is of great relevance in combinatorial optimization [25].

In this chapter we focus on another problem variant, namely matrix completion problems. Given a matrix $A \in (\mathbb{K} \cup \{*\})^{m \times n}$ and a matrix class $\mathcal{F}$, the $\mathcal{F}$-Completion problem asks whether there exist values $\tilde{a}_{i,j} \in \mathbb{K}$ for all i, j with $a_{i,j} = *$ such that the matrix $\tilde{A} = (\tilde{a}_{i,j}) \in \mathcal{F}$, where $\tilde{a}_{i,j} = a_{i,j}$, if $a_{i,j} \neq *$.

For some matrix classes, the matrix completion problem is already well-studied in the literature. One major example, with a huge amount of applications, is matrix completion of low rank matrices [27]. Peeters [105] shows that the problem is already NP-hard for rank 3 matrices. But the case of rank 1 matrices is efficiently solvable, even for noisy input data [34]. Another well-studied example is the Latin square completion problem, which is also NP-hard [33]. There also exist many kinds of casual games based on the idea of matrix completion. The most famous example is Sudoku, which is also known to be NP-complete [119].

In this thesis we obtain new results about the computational complexity of matrix completion problems. We focus mainly on matrices which play an important role in connection with efficiently solvable cases of hard combinatorial optimization problems

such as Large matrices, Monge matrices, bottleneck Monge matrices and matrices with monotonicity properties in their rows and columns, as well as permuted variants of those classes.

In the following we call a matrix $A \in \mathbb{R}^{m \times n}$ a completed matrix and a matrix which contains $*$-entries, i.e. $A \in (\mathbb{R} \cup \{*\})^{m \times n}$ which is not completed, a partial matrix.

For the hardness results we use reductions from the following well-known NP-complete problems 3SAT and Betweenness [102].

Problem: 3-Satisfiability (3SAT)

Instance: Variables $x_1, x_2, \ldots, x_n$ and length 3 clauses $c_1, c_2, \ldots, c_m$ of the form $c_j = (l_1^{(j)} \vee l_2^{(j)} \vee l_3^{(j)})$, where $l_i^{(j)}$ are literals of the given variables.

Question: Does there exist a truth assignment to the variables $x_1, x_2, \ldots, x_n$ such that $c_1 \wedge c_2 \wedge \cdots \wedge c_m$ is true?

Problem: Betweenness

Instance: Set of n integers S and a set of triples $T \subseteq S^3$.

Question: Does there exist a total order $<$ of the elements in S with the property that for each triple $(t_1, t_2, t_3) \in T$ it holds that $t_1 < t_2 < t_3$ or $t_3 < t_2 < t_1$?

5.2 Large matrices

A matrix $A = (a_{i,j}) \in \mathbb{R}^{m \times n}$ is called *large*, if there exist vectors $\alpha \in \mathbb{R}^m$ and $\beta \in \mathbb{R}^n$ such that $a_{i,j} = \max\{\alpha_i, \beta_j\}$. We denote the set of all large matrices by $\mathcal{F}_L$. The Large Matrix Completion ($\mathcal{F}_L$-Completion) problem asks for a given partial matrix A such that $A = (a_{i,j}) \in (\mathbb{R} \cup \{*\})^{m \times n}$, if it can be completed to a large matrix. Let $\Omega = \{(i,j) \colon a_{i,j} \neq *\}$, then we want to check if there exist vectors $\alpha \in \mathbb{R}^m$, $\beta \in \mathbb{R}^n$ such that $a_{i,j} = \max\{\alpha_i, \beta_j\}$ for all $(i,j) \in \Omega$. First, it is important to note that permutations of the rows and/or columns do not influence the problem, so $\mathcal{F}_L$-Completion is equivalent to $\mathcal{F}_L^{\pi,\sigma}$-Completion.

We show below that the following algorithm (Algorithm 5.1) solves the $\mathcal{F}_L$-Completion problem. The main idea is to iteratively increase the α and β values, as they are implied by the $a_{i,j}$ values in non-decreasing order. This process either leads to vectors α, β that match the current partial matrix (Lemma 5.1) or proves infeasibility of the current instance (Lemma 5.2).

Algorithm 5.1: Algorithm for the LARGE MATRIX COMPLETION problem.

Sort the given data entries $a_{i,j}$ for $(i,j) \in \Omega$ to obtain an ordering such that $a_{i_1,j_1} \leq a_{i_2,j_2} \leq \cdots \leq a_{i_{|\Omega|},j_{|\Omega|}}$.
Initialize $\alpha_i = -\infty, \beta_j = -\infty$ for all $i = 1,2,\ldots,m$ and $j = 1,2,\ldots,n$.
for $k := 1,\ldots,|\Omega|$ **do**
  $(i,j) := (i_k, j_k)$
  if $\alpha_i = \beta_j = -\infty$ **then**
1 $\alpha_i := a_{i,j}$
   $\beta_j := a_{i,j}$
  else if $\alpha_i = -\infty \wedge \beta_j > -\infty$ **then**
2 $\alpha_i := a_{i,j}$
  else if $\alpha_i > -\infty \wedge \beta_j = -\infty$ **then**
3 $\beta_j := a_{i,j}$
  else if $-\infty < \alpha_i < a_{i,j} \wedge -\infty < \beta_j < a_{i,j}$ **then**
4 **return** *'no feasible solution'*
Set all α_i, β_j that still have value $-\infty$ arbitrarily.
return (α, β)

Lemma 5.1. *If Algorithm 5.1 terminates with a solution (α,β) then $\bar{A} = (\bar{a}_{i,j})$ with $\bar{a}_{i,j} = \max\{\alpha_i, \beta_j\}$ is a feasible solution to the* LARGE MATRIX COMPLETION *problem.*

Proof. By construction $\bar{A}$ is a large matrix. Let $(i,j) \in \Omega$. Then there is a k such that $(i_k, j_k) = (i,j)$. First observe, that after step k we have $\max\{\alpha_i, \beta_j\} = a_{i,j}$. Before step k it holds that $\alpha_{i'} \leq a_{i,j}$ and $\beta_{j'} \leq a_{i,j}$ for all i', j'. If in step k we are in one of the Cases 1-3 of the algorithm, we set at least one of the two values α_i, β_j to $a_{i,j}$ from which the claim follows, since all values set earlier are less or equal to $a_{i,j}$. Case 4 does not apply, since we assume that the algorithm terminates with the solution (α,β). If none of the Cases 1-4 apply we have that both $-\infty < \alpha_i$, $-\infty < \beta_j$ and either $\alpha_i = a_{i,j}$ or $\beta_j = a_{i,j}$, so again $\max\{\alpha_i, \beta_j\} = a_{i,j}$.

In all future steps, if the algorithm handles an element $a_{i,j'}$ or $a_{i',j}$ we do not set α_i and β_j to a new value. So at the end of the algorithm it holds that $\max\{\alpha_i, \beta_j\} = a_{i,j}$. □

Lemma 5.2. *If Algorithm 5.1 returns 'no feasible solution', there exists no feasible solution to the* LARGE MATRIX COMPLETION *problem.*

Proof. Since the algorithm terminated with 'no feasible solution' there exists a step k in which it terminates in Case 4. Let $(i,j) = (i_k, j_k)$. If there would be a feasible completion (α', β'), then either $\alpha'_i = a_{i,j}$ or $\beta'_j = a_{i,j}$.

But in step k of the algorithm for the current vectors α, β it holds that both $-\infty < \alpha_i < a_{i,j}$ and $-\infty < \beta_i < a_{i,j}$. This implies that there exist i', j', such that both $a_{i,j'} < a_{i,j}$ and $a_{i',j} < a_{i,j}$, else the algorithm would not have set α_i and β_j to these values.

Since in (α', β') either $\alpha'_i = a_{i,j}$ or $\beta'_j = a_{i,j}$, we have either $\max\{\alpha'_i, \beta'_{j'}\} \geq a_{i,j} > a_{i,j'}$ or $\max\{\alpha'_{i'}, \beta'_j\} \geq a_{i,j} > a_{i',j}$ both, contradicting the assumption that (α', β') is a feasible completion. □

Combining both Lemma 5.1 and Lemma 5.2 we obtain the following theorem.

Theorem 5.3. *Algorithm 5.1 solves the* Large Matrix Completion *problem and runs in* $O(|\Omega|^2 \log |\Omega| + n^2)$ *time.*

Proof. Correctness follows directly from the two preceding lemmas. The running time bound is obtained from the time needed for sorting and the time to check each matrix element. □

5.3 Permuted Matrices with Non-Decreasing Rows and Columns

A matrix $A = (a_{i,j}) \in \mathbb{R}^{m \times n}$ has the non-decreasing rows and columns (NDRC) property, if for all $1 \leq i \leq k \leq m$ and $1 \leq j \leq l \leq n$ it holds that $a_{i,j} \leq a_{k,j}$ and $a_{i,j} \leq a_{i,l}$. We call such a matrix NDRC matrix, and denote the set of such matrices by $\mathcal{F}_{\leq,\leq}$.

Theorem 5.4. *The* $\mathcal{F}_{\leq,\leq}$-Completion *problem can be solved in polynomial time.*

Proof. This problem can be reduced to the feasibility problem for linear programming. One simply introduces variables $x_{i,j}$ for each $i \in [m], j \in [n]$. If $a_{i,j} \neq *$ we add the constraint $x_{i,j} = a_{i,j}$. In addition we add constraints $x_{i,j} \leq x_{k,j}$ and $x_{i,j} \leq x_{i,l}$ for all $1 \leq i \leq k \leq m$ and $1 \leq j \leq l \leq n$. This set of linear constraints clearly has a feasible solution if and only if the given matrix can be completed to a matrix with the NDRC property. □

A matrix A, where $A = (a_{i,j})$, is called permuted with non-decreasing rows and columns (PNDRC), if there exist permutations $\pi \in S_m, \sigma \in S_n$ such that $A^{(\pi,\sigma)} = (a_{\pi(i),\sigma(j)})$ has the NDRC property. The set of such matrices is denoted by $\mathcal{F}^{\pi,\sigma}_{\leq,\leq}$. The PNDRC Matrix Completion ($\mathcal{F}^{\pi,\sigma}_{\leq,\leq}$-Completion) problem asks for a given partial matrix $A = (a_{i,j}) \in (\mathbb{R} \cup \{*\})^{m \times n}$, if we can complete it to a PNDRC matrix. Let again $\Omega = \{(i,j) \colon a_{i,j} \neq *\}$, then we want to check if there exist values $v_{i,j}$ for all $(i,j) \in \Omega$ such that $\bar{A} = (\bar{a}_{i,j})$ with $\bar{a}_{i,j} = v_{i,j}$ for all $(i,j) \in \Omega$ and $\bar{a}_{i,j} = a_{i,j}$ for all $(i,j) \notin \Omega$ is a PNDRC matrix. In this case, in contrast to large matrices, the problem itself and its computational complexity completely change as soon as we allow the permutation of rows and columns.

Theorem 5.5. *The* $\mathcal{F}^{\pi,\sigma}_{\leq,\leq}$-Completion *problem is NP-complete.*

Proof. We prove this by reduction from the 3SAT problem.

Given a 3SAT-instance we create a matrix $A = (a_{i,j})$ of size $(2n+1) \times (3m+1)$. This matrix contains a special row/column which we fix to be the topmost/leftmost by setting

$a_{0,0} := 0$ (the only 0 entry of the matrix) and by setting $a_{0,3j-1} = a_{0,3j-2} = a_{0,3j-3} = j$ for $j = 1, 2, \ldots, m$, and $a_{2i-1,0} = a_{2i-2,0} = i$ for $i = 1, 2, \ldots, n$.

The two rows with entries set to i correspond to the variable x_i and the three columns with entries set to j correspond to the clause c_j. Now we describe how the remaining matrix entries are set. For the 2×3 submatrix corresponding to variable x_i and clause c_j four out of the six entries are set to $*$ and the remaining ones are assigned the values $b_{i,j} = 2(i + j - 1) - 1$ and $\hat{b}_{i,j} = 2(i + j - 1)$. Note that $b_{i,j} \leq \hat{b}_{i,j}$ and that $\hat{b}_{i,j} < b_{i',j}$ holds for $i < i'$ and $b_{i,j} < b_{i,j'}$ holds for $j < j'$. We still need to specify where the four $*$ are placed and where $b_{i,j}$ and where $\hat{b}_{i,j}$. That depends on the clause c_j.

We assume that the literals of c_j are sorted according to the index of their variables. Then, we use the following patterns for the 2×3 submatrices corresponding to the literals of c_j.

If $l_1^{(j)} = x_i$, respectively $l_1^{(j)} = \neg x_i$ we use the submatrices

$$\begin{pmatrix} b_{i,j} & * & * \\ * & \hat{b}_{i,j} & * \end{pmatrix}, \text{ respectively } \begin{pmatrix} * & \hat{b}_{i,j} & * \\ b_{i,j} & * & * \end{pmatrix}.$$

Analogously for the literals $l_2^{(j)}$ and $l_3^{(j)}$ we use the submatrices

$$\begin{pmatrix} * & b_{i,j} & * \\ * & * & \hat{b}_{i,j} \end{pmatrix}, \text{ respectively } \begin{pmatrix} * & * & \hat{b}_{i,j} \\ * & b_{i,j} & * \end{pmatrix}$$

and

$$\begin{pmatrix} * & * & b_{i,j} \\ \hat{b}_{i,j} & * & * \end{pmatrix}, \text{ respectively } \begin{pmatrix} \hat{b}_{i,j} & * & * \\ * & * & b_{i,j} \end{pmatrix}.$$

All other matrix entries are set to $*$.

With respect to choosing the permutation for the rows, the only freedom is to select the order of the two rows corresponding to variable x_i for each i. Changing the order corresponds to setting x_i to false and leaving it as is corresponds to setting x_i to true.

Observe, that when the rows corresponding to the variables of a clause j are set in such a way that all literals are false, every permutation of the columns leads to an infeasible row or column as we end up with a cycle of strict inequalities among the columns. This follows by the fact that in this case after permutation the matrices

$$\begin{pmatrix} * & \hat{b}_{i,j} & * \\ b_{i,j} & * & * \end{pmatrix}, \begin{pmatrix} * & * & \hat{b}_{i,j} \\ * & b_{i,j} & * \end{pmatrix}, \begin{pmatrix} \hat{b}_{i,j} & * & * \\ * & * & b_{i,j} \end{pmatrix}$$

all appear in the c_j columns, which imply the following order on the three columns: $1 < 2$, $2 < 3$ and $3 < 1$, which is a contradiction.

Otherwise, i.e. as soon as one of those orders among the columns does not need to hold, it is easy to resolve any infeasibility by just permuting the columns. □

The $\mathcal{F}_{\leq,\leq}^{\pi,\sigma}$-Completion problem for $\{0,1\}$-matrices can be solved in polynomial time, as was shown by Golumbic [65, Section 2].

0	1	1	1	2	2	2	...	m	m	m
1	$*$	2	$*$							
1	1	$*$	$*$							
2	$*$	$*$	4							
2	$*$	3	$*$							
3	6	$*$	$*$							
3	$*$	$*$	5							
$\vdots$										
n	$*$	$*$	$*$							
n	$*$	$*$	$*$							

Figure 5.1: Structure of the matrix in the 3SAT reduction in Theorem 5.5. The entries for clause c_1 correspond to $c_1 = \neg x_1 \vee \neg x_2 \vee \neg x_3$.

5.4 Bottleneck Monge Matrices

Bottleneck Monge matrices are another type of matrices which play a prominent role in the combinatorial optimization literature. They fulfill the property that for every $i < i'$ and $j < j'$ it holds that

$$\max\{a_{i,j}, a_{i',j'}\} \leq \max\{a_{i,j'}, a_{i',j}\}.$$

The set of all bottleneck Monge matrices is denoted by $\mathcal{F}_{BM}$. The $\mathcal{F}_{BM}$-Completion problem can be solved in polynomial time, by an LP-approach similar to the one in Theorem 5.4. For the permuted case we are able to show NP-hardness already for the $\{0,1\}$-matrix case.

Theorem 5.6. *The $\mathcal{F}_{BM}^{\pi,\sigma}$-Completion problem is NP-complete, even if restricted to $\{0,1\}$-matrices.*

Proof. This follows again via reduction from the Betweenness problem. Let S be a set of n integers and $T \subseteq S^3$, be a given instance of the Betweenness problem.

We create the following partial matrix as an instance of the Permuted Bottleneck Monge Completion problem. The columns correspond to the n elements of S. For each triple $(s_1, s_2, s_3) \in T$ we create two rows of the following form.

			s_1				s_2				s_3			
			$\downarrow$				$\downarrow$				$\downarrow$			
$*$	...	$*$	0	$*$	...	$*$	0	$*$	...	$*$	1	$*$	...	$*$
$*$	...	$*$	1	$*$	...	$*$	0	$*$	...	$*$	0	$*$	...	$*$

It is easy to see that if s_2 is not between s_1 and s_3 in the column order, the matrix cannot be completed to a Bottleneck Monge matrix, since no matter which order the two rows have, the s_2 column contradicts the bottleneck Monge property with one of s_1 or s_2.

On the other hand, if s_2 is between s_1 and s_3 in the column order, the 6 fixed entries in the two rows fulfill the Monge property directly if the order is s_1, s_2, s_3 and by swapping the two rows, if the order is s_3, s_2, s_1. It is important to notice that for the arguments above only the relative order of the two rows associated with a triple is important and not their exact position relative to other rows. What remains to be shown is that in the case in which a feasible total ordering exists the $*$-entries can be completed such that the full matrix is bottleneck Monge.

This can be achieved by setting every entry left of the leftmost 1 (in the second row) to 1 and everything right of the rightmost 1 (in the first row) also to 1. All other entries in the two rows are set to 0. For example for the feasible ordering above one would fill the $*$ entries in the following way.

$$\begin{array}{cccccccccccccccc} & & & & s_1 & & & & s_2 & & & & s_3 & & & \\ & & & & \downarrow & & & & \downarrow & & & & \downarrow & & & \\ 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 1 & 1 & \dots & 1 \\ 1 & \dots & 1 & 1 & 0 & \dots & 0 & 0 & 0 & \dots & 0 & 0 & 0 & \dots & 0 \end{array}$$

First observe that locally in these two rows the bottleneck Monge property is fulfilled.

Now we move all first rows of each triple to the top and all second rows to the bottom. It is important to notice that we order these rows with respect to the order of their rightmost element (i.e. the beginning of the block of 1's, starting with the longest block.

Analogously we move all the second rows of each triple to the bottom, again sorting them with respect to the length of their blocks of 1's, in this case starting with the shortest block. This order of rows then leads to a bottleneck Monge property, because it is a *double staircase matrix* [83] as can easily be checked. □

5.5 Monge Matrices

Analogously to bottleneck Monge matrices, we call a matrix $A = (a_{i,j})$ a *Monge* matrix if for every $i < i'$ and $j < j'$ it holds that

$$a_{i,j} + a_{i',j'} \leq a_{i,j'} + a_{i',j}.$$

The set of all Monge matrices is denoted by $\mathcal{F}_M$. Again the $\mathcal{F}_M$-Completion problem can be solved in polynomial time using an LP-approach.

Similarly to Theorem 5.6, the permuted case turns out to be NP-complete.

Theorem 5.7. $\mathcal{F}_M^{\pi,\sigma}$-Completion *is NP-complete.*

Proof. This follows via reduction form the Betweenness problem. Let S be a set of n integers and $T \subseteq S^3$ be a given instance of the Betweenness problem.

We create the following partial matrix as an instance of the Permuted Monge Completion problem. The columns correspond to the n elements of S. For the i-th

triple $(s_1, s_2, s_3) \in T$ we create two rows of the following form.

$$\begin{array}{ccccccccccccccccccc} & & & & & & & s_1 & & & & & & s_2 & & & & & s_3 \\ & & & & & & & \downarrow & & & & & & \downarrow & & & & & \downarrow \\ * & \ldots & * & 1 & * & \ldots & * & 3 & * & \ldots & * & 5 & * & \ldots & * \\ * & \ldots & * & 6 & * & \ldots & * & 7 & * & \ldots & * & 8 & * & \ldots & * \end{array}$$

These two rows are in addition multiplied by nM^i for a properly chosen large M specified below. We call i the triple-index of these rows.

It is easy to verify that the given row-order and the Monge property enforces the columns to be in order s_1, s_2, s_3 (increasing order) and the swapped row-order enforces the columns to be in order s_3, s_2, s_1 (decreasing order). Depending on these cases we say that a row is in increasing/decreasing order accordingly. This directly implies that any feasible Monge matrix completion together with the column permutation corresponds to a feasible solution of the betweenness problem.

Given a solution to the betweenness problem we permute all rows and columns according to this solution. In addition, all rows which are in increasing order are moved to the top of the matrix and all rows which are in decreasing order are moved to the bottom. In the increasing part the rows are sorted in increasing order according to their triple-index and in the decreasing part in decreasing order according to their triple-index. The *-entries in these rows are filled in decreasing/increasing order, such that two consecutive values differ by at least M^i for rows with triple-index i.

For a 2×2 submatrix $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ corresponding to an increasing and a decreasing row ($a < b$ and $c > d$) the Monge property holds trivially. Otherwise let $\left(\begin{smallmatrix} a & b \\ c & d \end{smallmatrix}\right)$ be a 2×2 submatrix inside the decreasing part of the matrix and let the row containing a, b correspond to the i-th triple and the row containing c, d correspond to the j-th triple. It still holds that $c - d \geq M^j \geq M^{i+1}$. By setting $M := 10n$ it holds that $M^{i+1} \geq 10nM^i > a \geq a - b$. An analogous argument holds for the increasing a submatrix within the increasing part of the matrix. We have that $b - a \geq M^i \geq M^{j+1} \geq 10nM^j > d \geq d - c$. □

In contrast to matrices with the NDRC property, the case of permuted $\{0, 1\}$-matrices has not been solved for the Monge property. We resolve the complexity status of this special case by using the following characterization of $\{0, 1\}$-Monge matrices in terms of block matrices obtained by Klinz et al. [84].

Corollary 5.8 (of Theorem 4.5 in Klinz et al. [84])**.** *A $\{0, 1\}$-matrix A is a permuted Monge matrix if and only if its rows and columns can be permuted such that it or its transpose has the block structure*

$$\left(\begin{array}{c|c|c|c} 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 1 \end{array}\right) \quad \textit{or} \quad \left(\begin{array}{c|c|c|c} 0 & 0 & 1 & 1 \\ \hline 0 & 1 & 1 & 1 \\ \hline 0 & 1 & 0 & 1 \end{array}\right),$$

where 0 *and* 1 *correspond to blocks of only* 0*'s and* 1*'s.*

Proof. This follows directly from Theorem 4.5 shown in [84]. We observe that if two rows are identical after the removal of an identical column, they have also been identical before. This observation is true since the removed columns entries in the two rows are identical to the entries of the other identical column that was not removed. □

Because of symmetry it is enough to consider the problem of completing a matrix into a permuted block matrix of the following form.

$$\left(\begin{array}{c|c|c|c} 0 & 0 & 1 & 1 \\ \hline 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 1 \end{array}\right) \tag{5.1}$$

Lemma 5.9. *Consider the columns containing both fixed* 0- *and* 1*-entries and possibly some* $*$*-entries. The matrix can only be completed and permuted into form (5.1), if these columns can be completed and permuted into one of the following block forms:*

$$\left(\begin{array}{c|c} 0 & 1 \\ \hline 0 & 0 \\ \hline 1 & 0 \end{array}\right), \left(\begin{array}{c} 0 \\ \hline 0 \\ \hline 1 \end{array}\right), \left(\begin{array}{c} 1 \\ \hline 0 \\ \hline 0 \end{array}\right).$$

Proof. If the entire matrix is completable and permutable into form (5.1), then these columns mentioned in the statement of the lemma must belong to the two center block columns of (5.1), since only those contain both 0- and 1-entries. So removing any other columns from the final completed and permuted matrix gives a block matrix of one of the three forms provided in the statement of the lemma. □

Based on the observation above we state the following reduction procedure.

1. Check if there are columns with only 0- or $*$-entries and columns with only 1- or $*$-entries. If so, complete the $*$-entries to generate all 0 columns and all 1 columns respectively and remove them.

2. Next we have to check if the rest of the matrix can be permuted and completed into the block form
$$\left(\begin{array}{c|c} 0 & 1 \\ \hline 0 & 0 \\ \hline 1 & 0 \end{array}\right).$$
We check if there are rows with only 0- and $*$-entries. If so, we set all entries in these rows to 0 and remove them.

3. For the remaining part of the matrix it holds that in every row and in every column there is at least one 0 and at least one 1 entry. Hence we reduced the original problem to the new problem to check if the matrix can be permuted and completed into the block form
$$\left(\begin{array}{c|c} 0 & 1 \\ \hline 1 & 0 \end{array}\right). \tag{5.2}$$

In the sequel we describe a reduction of the new problem mentioned in step 3 of the proof above to a problem in graphs. Given a $\{0,1\}$-matrix we obtain a bipartite graph $G = (R \dot\cup C, E^0 \dot\cup E^1)$. The vertex set R corresponds to the rows of A and the vertex set C corresponds to the columns of A. Vertices $r \in R, c \in C$ form an edge $\{r,c\} \in E^i$, if the entry $a_{r,c} = i$.

Observation 5.10. *A $\{0,1\}$-matrix is in permuted block form (5.2) if and only if the graph $G = (R \dot\cup C, E^1)$ is the disjoint union of two complete bipartite graphs.*

Since E^1 and E^0 are disjoint we have to find a partition of the vertices R and C into two groups such that the edges in E^1 only occur within the groups and no edge in E^0 is contained within any of the two groups. Such a partition is called *feasible*.

We achieve such a labeling of the vertices into two groups I and II using the following breadth first search procedure for each connected component of G.

We start by labeling an arbitrary vertex v_0 with label I. Then we process the vertices in breadth first search order starting with v_0. When processing a vertex v, if v has label I we check if all its labeled E^1 neighbors have label I and all its labeled E^0 neighbors have label II. If not, the procedure terminates with a contradiction. Otherwise, we label all its unlabeled E^1 neighbors with group I and all its unlabeled E^0 neighbors with label II. If v has label II, we perform the same steps interchanging I and II in the description above. If the breath first search terminates with no contradiction and not all vertices are yet labeled, we start another run of the same procedure using an arbitrary vertex that has not yet been labeled instead of v_0.

Lemma 5.11. *If the procedure above terminates with all vertices labeled, the partition of the graph $G = (R \dot\cup C, E^1)$ into groups I and II can be completed into two complete bipartite graphs formed by the vertices in group I and II. All edges in E^0 connect vertices from two different groups, hence the partition is feasible.*

Otherwise, if the procedure terminates with a contradiction, there exists no feasible partition of the vertices.

Proof. The positive case follows directly from Observation 5.10

For the validity of termination with a contradiction, observe that the assignment of class I for the first vertex is arbitrary. Afterwards, every new label set by the procedure is forced by the definition of a feasible partition. So as soon as the procedure finds a vertex it would label differently than with its current label we obtain a connected component in E^1 containing an edge in E^0 or vice versa, a contradiction to the existence of a feasible partition. □

Lemma 5.12. *If a graph $G = (R \dot\cup C, E^0 \dot\cup E^1)$ has a feasible partition, the procedure above terminates with all vertices labeled according to a feasible partition.*

Proof. Given a feasible partition, let the vertex set of G be labeled accordingly. Now without loss of generality, assume that the first vertex is labeled accordingly by the procedure. Then by construction and definition all its neighbors are labeled correctly. So by induction the whole connected component is labeled correctly. □

Combining the lemmas above, we obtain the following result.

Theorem 5.13. *The* $\mathcal{F}_M^{\pi,\sigma}$-Completion *problem for* $\{0,1\}$ *matrices can be solved in polynomial time.*

5.6 Open Questions

Our research on matrix completion problems leads to the following open questions in this area.

1. Can the $\mathcal{F}_{\leq,\leq}^{\pi,\sigma}$-Completion problem for $\{0,1,2\}$-matrices be solved in polynomial time?

2. Is matrix completion for permuted $\{0,1\}$-block-matrices of fixed structure solvable in polynomial time? What about general permuted block matrices?

Question 1 is a natural next step to gain a more-detailed understanding about the complexity of $\mathcal{F}_{\leq,\leq}^{\pi,\sigma}$-Completion. Question 2 is an interesting research direction on its own, since block matrix completion is a natural problem which does not yet have received attention in the literature.

6 Recoverable Robust Discrete Optimization

In this chapter we study recoverable robust optimization with interval uncertainties for different sets of feasible solutions. In Section 6.1 we focus on the case where the set of feasible solutions is the set of all subsets of cardinality q, and $q \in \mathbb{N}$ is a given input parameter. This is a robust version of the classic SELECTION problem. Section 6.2 studies the recoverable robust optimization problem with interval uncertainties, and variants of it, for the case where the set of feasible solutions are bases of a matroid. Also, variants of the problem are studied which lead to a result of independent interest in the intersection of combinatorial optimization and matroid theory. In addition, we generalize the recoverable robustness model to polymatroids.

6.1 Efficient Algorithms for the Recoverable (Robust) Selection Problem

6.1.1 Introduction

The SELECTION problem is widely studied in the computer science and optimization literature. Let $E = [n]$ be the set of base elements, with associated non-negative costs α_i for each $i \in E$. Given an integer $p \in [n]$ we want to choose a subset $A \subseteq E$ of size p that minimizes the cost $\sum_{i \in X} \alpha_i$. The problem can be solved in $O(n)$ time using a linear time algorithm for finding the p-th largest element [14, 48, 85]. The SELECTION problem can also be viewed as a polynomially solvable special case of the KNAPSACK problem, setting all element weights to 1. We denote by $\mathcal{S}_p^\alpha(E)$ an arbitrary instance of this problem and write $A = \mathcal{S}_p^\alpha(E)$ if A is an optimal solution to the given instance of the selection problem.

In this section we investigate a natural generalization of the SELECTION problem. For the element set $E = [n]$ we are given two cost vectors $\alpha, \beta \in \mathbb{R}^n$ and parameters $p, q \in \mathbb{N}$. We have to select $A, B \subseteq E$ both of size p that minimize the cost $\sum_{i \in A} \alpha_i + \sum_{j \in B} \beta_j$ subject to the constraint $|A \cap B| \geq q$ for the intersection of A and B. We denote an instance of this problem, the RECOVERABLE SELECTION problem, by $\mathcal{RS}_{p,q}^{\alpha,\beta}(E)$ and if (A^*, B^*) is an optimal solution of this problem we write $(A^*, B^*) = \mathcal{RS}_{p,q}^{\alpha,\beta}(E)$.

Kasperski and Zieliński [80] studied this problem as part of their fundamental work on (recoverable) robust discrete optimization (see [79] for a recent review) and obtained an $O(qn^2)$ time algorithm based on a reduction to a minimum cost flow problem.

We show in Section 6.1.2 that the RECOVERABLE SELECTION problem can be solved using a simple greedy algorithm. This is of special interest since for this problem no

matroidal structure is known that would directly imply such a result. In Section 6.1.3 we study the structure of optimal solutions with respect to two parameters and obtain discrete-convexity and unimodality results. Based on this detailed mathematical analysis we are able to obtain a linear time algorithm using prune and search [99].

Recoverable Robust Optimization – An Application The concept of recoverable robust optimization was introduced by Liebchen et al. [94]. The Recoverable Selection problem can be used to solve the recoverable robust version of the classic Selection problem with interval uncertainties [80]. In this case, instead of fixed costs for each element, we are given a scenario set $\mathcal{U}$ and for each scenario $s \in \mathcal{U}$ the costs of element $i \in E$ are denoted by $c_i^s \geq 0$ and setup costs C_i. In the robust optimization literature the interval uncertainty representation is a popular choice for defining scenario sets [89], which we denote by $\mathcal{U}^I$. For each $i \in E$ we are given an interval $[\underline{c}_i, \bar{c}_i]$ of possible cost realizations. The scenario set with interval uncertainties is given by $\mathcal{U}^I = \prod_{i \in E}[\underline{c}_i, \bar{c}_i]$. Then the Recoverable Robust Selection problem to be solved is

$$\min_{X \subseteq E\colon |X|=p} \max_{s \in \mathcal{U}^I} \sum_{i \in X} C_i + \min_{\substack{Y \subseteq E, |Y|=p \\ |Y \setminus X| \leq k}} \sum_{i \in Y} c_i^s.$$

Kasperski and Zieliński [80] observed that this is equivalent to solving $\mathcal{RS}_{p,p-k}^{C,\bar{c}}(E)$.

6.1.2 A Greedy Algorithm for the Recoverable Selection Problem

Algorithm 6.1 is a simple to implement greedy algorithm that solves the Recoverable Selection problem to optimality.

Algorithm 6.1: Greedy algorithm with growing selection parameter.

```
1 A := ∅, B := ∅
2 for l := 1, ..., p do
3 |   (a, b) := argmin{α_i + β_j : (i, j) ∈ E², i ∈ E \ A, j ∈ E \ B,
  |                                |(A + i) ∩ (B + j)| ≥ q − (p − l)}
4 |   A := A ∪ {a}
  |   B := B ∪ {b}
5 return (A, B)
```

To obtain an efficient implementation and prove correctness, it is necessary to have a closer look at the different cases in which the minimum in line 3 in each iteration of the for loop can occur. We distinguish the following cases.

α-greedy and β-greedy This is the case if we select $a = \operatorname{argmin}\{\alpha_i : i \in E \setminus A\}$ and $b = \operatorname{argmin}\{\beta_j : j \in E \setminus B\}$.

α-greedy and β-filling In this case we select $a = \operatorname{argmin}\{\alpha_i : i \in E \setminus A\}$, and dependent on a then $b = \operatorname{argmin}\{\beta_i : i \in E \setminus B, i \in A \cup \{a\}\}$. We call b the filling element of the step.

β-greedy and α-filling The symmetric case, i.e. $b = \operatorname{argmin}\{\beta_j \colon j \in E \setminus B\}$, and dependent on b then $a = \operatorname{argmin}\{\alpha_i \colon i \in E \setminus A, i \in B \cup \{b\}\}$. We call a the filling element.

$(\alpha+\beta)$-greedy The case where $a = b = \operatorname{argmin}\{\alpha_i + \beta_i \colon i \in E \setminus (A \cup B)\}$.

It is easy to see that each selection step of (a, b) in line 3 of Algorithm 6.1 belongs to at least one of the four types, but it can also happen that a step fits multiple of these types. Based on the case distinction above we can implement Algorithm 6.1 in $O(n \log n)$ time using sorting and priority queues. The proof of correctness is given in Section 6.1.2.1. In summary we obtain the following result.

Theorem 6.1. *The greedy algorithm (Algorithm 6.1) solves the* Recoverable Selection *problem in $O(n \log n)$ time.*

6.1.2.1 Proof of Correctness of Algorithm 6.1

The following lemmas summarize some properties of the algorithm that will be useful in the proof of its correctness. We introduce the following notation for subsets of selected elements A and B throughout the algorithm: $\tilde{Z} = A \cap B$, $\tilde{A} = A \setminus \tilde{Z}$, $\tilde{B} = B \setminus \tilde{Z}$, $\tilde{F} = E \setminus (A \cup B)$.

Lemma 6.2. *If (A, B) is the solution of Algorithm 6.1 after iteration l and $|X \cap Y| > q - (p - l)$, then the l-th step is X-greedy and Y-greedy.*

If in addition (A^-, B^-) is the solution after the $(l-1)$-th step and $|A^- \cap B^-| = q - (p - (l-1))$, then the q-th step is also α-greedy and β-filling as well as β-greedy and α-filling.

Proof. If at the $(l-1)$-th step $|A^- \cap B^-| > q - (p - (l-1))$, then the l-th step is a α- and β-greedy step by the greedy nature of Algorithm 6.1.

If $|A^- \cap B^-| = q - (p - (l-1))$ after step $l-1$, the only way to get $|A \cap B| < q - (p - l)$ after step l is by performing a step that is both α- and β-filling with $a \neq b$ (see Figure 6.1). But since the algorithm is greedy by construction it only chooses such a step if it is an α-greedy and β-filling step and a β-greedy and α-filling stepat the same time, implying that this step is also an α- and β-greedy step. □

Figure 6.1: A α- and β-filling step.

Lemma 6.3. *The following necessary conditions for optimality are invariants of Algorithm 6.1.*

1. *If step l of Algorithm 6.1 is α- and β-greedy and (A, B) is the solution after step l, then for all $x \in A, x' \notin A$ and $y \in B, y' \notin B$ it holds that*

a) $\alpha_x \leq \alpha_{x'}$,

b) $\beta_y \leq \beta_{y'}$.

This implies A and B are also optimal solutions to the Selection *problem for l elements with costs α and β.*

2. *For $x \in \tilde{A}, y \in \tilde{B}, z \in \tilde{Z}$ it holds that*

   a) $\alpha_z \leq \alpha_y$,

   b) $\beta_z \leq \beta_x$.

Proof. We only prove claims (1a) and (2a) since the claims (1b) and (2b) can be shown by analogous arguments. We again assume that (A^-, B^-) is the solution obtained by Algorithm 6.1 after step $l-1$. The results are shown by induction over the steps of the algorithm. The inductive base $l = 0$ is trivial.

We start by proving that (1) holds after step l by assuming both (1) and (2) hold for all steps before the current l-th step (a, b). If the l-th step is not α- and β-greedy there is nothing to show, so let the l-th step (a, b) be α- and β-greedy. If also the $(l-1)$-th step is α- and β-greedy the claim follows directly. Else, we know that $\alpha_a \leq \beta_{x'}$ because the step is α-greedy. By Lemma 6.2 we know that (a, b) is also a β-greedy and α-filling step so $a \in \tilde{B}^-$. By the induction hypothesis we hence obtain via (2a) that $\alpha_z \leq \alpha_a \leq \alpha_{x'}$ for all $z \in \tilde{Z}^-$. For elements $x \in \tilde{A}$ the claim $\alpha_x \leq \alpha_{x'}$ holds, because such an element x has been chosen α-greedy in a previous step.

To prove claim (2a) we distinguish between the different possible types of the l-th step where (a, b) gets selected.

α- and β-greedy In this case we have already shown that (1a) holds which implies (2a).

α-greedy and β-filling Since b is a β-filling step we know that $b \in \tilde{A}^-$, and becomes part of $\tilde{Z}$. It is the only new element of $\tilde{Z}$ after step l and $\tilde{B}$ stays the same. For all other $z \in \tilde{Z} \setminus \{b\}$ it follows by induction that $\alpha_z \leq \alpha_y$. Since $b \in \tilde{A}^-$, we know that it is selected in an earlier α-greedy step. So obviously, $\alpha_b \leq \alpha_y$.

α-filling and β-greedy If $a \neq b$ we know that $\alpha_z \leq \alpha_a$, since $a \in \tilde{B}^-$. So a is now the element in $\tilde{Z}$ with largest α value. By the choice of a we know that $\alpha_a \leq \alpha_y$, since it is chosen as the smallest α-filling element. Also $\alpha_a \leq \alpha_b$, since b is also a possible choice instead of a.

In the case that $a = b$ we know that $\alpha_a \leq \alpha_y$, since a is the best choice for a X-filling step. The rest follows by induction.

$(\alpha+\beta)$-greedy We know that $\alpha_a + \beta_a \leq \alpha_y + \beta_a$, else the algorithm does not execute the row-greedy step (a, a), since a β-greedy and α-filling step with smaller cost exists. This implies $\alpha_a \leq \alpha_y$. Since a is the only new element in $\tilde{Z}$, and $\tilde{B} = \tilde{B}^-$ the claim follows by induction.

□

Using this result we obtain that (A, B) is an optimal solution if $|\tilde{Z}| > q$ or $\tilde{F} = \emptyset$. Using this we show that Algorithm 6.1 always finds an optimal solution.

Lemma 6.4. *Let $E' \subseteq E$ and $(A, B), (A', B')$ be the solution obtained by running Algorithm 6.1 on E and E' for the same parameters p and q. Then*

$$\sum_{x \in A} \alpha_x + \sum_{y \in B} \beta_y \leq \sum_{x' \in A'} \alpha_{x'} + \sum_{y' \in B'} \beta_{y'}$$

Proof. Let (A, B) be the solution obtained by Algorithm 6.1 on E and (A', B') a solution obtained by Algorithm 6.1 on a subset $E' \subseteq E$. We assume that $|\tilde{Z}| = q$, else the claim follows from optimality of (A, B) shown above. We define an assignment of elements in (A, B) to elements in (A', B'). Elements in A are always assigned to elements in A' and elements in B are assigned to elements in B'. We only decide to assign a single element $x \in A$ ($y \in B$) to a single element $x' \in A'$ ($y' \in B'$) if it holds that $\alpha_x \leq \alpha_{x'}$ ($\beta_y \leq \beta_{y'}$). We decide to assign a pair of elements $(x, y) \in A \times B$ to another pair of elements $(x', y') \in A' \times B'$, if $\alpha_x + \beta_y \leq \alpha_{x'} + \beta_{y'}$. The existence of such an assignment implies that $\sum_{x \in A} \alpha_x + \sum_{y \in B} \beta_y \leq \sum_{x' \in A'} \alpha_{x'} + \sum_{y' \in B'} \beta_{y'}$. Before we construct such an assignment, we observe that $\tilde{Z} \cap E' \subseteq \tilde{Z}'$ because of Lemma 6.3 (2).

All the elements in $A \cap A'$ and $B \cap B'$ are assigned to themselves.

For elements $z \in \tilde{Z} \setminus E'$, we need both an element from $A' \setminus A$ and $B' \setminus B$. First observe that z is added to $\tilde{Z}$ in a row-greedy or a filling step (a, b). Since in a filling step the filling element is selected in an earlier greedy step for the other cost function we have that $\alpha_z + \beta_z \leq \alpha_a + \beta_b$.

Since we know that $|\tilde{Z}| = q \leq |\tilde{Z}'|$ we have that $|\tilde{Z} \setminus E'| \leq |\tilde{Z}' \setminus \tilde{Z}|$. So for each $z \in \tilde{Z} \setminus E'$ there is a $z' \in \tilde{Z}' \setminus \tilde{Z}$, which we select arbitrarily for each z. (If $|\tilde{Z} \setminus E'| < |\tilde{Z}' \setminus \tilde{Z}|$ we ensure that the not selected elements z' were added to $\tilde{Z}'$ via a α- or β-greedy step.) If $z' \notin A \cup B$ we know that $\alpha_z + \beta_z \leq \alpha_a + \beta_b \leq \alpha_{z'} + \beta_{z'}$, since (z', z') is a feasible step at the time step (a, b) is chosen. So we assign (z, z) to (z', z').

Else assume $z' \in A$ ($z' \in B$ analogously). If at the time of step (a, b) we have that $z' \notin A$ (z' could even be a) we know that $\alpha_z + \beta_z \leq \alpha_a + \beta_b \leq \alpha_{z'} + \beta_{z'} \leq \alpha_{x'} + \beta_{z'}$, where $x' \in \tilde{A}' \setminus A$, since (z', z') is again a feasible step instead of (a, b). Also z' is a greedy-chosen element in A, so $\alpha_{z'} \leq \alpha_{x'}$ for all $x' \in \tilde{A}' \setminus A$. Also an unassigned element x' must still be available, using $z' \in A$ and elementary counting arguments. So we assign (z, z) to (x', z'). If to the contrary at the time of step (a, b) we have $z' \in A$ we have that $\alpha_z + \beta_z \leq \alpha_a + \beta_b \leq \alpha_{x'} + \beta_{z'}$, for a $x' \in \tilde{A}' \setminus A$, since here (x', z') would have been a feasible step instead of (a, b). We again assign (z, z) to (x', z').

Left for assignment are only elements in $\tilde{A} \setminus E'$ and $\tilde{B} \setminus E'$. These are all chosen α- or β-greedy and by easy counting arguments there are exactly $|\tilde{A} \setminus E'|$ unassigned elements left in $\tilde{A}' \setminus A$ and $|\tilde{B} \setminus E'|$ elements in $\tilde{B}' \setminus B$. Since all unassigned elements are feasible candidates for the greedy steps performed on E, we can assign them arbitrarily.

By this assignment the claimed inequality follows. □

Theorem 6.5. *The solution (A, B) obtained by Algorithm 6.1 is an optimal solution for the* RECOVERABLE SELECTION *problem.*

Proof. Let (A^*, B^*) be an optimal solution to the given instance of the problem and (A, B) the solution obtained by the greedy algorithm. We set $E' = A^* \cup B^*$ and let (A', B') be the solution obtained by running Algorithm 6.1 on E'. Since $E' = |A^* \cap B^*| + 2|A^* \setminus B^*| \leq 2p - q$ we know that either $A' \cup B' = E'$ or $|A' \cap B'| > |A^* \cap B^*| \geq q$, so in both cases (A', B') is an optimal solution, i.e. $\sum_{x \in A'} \alpha_x + \sum_{y \in B'} \beta_y = \sum_{x \in A^*} \alpha_x + \sum_{y \in B^*} \beta_y$. By Lemma 6.4 it follows that $\sum_{x \in A} \alpha_x + \sum_{y \in B} \beta_y = \sum_{x \in A^*} \alpha_x + \sum_{y \in B^*} \beta_y$. □

Theorem 6.5 proves correctness of Algorithm 6.1, hence Theorem 6.1 directly follows.

6.1.3 A Linear Time Algorithm for the Recoverable Selection Problem

While the greedy algorithm is easy to understand and implement, its drawback is that it does not run in linear time. The question whether the Recoverable Selection problem can be solved in linear time suggests itself. In the following we provide an affirmative answer to this question. The reader is warned beforehand that both the description and the analysis of the linear time algorithm are much more involved than what we experienced for the greedy algorithm.

To achieve a linear running time we repeatedly use the fact that $\mathcal{S}_p^\alpha(E)$ can be calculated in $O(n)$ time. The main idea is to introduce parameters for different structural properties of solutions and then perform prune and search for the optimal values of these parameters in a two stage approach. To obtain fast algorithms we analyze the objective function with respect to the chosen parameters in detail and prove important properties of these functions. We start with a simple preprocessing step which removes trivial to select pairs of elements.

6.1.3.1 Preprocessing

In this section we describe a preprocessing procedure (Algorithm 6.2) that given an arbitrary instance (E, p, q, α, β) obtains an instance $(E', p', q', \alpha, \beta)$ with the properties

- $p' \leq \hat{p} \leq p$,
- $X' = \mathcal{S}_{\hat{p}}^\alpha(E')$,
- $Y' = \mathcal{S}_{\hat{p}}^\beta(E')$,
- $E' = X' \cup Y'$,
- $X' \cap Y' = \emptyset$.

Algorithm 6.2: Preprocessing algorithm.

1 Find sets $X = \mathcal{S}_p^\alpha(E), Y = \mathcal{S}_p^\beta(E), Z = \mathcal{S}_q^{\alpha+\beta}(E)$.
2 Define $S := (X \cap Y) \cup (Z \setminus (X \cup Y))$.
3 Return the reduced instance $((X \cup Y) \setminus (X \cap Y), p - |S|, q - |S|, \alpha, \beta)$ and the set S.

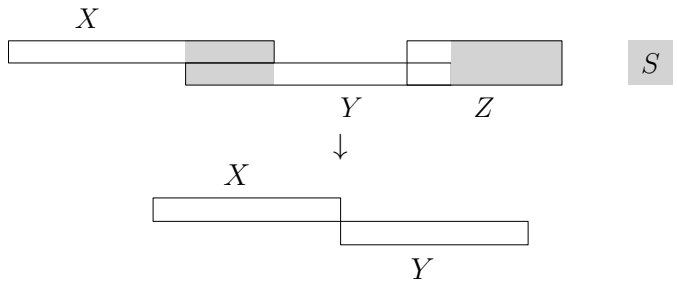


Figure 6.2: Visualization of the preprocessing step. The shaded gray area corresponds to the set S.

The following lemma shows that this preprocessing step is valid and allows us to obtain an instance $(E', p', q', \alpha, \beta)$ with the claimed properties. In addition we show that by using S, given an optimal solution for $(E', p', q', \alpha, \beta)$ we can obtain an optimal solution for the original instance.

Lemma 6.6. *Given an instance (E, p, q, α, β) of the* Recoverable Selection *problem and let (A^*, B^*) be an optimal solution to the reduced instance $(E', p', q', \alpha, \beta)$ produced by the preprocessing algorithm above together with the set S. Then $(A^* \cup S, B^* \cup S)$ is an optimal solution to the original instance.*

Proof. We show that there exists an optimal solution (A', B') such that $S \subseteq A', B'$. For the elements in $X \cap Y$ it is obvious that they can always be added to an arbitrary optimal solution. Let $z \in (Z \setminus (X \cup Y))$. If $z \notin A' \cap B'$ then it follows that $z \notin A' \cup B'$, since $z \notin X \cup Y$. Since $|A' \cap B'| \geq q$ it follows that there exists some $z' \in (A' \cap B') \setminus Z$. It holds that $\alpha_z + \beta_z \leq \alpha_{z'} + \beta_{z'}$ so we can swap z and z'. □

Since our preprocessing only needs to solve the selection problem three times for a set of cardinality n and afterwards builds intersections and unions of these sets it can be implemented in $O(n)$ time.

In the following, we assume that the given instance (E, p, q, α, β) is an instance obtained by running the preprocessing algorithm and we assume that the sets X and Y are given as part of the input and $E = X \dot\cup Y$.

6.1.3.2 Prune and Search for the Intersection Size s in X

The main idea to solve the Recoverable Selection problem is to introduce a parameter $s \in \{0, 1, \ldots, q\}$ counting the number of elements of $A \cap B$ that lie in X, i.e. $s = |X \cap A \cap B|$ (symmetrically $q - s$ elements of $A \cap B$ then lie in Y).

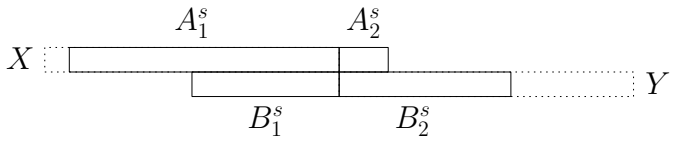


Figure 6.3: Illustration of the sets involved in the search for s^*.

For every $s \in \{0, 1, \dots, q\}$ let $A_1^s, B_1^s \subseteq X$ be an optimal solution to the problem

$$\begin{aligned} \min \ & \sum_{i \in A_1} \alpha_i + \sum_{j \in B_1} \beta_j \\ \text{s.t.} \ & |A_1| = p - (q - s) \\ & |B_1| = s \\ & A_1 \cap B_1 = B_1 \\ & A_1, B_1 \subseteq X \end{aligned}$$

which we denote by $\mathcal{RS}_{p,q,s}^{\alpha,\beta,1}(X)$ and $A_2^s, B_2^s \subseteq Y$ optimal solutions to the problem

$$\begin{aligned} \min \ & \sum_{i \in A_2} \alpha_i + \sum_{j \in B_2} \beta_j \\ \text{s.t.} \ & |B_2| = p - s \\ & |A_2| = q - s \\ & B_2 \cap A_2 = A_2 \\ & A_2, B_2 \subseteq Y \end{aligned}$$

which we denote by $\mathcal{RS}_{p,q,s}^{\alpha,\beta,2}(Y)$.

This directly implies that $(A, B) = (A_1^s \cup A_2^s, B_1^s \cup B_2^s)$ is a feasible solution to $\mathcal{RS}_{p,q}^{\alpha,\beta}(E)$ with exactly s elements of the intersection in X and $q - s$ in Y. In addition we observe that this approach also leads to an optimal solution.

Lemma 6.7. *There exist solutions such that for every $s \in \{0, 1, \dots, q-1\}$ it holds that $A_1^s \subseteq A_1^{s+1}$ and $B_1^s \subseteq B_1^{s+1}$.*

Symmetrically there also exist solutions such that $B_2^s \subseteq B_2^{s-1}$ and $A_2^s \subseteq A_2^{s-1}$ for $s \in \{1, 2, \dots, q\}$.

Proof. Let $z \in B_1^s$, hence $z \in A_1^s$. Assume $z \notin A_1^{s+1}$, hence $z \notin B_1^{s+1}$. If there exists an $r \in X \setminus A_1^s$ such that $r \in B_1^{s+1}$ we can swap z into A_1^{s+1} and B_1^{s+1} instead of r since $\alpha_z + \beta_z \leq \alpha_r + \beta_r$ by optimality of (A_1^s, B_1^s). Otherwise $B_1^{s+1} \subseteq A_1^s$, which implies there exists an $x_f \in A_1^s \setminus B_1^s$ and an $x_g \in X \setminus A_1^s$ such that $x_f \in B_1^{s+1}$ and $x_g \in A_1^{s+1}$. But again by optimality of (A_1^s, B_1^s) we have $\alpha_z + \beta_z \leq \alpha_{x_g} + \beta_{x_f}$.

Let $z \notin B_1^{s+1}$ but $z \in A_1^{s+1}$. If there is some $y \in (A_1^s \setminus B_1^s) \cap B_1^{s+1}$ we have $\beta_z \leq \beta_y$, so we can swap y and z. Otherwise there exist at least two distinct $z_1, z_2 \in B_2^{s+1} \setminus A_1^s$. This implies there is also an $x \in (A_1^s \setminus B_1^s)$ such that $x \notin A_1^{s+1}$. But now again $\alpha_x + \beta_z \leq \alpha_{z_1} + \beta_{z_1}$ so we can swap.

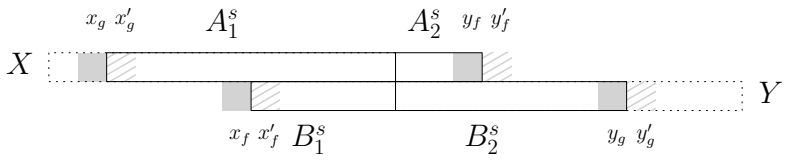


Figure 6.4: Solution changes when increasing/decreasing s.

Combining these two cases we know that $B_1^s \subseteq B_1^{s+1}$. Let $x \in (A_1^s \setminus B_1^s)$. This implies that $B_1^s \subseteq A_1^{s+1}$, so all elements in $X \setminus (A_1^s \setminus B_1^s)$ with cost α smaller than x are contained in A_1^{s+1}. This directly implies that $A_1^s \subseteq A_1^{s+1}$ since $|(A_1^s \setminus B_1^s)| = |(A_1^{s+1} \setminus B_1^{s+1})|$.

The claim for (A_2, B_2) follows by symmetry. □

Lemma 6.8. *There exists an $s \in \{0, 1, \ldots, q\}$ such that for $A = A_1^s \cup A_2^s$ and $B = B_1^s \cup B_2^s$ it holds that $(A, B) = \mathcal{RS}_{p,q}^{\alpha,\beta}(E)$.*

Proof. Let $(A^*, B^*) = \mathcal{RS}_{p,q}^{\alpha,\beta}(E)$ be an optimal solution to the given instance. Define $s := |X \cap B^*|$, $A_1 := A^* \cap X$, $B_1 := B^* \cap X$, $A_2 := A^* \cap Y$ and $B_2 := B^* \cap Y$.

A_1, B_1, A_2, B_2 are feasible solutions to the problems for which $A_1^s, B_1^s, A_2^s, B_2^s$ are optimal solutions. But A_1, B_1, A_2, B_2 are also optimal for these problems, since otherwise (A^*, B^*) could be improved. □

Let us denote by s^* a value of s for which an optimal solution (A^*, B^*) results in the way described above.

Note that a trivial algorithm to determine s^* would not lead to a linear time algorithm. To obtain a fast algorithm overall we need a fast algorithm to find s^*. To that end, we analyze the cost function with respect to the parameter s. For this purpose let

$$f(s) = \sum_{i \in A_1^s \cup A_2^s} \alpha_i + \sum_{j \in B_1^s \cup B_2^s} \beta_j.$$

In the following lemma we show that $f(s)$ is a discrete-convex function.

Lemma 6.9. *For every $s \in \{1, 2, \ldots, q-1\}$ it holds that $2f(s) \leq f(s-1) + f(s+1)$, hence $f(s)$ is discrete-convex.*

Proof. By Lemma 6.7 it holds that both $f(s+1) - f(s) = \alpha_{x_g} + \beta_{x_f} - \alpha_{y_f} - \beta_{y_g}$ and $f(s-1) - f(s) = \alpha_{y'_f} + \beta_{y'_g} - \alpha_{x'_g} - \beta_{x'_f}$. See Figure 6.4 for a visualization. For the corresponding costs we have that $\alpha_{x'_g} + \beta_{x'_f} \leq \alpha_{x_g} + \beta_{x_f}$ and $\alpha_{y_f} + \beta_{y_g} \leq \alpha_{y'_f} + \beta_{y'_g}$.

As a consequence we arrive at

$$f(s+1) - f(s) = \alpha_{x_g} + \beta_{x_f} - \alpha_{y_f} - \beta_{y_g} \geq \alpha_{x'_g} + \beta_{x'_f} - \alpha_{y'_f} - \beta_{y'_g} = f(s) - f(s-1).$$

□

A discrete function f is called unimodal if there is some s' such that for all $s \leq s'$ it holds that $f(s)$ is monotonically decreasing in s and for all $s \geq s'$ it holds that $f(s)$ is

monotonically increasing in s. A plateau of f is a sequence $s_1, s_2, \ldots, s_l$ with $l > 1$ such that $f(s_1) = f(s_2) = \cdots = f(s_l)$.

Using discrete-convexity the following is a well-known consequence.

Corollary 6.10. *It holds that $f(s)$ is unimodal in s for $s \in [q]$, and f has at most one plateau at its minimum.*

Based on Corollary 6.10 it is possible to efficiently check if $s = s^*$, $s^* > s$ or $s^* < s$ for any given s by determining $f(s+1)$ and $f(s-1)$, which results in a fast prune and search for s^* (see Algorithm 6.3), because determining $f(s-1)$ and $f(s+1)$ can be done in the same time as determining $f(s)$ (for details see Section 6.1.3.3). To simplify notation let $\text{mid}(a,b) = \lfloor \frac{a+b}{2} \rfloor$.

Algorithm 6.3: Prune and search for s^*.

```
1 s := 0, s̄ := q
2 while s̄ − s > 3 do
3     Let s := mid(s, s̄)
4     Calculate A_1^{s−1}, B_1^{s−1}, A_2^{s−1}, B_2^{s−1}, f(s − 1);
                A_1^s, B_1^s, A_2^s, B_2^s, f(s);
                A_1^{s+1}, B_1^{s+1}, A_2^{s+1}, B_2^{s+1}, f(s + 1)
                using the subprocedure stated in Section 6.1.3.3.
5     if f(s − 1) = f(s) or f(s) = f(s + 1) or f(s − 1) > f(s) < f(s + 1) then
          s* := s
          return s*, (A_1^s ∪ A_2^s, B_1^s ∪ B_2^s)
6     else if f(s − 1) < f(s) < f(s + 1) then
          s̄ := s − 1
7     else if f(s − 1) > f(s) > f(s + 1) then
          s := s + 1
8 return s minimizing f(s) for s ∈ {s, ..., s̄} with corresponding solution sets
```

To end up with an overall linear running time it is important that in each iteration of this procedure we determine the sets $A_1^s, B_1^s, A_2^s, B_2^s$ in time $O(\Delta s)$, where $\Delta s := \bar{s} - \underline{s}$ is the current size of the search region. In addition we need to make sure that $\Delta s_2 \leq \frac{\Delta s_1}{2}$, where $\Delta s_1, \Delta s_2$ are the sizes of the search region for two consecutive iterations. An overview of the approach to calculate the sets $A_1^s, B_1^s, A_2^s, B_2^s$ efficiently is given in Section 6.1.3.3.

A running time bound of $O(\Delta s)$ is possible only by providing the next iteration of Algorithm 6.3 with preprocessed sets of possible choices, which have to be obtained in addition to calculating $A_1^s, B_1^s, A_2^s, B_2^s$. Details about these preprocessed sets that have to be calculated in addition to $A_1^s, B_1^s, A_2^s, B_2^s$ in each iteration are given in Section 6.1.3.4.

6.1.3.3 Prune and Search for the Number of $(\alpha+\beta)$-Greedy Steps

In this section we explain how to find sets $A_1^s, B_1^s \subseteq X$ in the t-th iteration of Algorithm 6.3 in $O(\Delta s)$ time. The case of finding A_2^s, B_2^s can be handled symmetrically.

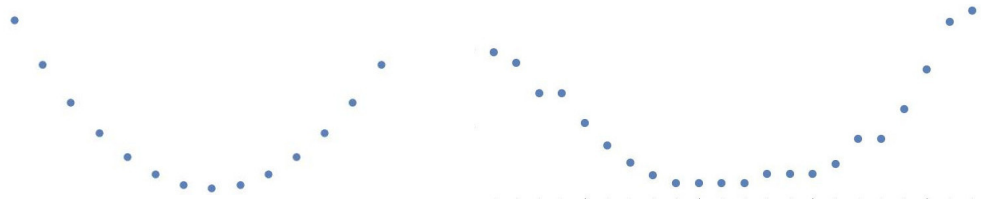


Figure 6.5: Visualization of the functions f (left) and h (right). Note the multiple plateaus in h that complicate the prune and search for r^*.

Again, the main idea of this subprocedure is to introduce a parameter $r \in \{0, 1, \ldots, s\}$, that can be loosely interpreted as the number of $(\alpha+\beta)$-greedy steps performed. Based on r we define the sets

- $G^r = \mathcal{S}_\alpha^{p-q+s-r}(X)$,
- $F^r = \mathcal{S}_\beta^{s-r}(G^r)$,
- $R^r = \mathcal{S}_{\alpha+\beta}^{r}(X \setminus G^r)$, where in the case of ties we agree on the following convention. If there are multiple elements e, f with $\alpha_e + \beta_e = \alpha_{e'} + \beta_{e'}$ we select the one with $\alpha_e < \alpha_{e'}$. If also $\alpha_e = \alpha_{e'}$, then we break ties in the same way as ties are broken for the calculation of G^r.

Observe that $(G^r \cup R^r, F^r \cup R^r)$ is a feasible solution for the problem $\mathcal{RS}_{p,q,s}^{\alpha,\beta,1}(X)$. We denote the cost of this solution by

$$h(r) = \sum_{i \in G^r \cup R^r} \alpha_i + \sum_{j \in F^r \cup R^r} \beta_j.$$

It is easy to see that there is always an optimal solution of the problem $\mathcal{RS}_{p,q,s}^{\alpha,\beta,1}(X)$ of the form $(G^r \cup R^r, F^r \cup R^r)$ for some appropriately chosen r, which we denote by r^*. A visualization of the decomposition of feasible solutions to $\mathcal{RS}_{p,q,s}^{\alpha,\beta,1}(X)$ in terms of the parameter r into subsets G^r, F^r, R^r is shown in Figure 6.6.

To find such a solution efficiently we first analyze properties of the function $h(r)$ in terms of r, similarly as we did for $f(s)$. Below we will show that $h(r)$ is unimodal. Observe however that $h(r)$ in contrast to $f(s)$ is not discrete-convex (see Figure 6.5 for a comparison of the structure of $f(s)$ and $h(r)$). Since in a unimodal function multiple plateaus can appear an efficient algorithm has to use some additional properties. The key result in this direction for $h(r)$ is, that all plateaus happen by the fact that the solutions of the form $(G^r \cup R^r, F^r \cup R^r)$ stay the same for a sequence of values for r (the plateau), except for a single possible exception which is then an optimal solution for $\mathcal{RS}_{p,q,s}^{\alpha,\beta,1}(X)$. This result is formally stated and proved in Lemma 6.15.

The following sequence of lemmas contains formal proves these claims about $h(r)$. Based on those we then state an Algorithm to solve the problem $\mathcal{RS}_{p,q,s}^{\alpha,\beta,1}(X)$.

Proposition 6.11. *It holds for all r that $G^{r+1} \subseteq G^r$, $F^{r+1} \subseteq F^r$ and $R^r \subseteq R^{r+1}$.*

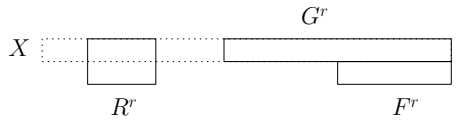


Figure 6.6: Visualization of sets involved in the definition of $h(r)$.

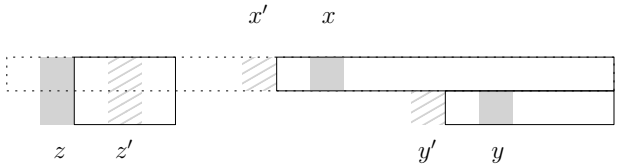


Figure 6.7: The case $z \neq x$ and $x' \neq y'$

Lemma 6.12. *The function $h(r)$ is unimodal in r, i.e. there is an $m \in \mathbb{N}$ such that for all $r \leq m$ it holds that $f(r-1) \geq f(r)$ and for all $r \geq m$ it holds that $f(r) \leq f(r+1)$.*

Proof. Given G^r, F^r and R^r let

- $R^{r-1} = R^r \setminus \{z'\}$, $R^{r+i} = R^{r+i-1} \cup \{z_i\}$,
- $G^{r-1} = G^r \cup \{x'\}$, $G^{r+i} = G^{r+i-1} \setminus \{x_i\}$,
- $F^{r-1} = F^r \cup \{y'\}$, $F^{r+i} = F^{r+i-1} \setminus \{y_i\}$,

for any $i \in \mathbb{N}$. To simplify notation let $x = x_1, y = y_1$ and $z = z_1$. Using this we have that

- $h(r) - h(r-1) = \alpha_{z'} + \beta_{z'} - (\alpha_{x'} + \beta_{y'})$
- $h(r+1) - h(r) = \alpha_z + \beta_z - (\alpha_x + \beta_y)$
- $h(r+i) - h(r+i-1) = \alpha_{z_i} + \beta_{z_i} - (\alpha_{x_i} + \beta_{y_i})$.

First observe that if $z \neq x$ it holds that $\alpha_z + \beta_z \geq \alpha_{z'} + \beta_{z'}$ and if $x' \neq y'$ we have $\beta_y \leq \beta_{y'}$ (see Figure 6.7). This implies that

$$h(r) - h(r-1) \leq h(r+1) - h(r) \iff 2h(r) \leq h(r-1) + h(r+1),$$

i.e. h is locally discrete-convex at r.

Based on this observation we prove the following two claims.

1. If $h(r-1) < h(r)$ it follows that $h(r+1) \geq h(r)$.
2. If $h(r-1) < h(r)$ and $h(r) = \cdots = h(r+d)$ it follows that $h(r+d) \leq h(r+d+1)$

Claim 1 and Claim 2 combined then imply that $h(r)$ is unimodal.

If local convexity holds at r it follows directly from $h(r-1) < h(r)$ that $h(r) < h(r+1)$, hence Claim 1 and Claim 2 are fulfilled. So assume $x = z$ (see Figure 6.8). It holds

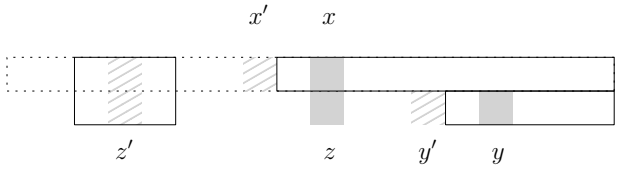


Figure 6.8: The case $z = x$

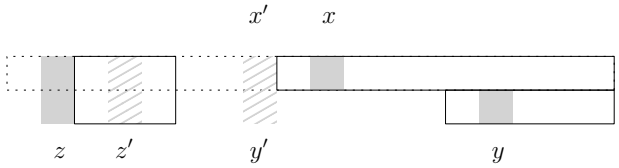


Figure 6.9: The case $x' = y'$

that $\beta_x \geq \beta_y$ so $\alpha_x + \beta_x \geq \alpha_x + \beta_y$. But this implies directly that $h(r+1) \geq h(r)$. The only other case in which discrete-convexity cannot hold locally in r is if $x' = y'$ (see Figure 6.9). In this case we have $\alpha_{z'} + \beta_{z'} \leq \alpha_{x'} + \beta_{x'} = \alpha_{x'} + \beta_{y'}$, which is equivalent to $h(r-1) \geq h(r)$ contradicting our assumption. So we have shown that $h(r) \leq h(r+1)$, i.e. Claim 1 is proved.

For the Claim 2 first observe that $\alpha_{x_i} \leq \alpha_{x'}$ and $\beta_{y_i} \leq \beta_{y'}$ (since $y' \neq x'$) for all i. If $z_i \neq x_i$ it also holds that $\alpha_{z_i} + \beta_{z_i} \geq \alpha_{z'} + \beta_{z'}$, since the only possible new choices for z_i that were no possible choices for z' are x_j for $j = 1, 2, \ldots, i-1$. But for these it holds that they are either no possible choices for z_i, since $z_j = x_j$, or by $z_j \neq x_j$ it follows that $\alpha_{x_j} + \beta_{x_j} \geq \alpha_{z'} + \beta_{z'}$.

If $z_{d+1} \neq x_{d+1}$, we have that

$$\begin{aligned} h(r+d+1) - h(r+d) &= \alpha_{z_{d+1}} + \beta_{z_{d+1}} - (\alpha_{x_{d+1}} + \beta_{y_{d+1}}) \\ &\geq \alpha_{z'} + \beta_{z'} - (\alpha_{x'} + \beta_{y'}) = h(r) - h(r-1) > 0, \end{aligned}$$

implying $h(r+d) < h(r+d+1)$. On the other hand if $z_{d+1} = x_{d+1}$ we have that

$$\alpha_{z_{d+1}} + \beta_{z_{d+1}} = \alpha_{x_{d+1}} + \beta_{x_{d+1}} \geq \alpha_{x_{d+1}} + \beta_{y_{d+1}},$$

directly implying $h(r+d) \leq h(r+d+1)$, hence Claim 2.

This concludes the proof of the lemma. □

Lemma 6.13. *Let $r_1 < r_2$ and $R^{r_i+1} = R^{r_i} \cup \{z_i\}$, $G^{r_i+1} = G^{r_i} \setminus \{x_i\}$, $F^{r_i+1} = F^{r_i} \setminus \{y_i\}$ for $i = 1, 2$. If $x_1 \neq y_1$ and $x_2 \neq z_2$ it holds that*

$$h(r_1 + 1) - h(r_1) \leq h(r_2 + 1) - h(r_2).$$

Proof. We have that $h(r_i+1) - h(r_i) = \alpha_{z_i} + \beta_{z_i} - (\alpha_{x_i} + \beta_{y_i})$. By the fact that $R^{r_1} \subseteq R^{r_2}$ and all z that become available for an $r \in \{r_1 + 1, \ldots, r_2 - 1\}$, it instantly is added to R^{r+1}, we have that $\alpha_{z_1} + \beta_{z_1} \leq \alpha_{z_2} + \beta_{z_2}$.

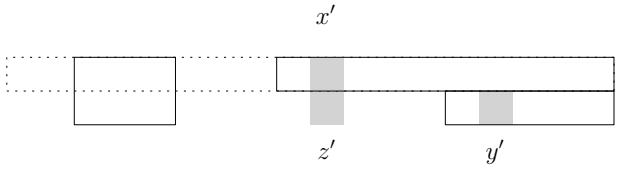


Figure 6.10: The case $x' = z'$

Since $G^{r_2} \subseteq G^{r_1}$ and $F^{r_2} \subseteq F^{r_1}$ we also have that $\alpha_{x_1} \geq \alpha_{x_2}$ and also $\beta_{y_1} \geq \beta_{y_2}$, since y_1 is the largest element in F^{r_1} by the fact that $x_1 \neq y_1$. Combining these facts the result follows.

□

Lemma 6.14. *If it holds that $h(r_1) = h(r_2)$ and the optimal solutions at both r_1 and r_2 are the same, then for each $r \in \{r_1, r_1 + 1, \ldots, r_2\}$ the optimal solution is also the same and hence $h(r) = h(r_1) = h(r_2)$.*

Proof. This follows directly by the fact that $R^r \subseteq R^{r+1}$, since by this fact real changes can never be undone. □

Lemma 6.15. *Given G^r, F^r and R^r let $R^{r+1} = R^r \cup \{z\}$, $G^{r+1} = G^r \setminus \{x\}$ and $F^{r+1} = F^r \setminus \{y\}$. If $x \neq z$ and $h(r) = h(r+1)$ we have that $r = r^*$.*

Proof. If $x = y$ the fact that $h(r) = h(r+1)$ implies $x = z$, since we resolve ties for pairs with respect to the order in which they are chosen by G^r. This is in contrast to our assumption.

We show by induction that for $r' > r$ it holds that $h(r') \geq h(r)$. Let (x', y', z') be the step from $r'-1$ to r'. If at step $r'-1$ we have that $x' \neq z'$, thus, by Lemma 6.13 we have that $h(r'-1) \leq h(r')$ and hence $h(r) \leq h(r')$ by induction. Otherwise if $x' = z'$ (see Figure 6.10) it directly follows that $h(r'-1) \leq h(r')$ since $\alpha_{x'} + \beta_{y'} \leq \alpha_{x'} + \beta_{x'} \leq \alpha_{z'} + \beta_{z'}$. Again by induction we have that $h(r) \leq h(r')$.

Similarly we also show that for $r' < r$ it holds that $h(r') \geq h(r)$ by induction. Again by Lemma 6.13 we have that $h(r'+1) \leq h(r')$ if $x' \neq y'$. Also again if $x' = y'$ we have that $h(r'+1) \leq h(r')$. Hence, by induction it follows that $h(r) \leq h(r')$. □

In the following, we explain how to use the unimodality of h in combination with Lemma 6.15 to obtain an algorithm for $\mathcal{RS}^{\alpha,\beta,1}_{p,q,s}(X)$ based on a prune and search for r^* and the corresponding optimal solutions A^s_1, B^s_1. To simplify the exposition let $\mathcal{H}(r) := (G^r \cup R^r, F^r \cup R^r)$. The basic idea is again similar as in Section 6.1.3.2. The major difference in this case is that long plateaus can appear in $h(r)$. This is why it does not suffice to just calculate the values $h(r-1), h(r), h(r+1)$, since if they are equal we cannot efficiently decide whether $r^* < r, r^* > r$ or $r^* = r$. This is why instead of evaluating at $r-1$ and $r+1$ in addition to the center point $r = \text{mid}(\underline{r}, \bar{r})$ the evaluation is performed at the quarter points $r^{\triangleleft} = \text{mid}(\underline{r}, r), r^{\triangleright} = \text{mid}(r, \bar{r})$ to the left and right. Algorithm 6.4 is a formal description of this approach. Observe that only cases given in steps 2.1–2.8 of Algorithm 6.4 can occur for the evaluation points, since $h(r^{\triangleleft}) < h(r) > h(r^{\triangleright})$

is a contradiction to unimodality of h. The major complication in this binary search are the cases where we have equality among the evaluation points (steps 2.6–2.8 of Algorithm 6.4). Here, we can prune the search space only due to the fact that when the solution $\mathcal{H}$ at the chosen evaluation points is the same we know that the whole range in between forms a plateau and otherwise by Lemma 6.15 the optimum r^* lies between these evaluation points. The way we handle this algorithmically is by setting r to the left endpoint of the plateau, r' to the right endpoint of the plateau and the points $r^{\triangleleft}, r^{\triangleright}$ are then moved further to the left and right, i.e. $r^{\triangleleft} = \text{mid}(\underline{r}, r)$ and $r^{\triangleright} = \text{mid}(r', \bar{r})$. This way we can either conclude optimality by Lemma 6.15 or prune half of the search region, which has size $\Delta r := \bar{r} - \underline{r} - (r' - r)$. Pruning either happens because we know that all the elements in between two evaluation points build a plateau or since the properties of unimodality imply that the minimum cannot lie in some regions.

It is now easy to see that the following holds for Algorithm 6.4.

Lemma 6.16.
- *Let $\Delta r_1, \Delta r_2$ be the sizes of the search region in two consecutive iterations of the loop in step 2 of Algorithm 6.4. Then it holds that $\Delta r_2 \leq \frac{\Delta r_1}{2}$.*
- *Algorithm 6.4 correctly calculates the sets A_1^s, B_1^s.*

To achieve the claimed running time of $O(\Delta s)$, it is not possible to run an algorithm for the SELECTION problem on the whole sets X, G^r and $X \setminus G^r$ during the execution of Algorithm 6.4. In section 6.1.3.4 we explain how to circumvent this issue by showing that certain sets of elements can be fixed and the selection can be performed on smaller sets of candidate elements.

6.1.3.4 Linear Time Implementation: Element Fixing and Candidate Sets

The main objective of this section is to prove that each iteration of the loop in step 2 of Algorithm 6.4 can be executed in $O(\Delta r)$ time. To achieve this already Algorithm 6.3 needs to maintain subsets $\bar{G}, \bar{F}, \bar{R}$ with $|\bar{G}|, |\bar{F}|, |\bar{R}| \leq \Delta s$ and additional sets $\hat{G}, \hat{F}, \hat{R}$ containing already fixed elements in the iterations before the current iteration. These are then given to Algorithm 6.4 such that it can run the classic selection algorithm only on subsets of $\bar{G}, \bar{F}, \bar{R}$ when determining G^r, F^r, R^r. These sets can be obtained in the following way. Algorithm 6.4 is extended to obtain in addition to $G^{r^*}, F^{r^*}, R^{r^*}$ also sets $\bar{G}, \bar{F}, \bar{R}$ containing the Δs next best elements, respectively, if those are still available in the current sets $\bar{G}, \bar{F}, \bar{R}$.

Then, if the decision for the next iteration is to decrease s (step 6 of Algorithm 6.3), we can keep the fixed elements the same and set $\bar{G} := G^{r^*} \setminus \hat{G}$. Based on that we can calculate $\bar{F} := \mathcal{S}_{\beta}^{\Delta s}(\bar{F}_t)$ and $\bar{R} := \mathcal{S}_{\alpha+\beta}^{\Delta s}(\bar{R})$.

Otherwise, if s is increased (step 7 of Algorithm 6.3) we can set $\hat{G} := G^{r^*}, \hat{F} := F^{r^*}$ and $\hat{R} := R^{r^*}$. In addition we can obtain $\bar{G} := \mathcal{S}_{\alpha}^{\Delta s}(\bar{G} \setminus \hat{G})$, $\bar{F} := \mathcal{S}_{\beta}^{\Delta s}(\bar{F} \setminus \hat{F})$ and $\bar{R} := \mathcal{S}_{\alpha+\beta}^{\Delta s}(\bar{R} \setminus \hat{R})$.

These choices of candidate sets give the following result.

Algorithm 6.4: Prune and search for r^*.

```
1   r := 0; r̄ := s; r := mid(r, r̄); r' := mid(r, r̄)
2   while r̄ − r > 3 do
        r◁ := mid(r, r); r▷ := mid(r', r̄)
2.0     Evaluate h(r◁), h(r), h(r▷)
2.1     if h(r◁) > h(r) < h(r▷) then
            r := r◁; r̄ := r▷
2.2     else if h(r◁) > h(r) > h(r▷) then
            r := r; r̄ := r̄; r := mid(r, r̄); r' := mid(r, r̄)
2.3     else if h(r◁) < h(r) < h(r▷) then
            r := r; r̄ := r; r := mid(r, r̄); r' := mid(r, r̄)
2.4     else if h(r◁) = h(r) > h(r▷) then                    // Same as 2.2
            r := r; r̄ := r̄; r := mid(r, r̄); r' := mid(r, r̄)
2.5     else if h(r◁) < h(r) = h(r▷) then                    // Same as 2.3
            r := r; r̄ := r; r := mid(r, r̄); r' := mid(r, r̄)
2.6     else if h(r◁) > h(r) = h(r▷) then
2.6 (a)     if H(r) = H(r▷) then
                r := r◁; r := r; r' := r▷; r̄ := r̄ // Plateau from r to r▷
2.6 (b)     else if H(r) ≠ H(r▷) then
                r := r // We could also set r̄ := r▷
                r := mid(r, r̄); r' := mid(r, r̄)
2.7     else if h(r◁) = h(r) < h(r▷) then
2.7 (a)     if H(r◁) = H(r) then
                r := r; r' := r; r := r◁; r̄ := r▷ // Plateau from r◁ to r
2.7 (b)     else if H(r◁) ≠ H(r) then
                r̄ := r // We could also set r := r◁
                r := mid(r, r̄); r' := mid(r, r̄)
2.8     else if h(r◁) = h(r) = h(r▷) then
2.8 (a)     if H(r◁) = H(r) = H(r▷) then
                r := r◁; r' := r▷ // Plateau from r◁ to r▷
2.8 (b)     else if H(r◁) = H(r) ≠ H(r▷) then
                r = r; r := mid(r, r̄); r' := mid(r, r̄)
                // r◁ to r is a plateau; we could also set r̄ := r▷
2.8 (c)     else if H(r◁) ≠ H(r) = H(r▷) then
                r̄ := r; r := mid(r, r̄); r' := mid(r, r̄)
                // r to r▷ is a plateau; we could also set r := r◁
2.8 (d)     else if H(r◁) ≠ H(r) ≠ H(r▷) then
                // r◁ to r▷ is a plateau and optimal
                return r and H(r)
3   return r and H(r) minimizing h(r) for r ∈ {r, ..., r̄}
```

Lemma 6.17. *After each iteration of Algorithm 6.3 it holds that* $|\bar{G}|, |\bar{F}|, |\bar{R}| \leq \Delta s$*. It also holds that there is an optimal solution containing all the fixed elements and only elements from the candidate sets.*

In the following, we show how these sets can be used to initialize additional sets of candidates and fixed elements that are then used to achieve the claimed running time during the prune and search in Algorithm 6.4. Since the search for r^* always uses search points to the left and right of the center point that are not directly next to it we introduce sets of fixed elements and search sets to the left and right denoted by $\hat{G}^{\triangleleft}, \hat{F}^{\triangleleft}, \hat{R}^{\triangleleft}, \bar{G}^{\triangleleft}, \bar{F}^{\triangleleft}, \bar{R}^{\triangleleft}$ and $\hat{G}^{\triangleright}, \hat{F}^{\triangleright}, \hat{R}^{\triangleright}, \bar{G}^{\triangleright}, \bar{F}^{\triangleright}, \bar{R}^{\triangleright}$, that are used for the evaluation at $r^{\triangleleft}$ and $r^{\triangleright}$. The sets $\hat{G}^{\triangleleft}$, $\hat{F}^{\triangleleft}$, $\hat{R}^{\triangleleft}$, $\bar{G}^{\triangleleft}, \bar{F}^{\triangleleft}, \bar{R}^{\triangleleft}$ are valid fixed elements and candidate sets for the whole range from $\underline{r}$ to r and the sets $\hat{G}^{\triangleright}, \hat{F}^{\triangleright}, \hat{R}^{\triangleright}, \bar{G}^{\triangleright}, \bar{F}^{\triangleright}, \bar{R}^{\triangleright}$ are valid for the whole range from r' to $\bar{r}$. The evaluation at r is only necessary in the first iteration, since in each further iteration of the algorithm r was already an evaluation point in the iteration before. The split in left and right sets helps to handle the possible plateau in the center efficiently.

For the initial evaluation, we have that the search sets $\bar{G}, \bar{F}, \bar{R}$ are of size $\leq \Delta s = \Delta r$, hence the function values can be obtained in the corresponding time at positions $\underline{r}, \bar{r}, r^{\triangleleft}, r^{\triangleright}, r$. At the beginning of each execution of Algorithm 6.4, we initialize $\hat{G}^{\triangleleft} := G^r$, $\hat{F}^{\triangleleft} := F^r$, $\hat{R}^{\triangleleft} := \hat{R}$ and $\bar{G}^{\triangleleft} := \bar{G} \setminus \hat{G}^{\triangleleft}$, $\bar{F}^{\triangleleft} := \bar{F} \setminus \hat{F}^{\triangleleft}$, $\bar{R}^{\triangleleft} := R^r \setminus \hat{R}^{\triangleleft}$. Analogously we set $\hat{G}^{\triangleright} := \hat{G}$, $\hat{F}^{\triangleright} := \hat{F}$, $\hat{R}^{\triangleright} := R^r$ and $\bar{G}^{\triangleright} := G^r \setminus \hat{G}^{\triangleright}$, $\bar{F}^{\triangleright} := F^r \setminus \hat{F}^{\triangleleft}$, $\bar{R}^{\triangleright} := \bar{R} \setminus \hat{R}^{\triangleleft}$.

Based on these sets we can calculate $G^{r^{\triangleleft}}, F^{r^{\triangleleft}}, R^{r^{\triangleleft}}$; $G^{r^{\triangleright}}, F^{r^{\triangleright}}, R^{r^{\triangleright}}$ and their costs in step 2.0 of Algorithm 6.4 in $O(\Delta r)$ time in the following way.

- $\tilde{G}^{r^{\triangleleft}} := \mathcal{S}^{\alpha}_{p-q+s-r^{\triangleleft}-|\hat{G}^{\triangleleft}|}(\bar{G}^{\triangleleft}) \Rightarrow G^{r^{\triangleleft}} = \hat{G}^{\triangleleft} \cup \tilde{G}^{\triangleleft}$
- $\tilde{F}^{r^{\triangleleft}} := \mathcal{S}^{\beta}_{s-r^{\triangleleft}-|\hat{F}^{\triangleleft}|}(\bar{F}^{\triangleleft} \cup \tilde{G}^{r^{\triangleleft}}) \Rightarrow F^{r^{\triangleleft}} = \hat{F}^{\triangleleft} \cup \tilde{F}^{\triangleleft}$
- $\tilde{R}^{r^{\triangleleft}} := \mathcal{S}^{\alpha+\beta}_{r^{\triangleleft}-|\hat{R}^{\triangleleft}|}(\bar{R}^{\triangleleft} \setminus \tilde{G}^{\triangleleft}) \Rightarrow R^{r^{\triangleleft}} = \hat{R}^{\triangleleft} \cup \tilde{R}^{\triangleleft}$
- $\tilde{G}^{r^{\triangleright}} := \mathcal{S}^{\alpha}_{p-q+s-r^{\triangleright}-|\hat{G}^{\triangleright}|}(\bar{G}^{\triangleright}) \Rightarrow G^{r^{\triangleright}} = \hat{G}^{\triangleright} \cup \tilde{G}^{\triangleright}$
- $\tilde{F}^{r^{\triangleright}} := \mathcal{S}^{\beta}_{s-r^{\triangleright}-|\hat{F}^{\triangleright}|}(\bar{F}^{\triangleright} \cup \tilde{G}^{r^{\triangleright}}) \Rightarrow F^{r^{\triangleright}} = \hat{F}^{\triangleright} \cup \tilde{F}^{\triangleright}$
- $\tilde{R}^{r^{\triangleright}} := \mathcal{S}^{\alpha+\beta}_{r^{\triangleright}-|\hat{R}^{\triangleright}|}(\bar{R}^{\triangleright} \setminus \tilde{R}^{r^{\triangleright}}) \Rightarrow R^{r^{\triangleright}} = \hat{R}^{\triangleright} \cup \tilde{R}^{\triangleright}$

Then, after Algorithm 6.4 executes one of the if statements stated in the steps 2.1–2.8, Δr is halved. Hence we have to update the candidate sets to half of their current size. The following list shows the operations to be performed in the end of each of the steps 2.1–2.8 of Algorithm 6.4 to achieve this.

2.1 $\hat{R}^{\triangleleft} := R^{r^{\triangleleft}}$; $\bar{G}^{\triangleleft} := G^{r^{\triangleleft}} \setminus \hat{G}^{\triangleleft}$; $\bar{F}^{\triangleleft} := F^{r^{\triangleleft}} \setminus \hat{F}^{\triangleleft}$; $\bar{R}^{\triangleleft} := \bar{R}^{\triangleleft} \setminus \hat{R}^{\triangleleft}$.
$\hat{G}^{\triangleright} := G^{r^{\triangleright}}$; $\hat{F}^{\triangleright} := F^{r^{\triangleright}}$; $\bar{G}^{\triangleright} := \bar{G}^{\triangleright} \setminus \hat{G}^{\triangleright}$; $\bar{F}^{\triangleright} := \bar{F}^{\triangleright} \setminus \hat{F}^{\triangleright}$; $\bar{R}^{\triangleright} := R^{r^{\triangleright}} \setminus \hat{R}^{\triangleright}$.

2.2 $\hat{R}^{\triangleleft} := R^r$; $\bar{G}^{\triangleleft} := G^r \setminus \hat{G}^{\triangleleft}$; $\bar{F}^{\triangleleft} := F^r \setminus \hat{F}^{\triangleleft}$; $\bar{R}^{\triangleleft} := R^{r^{\triangleright}} \setminus \hat{R}^{\triangleleft}$.
$\hat{R}^{\triangleright} := R^{r^{\triangleright}}$; $\bar{G}^{\triangleright} := G^{r_r} \setminus \hat{G}^{\triangleright}$; $\bar{F}^{\triangleright} := F^{r^{\triangleright}} \setminus \hat{F}^{\triangleright}$; $\bar{R}^{\triangleright} := \bar{R}^{\triangleright} \setminus \hat{R}^{\triangleright}$.

2.3 $\hat{G}^{\triangleleft} := G^{r^{\triangleleft}}$; $\hat{F}^{\triangleleft} := F^{r^{\triangleleft}}$; $\bar{G}^{\triangleleft} := \bar{G}^{\triangleleft} \setminus \hat{G}^{\triangleleft}$; $\bar{F}^{\triangleleft} := \bar{F}^{\triangleleft} \setminus \hat{F}^{\triangleleft}$; $\bar{R}^{\triangleleft} := R^{r^{\triangleleft}} \setminus \hat{R}^{\triangleleft}$.
$\hat{G}^{\triangleright} := G^r$; $\hat{F}^{\triangleright} := F^r$; $\hat{R}^{\triangleright} := R^{r^{\triangleleft}}$; $\bar{G}^{\triangleright} := G^{r^{\triangleleft}} \setminus \hat{G}^{\triangleright}$; $\bar{F}^{\triangleright} := F^{r^{\triangleleft}} \setminus \hat{F}^{\triangleright}$; $\bar{R}^{\triangleright} := R^r \setminus \hat{R}^{\triangleright}$.

2.4 Same as 2.2

2.5 Same as 2.3

2.6 (a) $\hat{R}^{\triangleleft} := R^{r^{\triangleleft}}$; $\bar{G}^{\triangleleft} := G^{r^{\triangleleft}} \setminus \hat{G}^{\triangleleft}$; $\bar{F}^{\triangleleft} := F^{r^{\triangleleft}} \setminus \hat{F}^{\triangleleft}$; $\bar{R}^{\triangleleft} := \bar{R}^{\triangleleft} \setminus \hat{R}^{\triangleleft}$.
$\hat{R}^{\triangleright} := R^{r^{\triangleright}}$; $\bar{G}^{\triangleright} := G^{r_r} \setminus \hat{G}^{\triangleright}$; $\bar{F}^{\triangleright} := F^{r^{\triangleright}} \setminus \hat{F}^{\triangleright}$; $\bar{R}^{\triangleright} := \bar{R}^{\triangleright} \setminus \hat{R}^{\triangleright}$.

(b) Same as 2.2

2.7 (a) $\hat{G}^{\triangleleft} := G^{r^{\triangleleft}}$; $\hat{F}^{\triangleleft} := F^{r^{\triangleleft}}$; $\bar{G}^{\triangleleft} := \bar{G}^{\triangleleft} \setminus \hat{G}^{\triangleleft}$; $\bar{F}^{\triangleleft} := \bar{F}^{\triangleleft} \setminus \hat{F}^{\triangleleft}$; $\bar{R}^{\triangleleft} := R^{r^{\triangleleft}} \setminus \hat{R}^{\triangleleft}$.
$\hat{G}^{\triangleright} := G^{r^{\triangleright}}$; $\hat{F}^{\triangleright} := F^{r^{\triangleright}}$; $\bar{G}^{\triangleright} := \bar{G}^{\triangleright} \setminus \hat{G}^{\triangleright}$; $\bar{F}^{\triangleright} := \bar{F}^{\triangleright} \setminus \hat{F}^{\triangleright}$; $\bar{R}^{\triangleright} := R^{r^{\triangleright}} \setminus \hat{R}^{\triangleright}$.

(b) Same as 2.3

2.8 (a) $\hat{G}^{\triangleleft} := G^{r^{\triangleleft}}$; $\hat{F}^{\triangleleft} := F^{r^{\triangleleft}}$; $\bar{G}^{\triangleleft} := \bar{G}^{\triangleleft} \setminus \hat{G}^{\triangleleft}$; $\bar{F}^{\triangleleft} := \bar{F}^{\triangleleft} \setminus \hat{F}^{\triangleleft}$; $\bar{R}^{\triangleleft} := R^{r^{\triangleleft}} \setminus \hat{R}^{\triangleleft}$.
$\hat{R}^{\triangleright} := R^{r^{\triangleright}}$; $\bar{G}^{\triangleright} := G^{r_r} \setminus \hat{G}^{\triangleright}$; $\bar{F}^{\triangleright} := F^{r^{\triangleright}} \setminus \hat{F}^{\triangleright}$; $\bar{R}^{\triangleright} := \bar{R}^{\triangleright} \setminus \hat{R}^{\triangleright}$.

(b) Same as 2.2

(c) Same as 2.3

(d) We can already terminate with the optimal solution.

Since after that the candidate sets can still contain more elements than necessary (and are too large for our claim), we reduce them by executing

- $\bar{G}^{\triangleleft} := \mathcal{S}^{\alpha}_{\Delta r}(\bar{G}^{\triangleleft})$; $\bar{F}^{\triangleleft} := \mathcal{S}^{\beta}_{\Delta r}(\bar{F}^{\triangleleft})$; $\bar{R}^{\triangleleft} := \mathcal{S}^{\alpha+\beta}_{\Delta r}(\bar{R}^{\triangleleft})$;
- $\bar{G}^{\triangleright} := \mathcal{S}^{\alpha}_{\Delta r}(\bar{G}^{\triangleright})$; $\bar{F}^{\triangleright} := \mathcal{S}^{\beta}_{\Delta r}(\bar{F}^{\triangleright})$; $\bar{R}^{\triangleright} := \mathcal{S}^{\alpha+\beta}_{\Delta r}(\bar{R}^{\triangleright})$.

Lemma 6.18. *This choice of sets of fixed elements and candidate sets is correct.*

Proof. For the fixed elements this follows by the monotonicity with respect to r given by Proposition 6.11.

The correctness of $\bar{G}^{\triangleleft}, \bar{G}^{\triangleright}$ follows also from this monotonicity, since only the "next" Δr best elements have to be considered. The sets $\bar{F}^{\triangleleft}, \bar{F}^{\triangleright}$ the same holds, since when calculating $\tilde{F}^r$ consider in addition also the currently added greedy elements $\tilde{G}^r$. The only additional crux with respect to $\bar{R}^{\triangleleft}, \bar{R}^{\triangleright}$ is that some of the elements in those sets could be chosen by $\tilde{G}^r$. But if $|\tilde{G}^r|$ elements are selected greedy the same amount of less rows are to be selected, so this is not a problem. □

Combining Lemma 6.17, Lemma 6.18 and the convergence of the geometric series to a constant, we obtain our main result.

Theorem 6.19. *The* Recoverable Selection *problem can be solved in $O(n)$ time.*

6.2 Min Cost Matroid Basis with Cardinality Constraints on the Intersection

6.2.1 Introduction

The Model Given two matroids $\mathcal{M}_1 = (E, \mathcal{B}_1)$ and $\mathcal{M}_2 = (E, \mathcal{B}_2)$ on a common ground set E with base sets $\mathcal{B}_1$ and $\mathcal{B}_2$, some integer $k \in \mathbb{N}$, and two cost functions $c_1, c_2 \colon E \to \mathbb{R}$, we consider the optimization problem to find a base $X \in \mathcal{B}_1$ and a base $Y \in \mathcal{B}_2$ minimizing $c_1(X)+c_2(Y)$ subject to either a lower bound constraint $|X \cap Y| \leq k$, an upper bound constraint $|X \cap Y| \geq k$, or an equality constraint $|X \cap Y| = k$ on the size of the intersection of the two bases X and Y. Here, as usual, we write $c_1(X) = \sum_{e \in X} c_1(e)$ and $c_2(Y) = \sum_{e \in Y} c_2(e)$ to shorten notation. Let us denote the following problem by $(P_{=k})$.

$$\begin{aligned} \min \quad & c_1(X) + c_2(Y) \\ \text{s.t.} \quad & X \in \mathcal{B}_1 \\ & Y \in \mathcal{B}_2 \\ & |X \cap Y| = k \end{aligned}$$

Accordingly, if constraint $|X \cap Y| = k$ is replaced by upper bound constraint $|X \cap Y| \leq k$, the problem is called $(P_{\leq k})$, and finally, if constraint $|X \cap Y| = k$ is replaced by the lower bound constraint $|X \cap Y| \geq k$, the problem is called $(P_{\geq k})$. Certainly, it only makes sense to consider integers k in the range between 0 and $K := \min\{\mathrm{rk}(\mathcal{M}_1), \mathrm{rk}(\mathcal{M}_2)\}$, where $\mathrm{rk}(\mathcal{M}_i)$ for $i \in \{1, 2\}$ is the rank of matroid $\mathcal{M}_i$, i.e., the cardinality of each basis in $\mathcal{M}_i$ which is unique. For details on matroids, we refer to [103].

The recoverable robust matroid basis problem – an application of $(P_{\geq})$. The special case of problem $(P_{\geq})$ in which $\mathcal{B}_1 = \mathcal{B}_2$ is known under the name "recoverable robust matroid basis problem (RecRobMatroid)", see e.g. the PhD thesis of Christina Büsing [26]. Büsing presented an algorithm for RecRobMatroid which is exponential in k. In 2017, Hradovich, Kaperski, and Zielinski [72] proved that RecRobMatroid can be solved in polynomial time via some iterative relaxation algorithm and asked for a strongly polynomial time algorithm. Shortly thereafter, the same authors presented in [71] a strongly polynomial time primal dual algorithm for the special case of RecRobMatroid on a graphical matroid. The question whether a strongly polynomial time algorithm for RecRobMatroid on general matroids exists was posed as an open question.

More on RecRobMatroid can be found in Section 6.2.5.

Our contribution. In Section 6.2.2 we show that the problems $(P_{\leq k})$ and $(P_{\geq k})$ can be polynomially reduced to weighted matroid intersection. Since weighted matroid intersection can be solved in strongly polynomial time by some very elegant primal-dual algorithm (cf. Lawler 1970), this answers the open question raised in [72] affirmatively. As we can find min cost matroid basis with lower or upper bound constraints on the cardinality of the intersection of the bases in strongly polynomial time, the question arises

whether or not the problem with equality constraint $(P_{=k})$ can be solved in strongly polynomial time. We manage to provide an affirmative answer for this question too. In Section 6.2.3 we suggest a strongly polynomial time algorithm that constructs an optimal solution for $(P_{=k})$. Finally, in Section 6.2.4, we study the generalizations which result when turning from matroids to polymatroids with lower and upper bound constraints on the size of the meet $|x \wedge y| := \sum_{e \in E} \min\{x_e, y_e\}$. Interestingly, as it turns out, the generalization of $(P_{\geq k})$ can be solved in strongly polynomial time via a reduction to some polymatroidal flow problem, while the generalizations of $(P_{\leq k})$ and $(P_{=k})$ can be shown to be weakly NP-hard, already for uniform polymatroids. The question whether the latter two problems are even strongly NP-hard remains open.

6.2.2 Reduction of $(P_{\leq k})$ and $(P_{\geq k})$ to Weighted Matroid Intersection

We first note that $(P_{\leq k})$ and $(P_{\geq k})$ are computationally equivalent. To see this, consider any instance $(\mathcal{M}_1, \mathcal{M}_2, k, c_1, c_2)$ of $(P_{\geq k})$, where $\mathcal{M}_1 = (E, \mathcal{B}_1)$, and $\mathcal{M}_2 = (E, \mathcal{B}_2)$ are two matroids on the same ground set E with base sets $\mathcal{B}_1$ and $\mathcal{B}_2$, respectively. Define $c_2^* = -c_2$, $k^* = \mathrm{rk}(\mathcal{M}_1) - k$, and let $\mathcal{M}_2^* = (E, \mathcal{B}_2^*)$ with $\mathcal{B}_2^* = \{E \setminus Y \mid Y \in \mathcal{B}_2\}$ be the dual matroid of $\mathcal{M}_2$. Since for $X \in \mathcal{B}_1, Y \in \mathcal{B}_2$ it holds that

(i) $|X \cap Y| \leq k \iff |X \cap (E \setminus Y)| = |X| - |X \cap Y| \geq \mathrm{rk}(\mathcal{M}_1) - k = k^*$, and

(ii) $c_1(X) + c_2(Y) = c_1(X) + c_2(E) - c_2(E \setminus Y) = c_1(X) + c_2^*(E \setminus Y) + c_2(E)$,

where $c_2(E)$ is a constant, it follows that (X, Y) is a minimizer of $(P_{\geq k})$ if and only if $(X, E \setminus Y)$ is a minimizer of $(P_{\leq k^*})$ for the instance $(\mathcal{M}_1, \mathcal{M}_2^*, k^*, c_1, c_2^*)$, and vice versa. Similarly, it can be shown that any problem of type $(P_{\leq k})$ polynomially reduces to an instance of type $(P_{\geq k^*})$.

Theorem 6.20. *$(P_{\leq k})$ and $(P_{\geq k})$ can be reduced to weighted matroid intersection.*

Proof. By our observation above, it suffices to show that $(P_{\leq k})$ can be reduced to weighted matroid intersection. Let $\tilde{E} := E_1 \dot{\cup} E_2$, where E_1, E_2 are two copies of our original ground set E. We consider $\mathcal{N}_1 = (\tilde{E}, \tilde{\mathcal{I}}_1)$ and $\mathcal{N}_2 = (\tilde{E}, \tilde{\mathcal{I}}_2)$, two special types of matroids on this new ground set $\tilde{E}$, where $\mathcal{I}_1, \mathcal{I}_2, \tilde{\mathcal{I}}_1, \tilde{\mathcal{I}}_2$ are the sets of independent sets of $\mathcal{M}_1, \mathcal{M}_2, \mathcal{N}_1, \mathcal{N}_2$ respectively. First, let $\mathcal{N}_1 = (\tilde{E}, \tilde{\mathcal{I}}_1)$ be the direct sum of $\mathcal{M}_1$ on E_1 and $\mathcal{M}_2$ on E_2. That is, for $A \subseteq \tilde{E}$ it holds that $A \in \tilde{\mathcal{I}}_1$ if and only if $A \cap E_1 \in \mathcal{I}_1$ and $A \cap E_2 \in \mathcal{I}_2$.

The second matroid $\mathcal{N}_2 = (\tilde{E}, \tilde{\mathcal{I}}_2)$ is defined as follows: we call $e_1 \in E_1$ and $e_2 \in E_2$ a pair, if e_1 and e_2 are copies of the same element in E. If e_1, e_2 are a pair, then we call e_2 the sibling of e_1 and vice versa. Then

$$\tilde{\mathcal{I}}_2 := \{A \subseteq \tilde{E} \colon A \text{ contains at most } k \text{ pairs}\}.$$

For any $A \subseteq \tilde{E}$, $X = A \cap E_1$ and $Y = A \cap E_2$ forms a feasible solution for $(P_{\leq k})$ if and only if A is a basis in matroid $\mathcal{N}_1$ and independent in matroid $\mathcal{N}_2$. Thus, $(P_{\leq k})$ is equivalent to the weighted matroid intersection problem

$$\max\{w(A) \colon A \in \tilde{\mathcal{I}}_1 \cap \tilde{\mathcal{I}}_2\}$$

with weight function

$$w(e) = \begin{cases} C - c_1(e) & \text{if } e \in E_1, \\ C - c_2(e) & \text{if } e \in E_2 \end{cases}$$

for some constant $C > 0$ chosen large enough to ensure that A is a basis in $\mathcal{N}_1$. To see that $\mathcal{N}_2$ is indeed a matroid, we first observe that $\tilde{\mathcal{I}}_2$ is non-empty and downward-closed (i.e., $A \in \tilde{\mathcal{I}}_2$, and $B \subset A$ implies $B \in \tilde{\mathcal{I}}_2$). To see that $\tilde{\mathcal{I}}_2$ satisfies the matroid-characterizing augmentation property

$$A, B \in \tilde{\mathcal{I}}_2 \text{ with } |A| \leq |B| \text{ implies } \exists e \in B \setminus A \text{ with } A + e \in \tilde{\mathcal{I}}_2,$$

take any two independent sets $A, B \in \tilde{\mathcal{I}}_2$. If A cannot be augmented from B, i.e., if $A + e \notin \tilde{\mathcal{I}}_2$ for every $e \in B \setminus A$, then A must contain exactly k pairs, and for each $e \in B \setminus A$, the sibling of e must be contained in A. This implies $|B| \leq |A|$, i.e., $\mathcal{N}_2$ is a matroid. □

6.2.3 A Strongly Polynomial Primal-Dual Algorithm for $(P_{=k})$

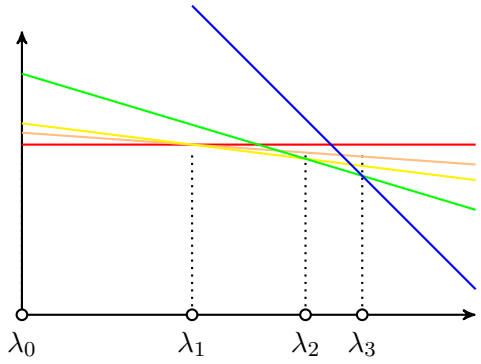
We saw in the previous section that both problems, $(P_{\leq k})$ and $(P_{\geq k})$, can be solved in strongly polynomial time via a weighted matroid intersection algorithm. This leads to the question whether we can solve the problem $(P_{=k})$ with equality constraint on the size of the intersection efficiently as well. At first sight it seems that we could use the same construction we used above to show that $(P_{\leq k})$ can be reduced to matroid intersection, and simply ask whether there exists a solution $A \subseteq \tilde{E}$ which is a basis in both, $\mathcal{N}_1$ and $\mathcal{N}_2$. Note, however, that a feasible solution to $(P_{=k})$ corresponds to a set A which is a basis in $\mathcal{N}_1$ and an independent set in $\mathcal{N}_2$ with exactly k elements, which is not necessarily a basis in $\mathcal{N}_2$. Furthermore, observe that it is not a priori clear whether the problem $(P_{=k})$ can be solved in polynomial time.

In this section, we describe a primal-dual strongly polynomial algorithm for $(P_{=k})$. Our algorithm can be seen as a generalization of the algorithm presented by Hradovich et al. in [72]. However, the analysis of our algorithm turns out to be much simpler than the analysis in [72].

A relaxation of $(P_{=k})$. Let us consider the following piecewise linear concave function

$$\text{val}(\lambda) = \min_{X \in \mathcal{B}_1, Y \in \mathcal{B}_2} c_1(X) + c_2(Y) - \lambda |X \cap Y|$$

which depends on the parameter $\lambda \geq 0$.



Note that $\text{val}(\lambda) + k\lambda$ is the Lagrangian relaxation of problem $(P_{=k})$. Observe that any base pair $(X, Y) \in \mathcal{B}_1 \times \mathcal{B}_2$ determines a line $L_{(X,Y)}(\lambda)$ that hits the y-axis at $c_1(X) + c_2(Y)$ and has negative slope $|X \cap Y|$. Thus, $\text{val}(\lambda)$ is the lower envelope of all such lines. It follows that every base pair $(X, Y) \in \mathcal{B}_1 \times \mathcal{B}_2$ which intersects with $\text{val}(\lambda)$ in either a segment or a breakpoint, and fulfills $|X \cap Y| = k$, is a minimizer of $(P_{=k})$.

Sketch of our algorithm. We first solve

$$\min\{c_1(X) + c_2(Y) \mid X \in \mathcal{B}_1,\ Y \in \mathcal{B}_2\}$$

without any constraint on the intersection, by solving the two independent minimum cost matroid base problems. These problems can be solved with a matroid greedy algorithm in $\mathcal{O}(m \log m + mT)$ time, where $m = |E|$ and T is the time needed to evaluate one call to the independence oracle. Let $(\bar{X}, \bar{Y})$ be an optimal solution of this problem.

1. If $|\bar{X} \cap \bar{Y}| = k$, we are done as $(\bar{X}, \bar{Y})$ is optimal for $(P_{=k})$.

2. If $|\bar{X} \cap \bar{Y}| = k' < k$, our algorithm starts with the optimal solution $(\bar{X}, \bar{Y})$ for $(P_{=k'})$, and iteratively increases k' by one until $k' = k$. Our algorithm maintains as invariant an optimal solution $(\bar{X}, \bar{Y})$ for the current problem $(P_{=k'})$, together with some dual optimal solution $(\bar{\alpha}, \bar{\beta})$ satisfying the optimality conditions, stated in Theorem 6.21 below, for the current breakpoint $\bar{\lambda}$. Details of the algorithm are described below.

3. If $|\bar{X} \cap \bar{Y}| > k$, we instead consider an instance of $(P_{=k^*})$ for $k^* = \text{rk}(\mathcal{M}_1) - k$, costs c_1 and $c_2^* = -c_2$, and the two matroids $\mathcal{M}_1 = (E, \mathcal{B}_1)$ and $\mathcal{M}_2^* = (E, \mathcal{B}_2^*)$. As seen above, an optimal solution $(X, E \setminus Y)$ of problem $(P_{=k^*})$ corresponds to an optimal solution (X, Y) of our original problem $(P_{=k})$, and vice versa. Moreover, $|\bar{X} \cap \bar{Y}| > k$ for the initial base pair $(\bar{X}, \bar{Y})$ implies that $|\bar{X} \cap (E \setminus \bar{Y})| = |\bar{X}| - |\bar{X} \cap \bar{Y}| < k^*$. Thus, starting with the initial feasible solution $(\bar{X}, E \setminus \bar{Y})$ for $(P_{=k^*})$, we can iteratively increase $|\bar{X} \cap (E \setminus \bar{Y})|$ until $|\bar{X} \cap (E \setminus \bar{Y})| = k^*$, as described in step 2.

An optimality condition. The following optimality condition turns out to be crucial for the design of our algorithm.

Theorem 6.21 (Sufficient pair optimality conditions)**.** *For fixed $\lambda \geq 0$, base pair $(X, Y) \in \mathcal{B}_1 \times \mathcal{B}_2$ is a minimizer of* $\mathrm{val}(\lambda)$ *if there exist $\alpha, \beta \in \mathbb{R}_+^{|E|}$ such that*

(i) X is a min cost basis for the cost vector $c_1 - \alpha$, and Y is a min cost basis for the cost vector $c_2 - \beta$;

(ii) $\alpha_e = 0$ for $e \in X \setminus Y$, and $\beta_e = 0$ for $e \in Y \setminus X$;

(iii) $\alpha_e + \beta_e = \lambda$ for each $e \in E$.

Proof. Consider the following linear relaxation of an integer programming formulation of the problem of computing $\mathrm{val}(\lambda)$. The letters in squared brackets indicate the associated dual variables. We denote this problem by (P_λ).

$$
\begin{aligned}
\min \quad & \sum_{e\in E} c_1(e)x_e + \sum_{e\in E} c_2(e)y_e - \lambda \sum_{e\in E} z_e & & \\
\text{s.t.} \quad & \sum_{e\in E} x_e = \mathrm{rk}_1(E) & & [\mu] \\
& \sum_{e\in U} x_e \leq \mathrm{rk}_1(U) & \forall U \subset E \quad & [w_U] \\
& \sum_{e\in E} y_e = \mathrm{rk}_2(E) & & [\nu] \\
& \sum_{e\in U} y_e \leq \mathrm{rk}_2(U) & \forall U \subset E \quad & [v_U] \\
& x_e - z_e \geq 0 & \forall e \in E \quad & [\alpha_e] \\
& y_e - z_e \geq 0 & \forall e \in E \quad & [\beta_e] \\
& x_e, y_e, z_e \geq 0 & \forall e \in E. &
\end{aligned}
$$

The dual program is then (D_λ):

$$
\begin{aligned}
\max \quad & \sum_{U\subset E} \mathrm{rk}_1(U)w_U + \mathrm{rk}_1(E)\mu + \sum_{U\subset E} \mathrm{rk}_2(U)v_U + \mathrm{rk}_2(E)\nu & & \\
\text{s.t.} \quad & \sum_{U\subset E\colon e\in U} w_U + \mu \leq c_1(e) - \alpha_e & & \forall e \in E \\
& \sum_{U\subset E\colon e\in U} v_U + \nu \leq c_2(e) - \beta_e & & \forall e \in E \\
& \alpha_e + \beta_e \geq \lambda & & \forall e \in E \\
& w_U, v_U \leq 0. & & \forall U \subset E \\
& \alpha_e, \beta_e \geq 0. & & \forall e \in E
\end{aligned}
$$

Applying the strong LP-duality theory to the two inner problems which correspond to the dual variables (w, μ) and (v, ν), respectively, yields

$$\max_{\alpha \geq 0} \Big\{ \sum_{U \subset E} \mathrm{rk}_1(U) w_U + \mathrm{rk}_1(E)\mu \mid \sum_{U \subset E:\, e \in U} w_U + \mu \leq c_1(e) - \alpha_e \ \forall e \in E, \\ w_U \leq 0 \ \forall U \subset E \Big\} = \min_{X \in \mathcal{B}_1} c_1(X) - \alpha(X) \quad (6.1)$$

$$\max_{\beta \geq 0} \Big\{ \sum_{U \subset E} \mathrm{rk}_2(U) v_U + \mathrm{rk}_2(E)\nu \mid \sum_{U \subset E:\, e \in U} v_U + \nu \leq c_2(e) - \beta_e \ \forall e \in E, \\ v_U \leq 0 \ \forall U \subset E \Big\} = \min_{Y \in \mathcal{B}_2} c_2(Y) - \beta(Y) \quad (6.2)$$

Thus, replacing the two inner problems by their respective duals, we can rewrite (D_λ) as follows:

$$\begin{aligned} \max \quad & \Big(\min_{X \in \mathcal{B}_1} (c_1(X) - \alpha(X)) + \min_{Y \in \mathcal{B}_2} (c_2(Y) - \beta(Y)) \Big) & \\ \text{s.t.} \quad & \alpha_e + \beta_e \geq \lambda & \forall e \in E \\ & \alpha_e, \beta_e \geq 0. & \forall e \in E \end{aligned}$$

Now, take any tuple (X, Y, α, β) satisfying the optimality conditions (i),(ii) and (iii) for λ. Observe that the incidence vectors x and y of X and Y, respectively, together with the incidence vector z of the intersection $X \cap Y$, constitutes a feasible solution of the primal LP (P_λ), while α and β yield a feasible solution of the dual LP (D_λ). Since

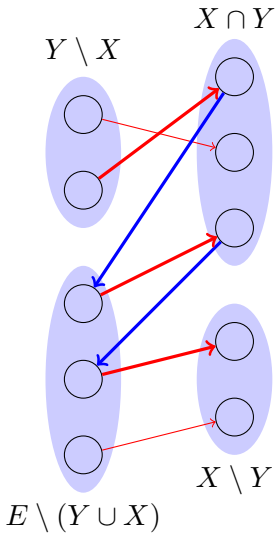
$$c_1(X) - \alpha(X) + c_2(Y) - \beta(Y) = c_1(X) + c_2(Y) - \sum_{e \in X \cap Y} (\alpha_e + \beta_e) = c_1(X) + c_2(Y) - \lambda |X \cap Y|,$$

the objective values of the primal and dual feasible solutions coincide. It follows that any tuple $(X, Y, \alpha, \beta, \lambda)$ satisfying optimality conditions (i),(ii) and (iii) must be optimal for (P_λ) and its dual $D_\lambda)$. Since $X \in \mathcal{B}_1, Y \in \mathcal{B}_2$ it follows that (X, Y) is also a minimizer for $\mathrm{val}(\lambda)$.

□

Construction of the auxiliary digraph. Given a tuple $(X, Y, \alpha, \beta, \lambda)$ satisfying the optimality conditions stated in Theorem 6.21, we construct a digraph $D = D(X, Y, \alpha, \beta)$ with red-blue colored arcs as follows:

- one vertex for each element in E;
- a red arc (e, f) if $e \notin X$, $X - f + e \in \mathcal{B}_1$, and $c_1(e) - \alpha_e = c_1(f) - \alpha_f$; and
- a blue arc (f, g) if $g \notin Y$, $Y - f + g \in \mathcal{B}_2$, and $c_2(g) - \beta_g = c_2(f) - \beta_f$.



Note: Although not depicted in the picture, there might well be blue arcs going from $Y \setminus X$ to either $E \setminus (X \cup Y)$ or $X \setminus Y$, or blue arcs going from $X \cap Y$ to $X \setminus Y$, or red arcs going from $Y \setminus X$ to $X \setminus Y$.

Observe that any red arc (e, f) represents a move from X, to the set $X' = X \cup \{e\} \setminus \{f\}$ which we denote by $X' := X \oplus (e, f)$. Analogously, any blue arc (e, f) represents a move from Y, to the set $Y' = Y \cup \{f\} \setminus \{e\}$, which we also denote by $Y' := Y \oplus (e, f)$. Given a red-blue alternating path P in D we denote by $X' = X \oplus P$ the set obtained from X by performing all moves corresponding to red arcs, and, accordingly, by $Y' = Y \oplus P$ the set obtained from Y by performing all moves corresponding to blue arcs.

Augmenting paths. We call any shortcut-free red-blue alternating path linking a vertex in $Y \setminus X$ to a vertex in $X \setminus Y$ an *augmenting path*. For example, every shortest (w.r.t. number of arcs) red-blue alternating path is shortcut-free.

Lemma 6.22. *If P is an augmenting path in D, then*

- *$X' = X \oplus P$ is a min cost basis in $\mathcal{B}_1$ w.r.t. the costs $c_1 - \alpha$,*
- *$Y' = Y \oplus P$ is a min cost basis in $\mathcal{B}_2$ w.r.t. the costs $c_2 - \beta$, and*
- *$|X' \cap Y'| = |X \cap Y| + 1$.*

Proof. By a well-known Lemma of A. Frank used to prove the correctness of the weighted matroid intersection algorithm [see Lemma 13.35 in Korte & Vygen, p. 373], we know that $X' = X \oplus P$ is a min cost basis in $\mathcal{B}_1$ w.r.t. the costs $c_1 - \alpha$, and $Y' = Y \oplus P$ is a min cost basis in $\mathcal{B}_2$ w.r.t. the costs $c_2 - \beta$. The fact that the cardinality of the intersection is increased by one follows directly from the construction of the digraph. □

Primal update: Given $(X, Y, \alpha, \beta, \lambda)$ satisfying the optimality conditions and the associated digraph D, we update (X, Y) to (X', Y') with $X' = X \oplus P$, and $Y' = Y \oplus P$, as long as some augmenting path P exists in D. It follows by construction and the Lemma above that $(X', Y', \alpha, \beta, \lambda)$ satisfies the optimality conditions and that $|X' \cap Y'| = |X \cap Y| + 1$.

Dual update: If D admits no augmenting path, and $|X \cap Y| < k$, let R denote the set of vertices/elements which are reachable from $Y \setminus X$ on some red-blue alternating path. Note that $Y \setminus X \subseteq R$ and $(X \setminus Y) \cap R = \emptyset$. For each $e \in E$ we define the residual costs

$$\bar{c}_1(e) := c_1(e) - \alpha_e, \quad \text{and} \quad \bar{c}_2(e) := c_2(e) - \beta_e.$$

Note that, by optimality of X and Y w.r.t. $\bar{c}_1$ and $\bar{c}_2$, respectively, we have $\bar{c}_1(e) \geq \bar{c}_1(f)$ whenever $X - f + e \in \mathcal{B}_1$, and $\bar{c}_2(e) \geq \bar{c}_2(f)$ whenever $Y - f + e \in \mathcal{B}_2$.

We compute a "step length" $\delta > 0$ as follows: Compute δ_1 and δ_2 via

$$\delta_1 := \min\{\bar{c}_1(e) - \bar{c}_1(f) \mid e \in R \setminus X,\ f \in X \setminus R \colon X - f + e \in \mathcal{B}_1\},$$

$$\delta_2 := \min\{\bar{c}_2(g) - \bar{c}_2(f) \mid g \notin Y \cup R,\ f \in Y \cap R \colon Y - g + f \in \mathcal{B}_2\}.$$

Note that it is possible that the sets over which the minima are calculated are empty. In these cases we define the corresponding minimum to be ∞. Note that in the special case where $\mathcal{M}_1 = \mathcal{M}_2$ this case cannot occur.

Since neither a red nor a blue arc goes from R to $E \setminus R$, we know that both, δ_1 and δ_2, are strictly positive, so that $\delta := \min\{\delta_1, \delta_2\} > 0$. Now, update

$$\alpha'_e = \begin{cases} \alpha_e + \delta & \text{if } e \in R \\ \alpha_e & \text{else.} \end{cases} \quad \text{and} \quad \beta'_e = \begin{cases} \beta_e & \text{if } e \in R \\ \beta_e + \delta & \text{else.} \end{cases}$$

If $\delta = \infty$ in the min above, the claimed statements hold for arbitrary $\delta > 0$.

Lemma 6.23. *(X, Y, α', β') satisfies the optimality conditions for $\lambda' = \lambda + \delta$.*

Proof. By construction, we have for each $e \in E$

- $\alpha'_e + \beta'_e = \alpha_e + \beta_e + \delta = \lambda + \delta = \lambda'$.
- $\alpha'_e = 0$ for $e \in X \setminus Y$, since $\alpha_e = 0$ and $e \notin R$ (as $(X \setminus Y) \cap R = \emptyset$).
- $\beta'_e = 0$ for $e \in Y \setminus X$, since $\beta_e = 0$ and $(Y \setminus X) \subseteq R$.

Moreover, by construction and choice of δ, we observe that X and Y are optimal for $c_1 - \alpha'$ and $c_2 - \beta'$, since

1. $c_1(e) - \alpha'_e \geq c_1(f) - \alpha'_f$ whenever $X - f + e \in \mathcal{B}_1$,
2. $c_2(g) - \beta'_g \geq c_2(f) - \beta'_f$ whenever $Y - f + g \in \mathcal{B}_2$.

To see claims 1. and 2., suppose for the sake of contradiction that $c_1(e)-\alpha'_e < c_1(f)-\alpha'_f$ for some pair $\{e,f\}$ with $e \notin X$, $f \in X$ and $X - f + e \in \mathcal{B}_1$. Then $e \in R$, $f \notin R$, $\alpha'_e = \alpha_e - \delta$, and $\alpha'_f = \alpha_f$, implying $\delta > c_1(e) - \alpha_e - c_1(f) + \alpha_f = \bar{c}_1(e) - \bar{c}_1(f)$, in contradiction to our choice of δ. Similarly, it can be shown that Y is optimal w.r.t. $c_2 - \beta'$. Thus, (X, Y, α', β') satisfies the optimality conditions for $\lambda' = \lambda + \delta$. □

Lemma 6.24. *If $\delta = \infty$ and $|X \cap Y| < k$ the given instance is infeasible.*

Proof. This follows by the fact that $\delta = \infty$ if and only if the set $(X \setminus Y) \cap R = \emptyset$, even if we construct the graph D' without requiring the condition that for red edges $c_1(e) - \alpha_e = c_1(f) - \alpha_f$ and for blue edges $c_2(g) - \beta_g = c_2(f) - \beta_f$ holds.

The non existence of such a path implies infeasibility of the instance by the classic feasibility conditions for non-weighted matroid intersection. □

Lemma 6.25. *If $(X, Y, \alpha, \beta, \lambda)$ satisfies the optimality conditions and $\delta < \infty$, a primal update can be performed after at most $|E|$ dual updates.*

Proof. With each dual update, at least one more vertex enters the set R' of reachable elements in digraph $D' = D(X, Y, \alpha', \beta')$. □

Algorithm 6.1. Summarizing, we obtain the following algorithm
Input: $\mathcal{M}_1 = (E, \mathcal{B}_1)$, $\mathcal{M}_2 = (E, \mathcal{B}_2)$, $c_1, c_2 : E \to \mathbb{R}$, $k \in \mathbb{N}$
Output: Optimal solution (X, Y) of $(P_{=k})$

1. Compute an optimal solution (X, Y) of $\min\{c_1(X) + c_2(Y) \mid X \in \mathcal{B}_1, Y \in \mathcal{B}_2\}$
2. If $|X \cap Y| = k$, return (X, Y) as optimal solution of $(P_{=k})$
3. Else, if $|X \cap Y| > k$, run Algorithm 6.1 on $\mathcal{M}_1$, $\mathcal{M}_2^*$, c_1, $c_2^* := -c_2$, and $k^* := \mathrm{rk}(\mathcal{M}_1) - k$
4. Else, set $\lambda = 0$, $\alpha = 0$, $\beta = 0$
5. While $|X \cap Y| < k$, do
   - Construct auxiliary digraph D based on $(X, Y, \lambda, \alpha, \beta)$
   - If there exists an augmenting path P in D, update primal
   $$X = X \oplus P, \quad Y = Y \oplus P$$
   - Else, compute step length $\delta > 0$ as follows: Compute δ_1 and δ_2 via
   $$\delta_1 := \min\{\bar{c}_1(e) - \bar{c}_1(f) \mid e \in R \setminus X,\ f \in X \setminus R \colon X - f + e \in \mathcal{B}_1\},$$
   $$\delta_2 := \min\{\bar{c}_2(g) - \bar{c}_2(f) \mid g \notin Y \cup R,\ f \in Y \cap R \colon Y - g + f \in \mathcal{B}_2\}.$$
   Note that it is possible that the sets over which the minima are calculated are empty. In these cases we define the corresponding minimum to be ∞.

$\delta := \min\{\delta_1, \delta_2\}$

If $\delta = \infty$, terminate with the message "infeasible instance"

Else, set $\lambda = \lambda + \delta$ and update dual:

$$\alpha_e = \begin{cases} \alpha_e + \delta & \text{if } e \text{ reachable} \\ \alpha_e & \text{otherwise.} \end{cases} \qquad \beta_e = \begin{cases} \beta_e & \text{if } e \text{ reachable} \\ \beta_e + \delta & \text{otherwise.} \end{cases}$$

- Iterate with $(X, Y, \lambda, \alpha, \beta)$

6. Return (X, Y)

As a consequence of our considerations, the following theorem follows.

Theorem 6.26. *Algorithm 6.1 solves $(P_{=k})$ using at most $k \times |E|$ primal and dual augmentations.*

6.2.4 Recoverable Polymatroid Base Problem

Recall that a function $f : 2^E \to \mathbb{R}$ is called *submodular* if $f(U) + f(V) \geq f(U \cup V) + f(U \cap V)$ for all $U, V \subseteq E$. Function f is called *monotone* if $f(U) \leq f(V)$ for all $U \subseteq V$, and *normalized* if $f(\emptyset) = 0$. Given a submodular, monotone and normalized function f, the pair (E, f) is called a *polymatroid*, and f is called *rank function* of the polymatroid (E, f). The associated *polymatroid base polytope* is defined as:

$$\mathcal{B}(f) := \left\{ x \in \mathbb{R}_+^{|E|} \mid x(U) \leq f(U)\ \forall U \subseteq E,\ x(E) = f(E) \right\},$$

where, as usual, $x(U) := \sum_{e \in U} x_e$ for all $U \subseteq E$. We refer to the book "Submodular Functions and Optimization" by Fujishige [55] for details on polymatroids and polymatroidal flows as referred to below.

Remark. *For the sake of simplicity the arguments below are presented for the case of submodular functions defined on the Boolean lattice $(2^E, \subseteq, \cap, \cup)$. But it can be seen that the arguments presented work also for the more general setting of submodular systems defined on arbitrary distributive lattices, as introduced in the book of Fujishige.*

Polymatroids generalize matroids in the following sense: if the polymatroid rank function f additionally satisfies the unit-increase property

$$f(S \cup \{e\}) \leq f(S) + 1 \quad \forall S \subseteq E,\ e \in E,$$

then the vertices of the associated polymatroid base polytope $\mathcal{B}(f)$ are exactly the incidence vectors of a matroid $(E, \mathcal{B})$ with $\mathcal{B} := \{B \subseteq E \mid f(B) = f(E)\}$. Conversely, the rank function $\mathrm{rk} : 2^E \to \mathbb{R}$ which assigns to every subset $U \subseteq E$ the maximal cardinality $\mathrm{rk}(U)$ of an independent set within U is a polymatroid rank function satisfying the unit-increase property. In particular, bases of a polymatroid base polytope are not necessarily $\{0, 1\}$-vectors anymore. Generalizing the set-theoretic intersection and union operations

from sets (a.k.a. $\{0,1\}$-vectors) to arbitrary vectors can be done via the following binary operations, called meet and join: given two vectors $x, y \in \mathbb{R}^{|E|}$ the meet of x and y is $x \wedge y := (\min\{x_e, y_e\})_{e \in E}$, and the join of x and y is $x \vee y := (\max\{x_e, y_e\})_{e \in E}$. Instead of the size of the intersection, we now talk about the size of the meet, abbreviated by

$$|x \wedge y| := \sum_{e \in E} \min\{x_e, y_e\}.$$

Similarly, the size of the join is $|x \vee y| := \sum_{e \in E} \max\{x_e, y_e\}$. Note that $|x| + |y| = |x \wedge y| + |x \vee y|$. It follows that for any base pair $(x, y) \in \mathcal{B}(f_1) \times \mathcal{B}(f_2)$, we have $|x| = f_1(E) = \sum_{e \in E:\, x_e > y_e} (x_e - y_e) - |x \wedge y|$ and $|y| = f_2(E) = \sum_{e \in E:\, y_e > x_e} (y_e - x_e) - |x \wedge y|$. Therefore, $|x \wedge y| \geq k$ if and only if both, $\sum_{e \in E:\, x_e > y_e} (x_e - y_e) \leq f_1(E) - k$ and $\sum_{e \in E:\, x_e < y_e} (y_e - x_e) \leq f_2(E) - k$. The following problem can be seen as a direct generalization of problem $(P_{\geq k})$ when moving from matroid bases to more general polymatroid base polytopes.

A generalization to polymatroid base polytopes. Let f_1, f_2 be two polymatroid rank functions with associated polymatroid base polytopes $\mathcal{B}(f_1)$ and $\mathcal{B}(f_2)$, let $c_1, c_2 : E \to \mathbb{R}$ be two cost functions on E, and let k be some integer. The following problem, which we denote by $(\hat{P}_{\geq k})$, is a direct generalization of the matroid version $(P_{\geq k})$ to polymatroids. We obtain

$$\begin{aligned} \min \quad & \sum_{e \in E} c_1(e) x(e) + \sum_{e \in E} c_2(e) y(e) \\ \text{s.t.} \quad & x \in \mathcal{B}(f_1) \\ & y \in \mathcal{B}(f_2) \\ & |x \wedge y| \geq k. \end{aligned}$$

In the following, we first show that $(\hat{P}_{\geq k})$ can be reduced to an instance of the *polymatroidal flow problem*, which is known to be computationally equivalent to a submodular flow problem and can thus be solved in strongly polynomial time. Afterwards, we show that the two problems $(\hat{P}_{\leq k})$ and $(\hat{P}_{=} k)$, which can be obtained from $(\hat{P}_{\geq k})$ by replacing the constraint $|x \wedge y| \geq k$ by either $|x \wedge y| \leq k$, or $|x \wedge y| = k$, respectively, are weakly NP-hard.

6.2.4.1 Reduction of Polymatroid Base Problem $(\hat{P}_{\geq k})$ to the Polymatroidal Flow Problem

The polymatroidal flow problem can be described as follows: we are given a digraph $G = (V, A)$, arc costs $\gamma : A \to \mathbb{R}$, lower bounds $l : A \to \mathbb{R}$, and two submodular functions f_v^+ and f_v^- for each vertex $v \in V$. Function f_v^+ is defined on $2^{\delta^+(v)}$, the set of subsets of the set $\delta^+(v)$ of v-leaving arcs, while f_v^- is defined on $2^{\delta^-(v)}$, the set of subsets of the set $\delta^-(v)$ of v-entering arcs. Given a flow $\varphi : A \to \mathbb{R}$, the net-flow at v is abbreviated

by $\partial\varphi(v) := \sum_{a\in\delta^-(v)} \varphi(a) - \sum_{a\in\delta^+(v)} \varphi(a)$. The associated polymatroidal flow problem can now be formulated as follows.

$$\begin{aligned}
\min \quad & \sum_{a\in A} \gamma(a)\varphi(a) & \\
\text{s.t.} \quad & l(a) \le \varphi(a) & (a \in A)\\
& \partial\varphi(v) = 0 & (v \in V)\\
& \varphi|_{\delta^+(v)} \in P(f_v^+) & (v \in V)\\
& \varphi|_{\delta^-(v)} \in P(f_v^-) & (v \in V)
\end{aligned}$$

Here, $P(f)$ means the submodular polyhedron of f defined as

$$P(f) := \{x \in \mathbb{R}^E \colon x(U) \le f(U)\ \forall U \subseteq E\}.$$

As described in Fujishige's book [55, page 127f], the polymatroidal flow problem is computationally equivalent to submodular flow problem and can thus be solved in strongly polynomial time.

Theorem 6.27. *The Recoverable Polymatroid Base problem can be reduced to the Polymatroidal Flow Problem.*

Proof. We create an instance of the Polymatroid Flow problem (see Figure 6.11 for a visualization). The underlying digraph G contains vertices s, u_1, u_2, v_1, v_2, t and for three copies E_X, E_Z, E_Y of the element set E one pair of vertices v_e^1, v_e^2 for all $e \in E_X \cup E_Z \cup E_Y$. The digraph contains an arc $e = (v_e^1, v_e^2)$ between every pair of those arcs and the cost $\gamma(e)$ of those arcs is equal to $c_1(e)$ if $e \in E_X$, equal to $c_2(e)$ if $e \in E_Y$ and equal to $c_1(e) + c_2(e)$ if $e \in E_Z$, where we abuse notation and extend the cost functions on E to the three copies of the elements. All the other arc costs are equal to 0. G contains arcs (u_1, v_e^1) for all $e \in E_X \cup E_Z$, which we call the "red arcs", and arcs (v_e^2, v_2) for all $e \in E_Z \cup E_Y$, which we call the "green arcs". In addition, there exist further arcs (u_2, v_e^1) for all $e \in E_Y$ and (v_e^2, v_1) for all $e \in E_X$. We connect the special vertices with the arcs $(s, u_1), (s, u_2), (v_1, t), (v_2, t), (t, s)$. We set $l((s, u_1)) = f_1(E)$ and $l((v_2, t)) = f_2(E)$ and for all other arcs e we set $l(e) = 0$. In addition we create an upper bound equal to $f_2(E) - k$ on the flow going into u_2 and an upper bound equal to $f_1(E) - k$ on the flow going out of v_1 by introducing constant submodular functions $f_{u_2}^+$ and $f_{v_1}^-$ with the corresponding values, accordingly. In addition the submodular function $\tilde{f}_i$ corresponds to the polymatroid obtained by introducing two copies of e_1, e_2 for each element $e \in E$ and defining

$$\tilde{f}_i(S) := f_i(\{e \in E \colon e_1 \in S \text{ or } e_2 \in S\}).$$

We assign $f_{u_1}^+ := \tilde{f}_1$ and $f_{v_2}^- := \tilde{f}_2$.

We now proof that a minimum cost polymatroidal flow corresponds to a solution to $(\hat{P}_{\ge k})$. Consider the two designated vertices u_1 and v_2 such that $\delta^+(u_1)$ are the red arcs, and $\delta^-(v_2)$ are the green arcs in the figure. Take any feasible polymatroidal flow φ and

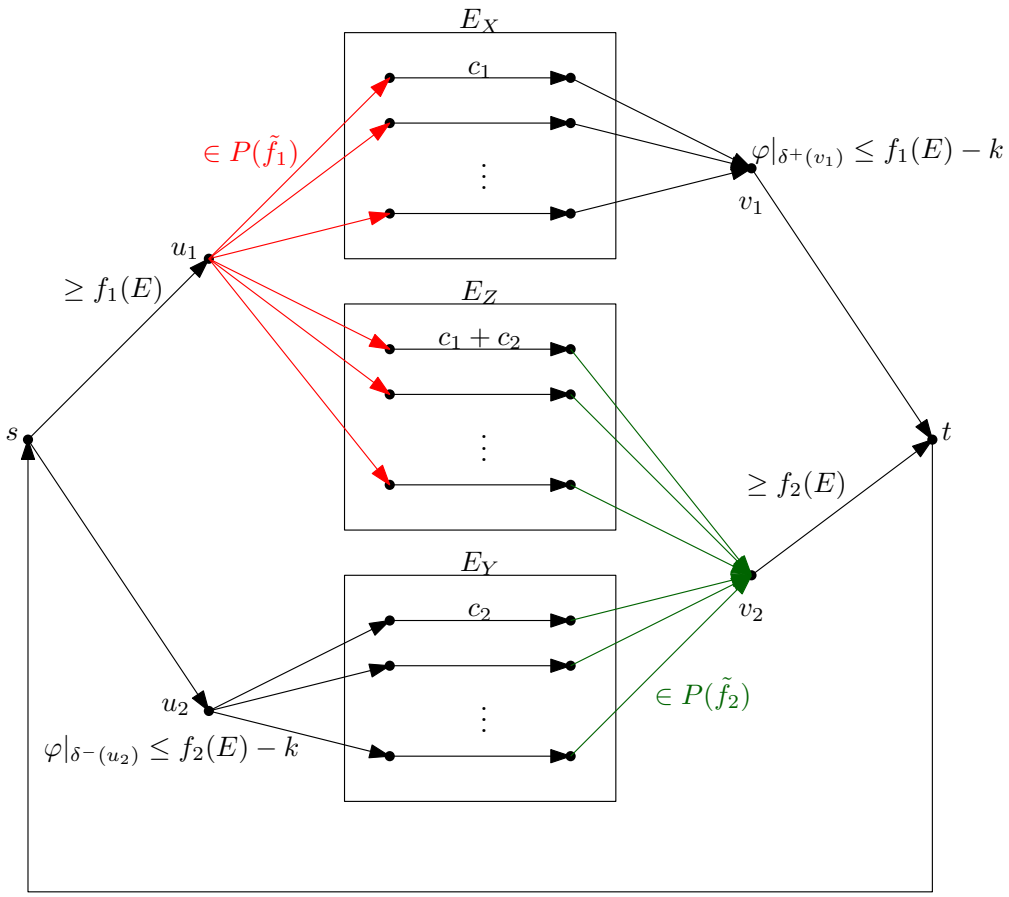


Figure 6.11: The Polymatroid Flow instance used to solve the Recoverable Polymatroid Base Problem.

let $\tilde{x}$ denote the restriction of φ to the red arcs, and $\tilde{y}$ denote the restriction of φ to the green arcs. Note that there is a unique arc entering u_1 which we denote by (s, u_1). Observe that the constraints $\varphi(s, u_1) \geq f_1(E)$, the 0 lower bound on the arcs in E_X, E_Y and $\varphi|_{\delta^+(u_1)} \in P(\tilde{f}_1)$ for the flow going into E_X and E_Z imply that the flow vector $\tilde{x}$ on the red arcs belongs to $\mathcal{B}(\tilde{f}_1)$. Analogously, the flow vector $\tilde{y}$ satisfies $\tilde{y} \in \mathcal{B}(\tilde{f}_2)$. By setting $x(e) := \sum_{e \in X} \tilde{x}_e + \sum_{e \in Z} \tilde{x}_e$ and $y(e) := \sum_{e \in Y} \tilde{y}_e + \sum_{e \in Z} \tilde{y}_e$, we obtain that the cost of the polymatroid flow is given by

$$\sum_{e \in E} c_1(e)x(e) + \sum_{e \in E} c_2(e)y(e).$$

From the constraint $\varphi|_{\delta^-(u_2)} \leq f_2(E) - k$ on the inflow into E_Y, and the constraint $\varphi|_{\delta^+(v_1)} \leq f_1(E) - k$ on the outflow out of E_X or the inflow into v_1, respectively, it follows that $|x \wedge y| \geq k$ holds.

□

Note that $(\hat{P}_{\geq k})$ is computationally equivalent to

$$\begin{aligned} \min \quad & \sum_{e \in E} c_1(e)x(e) + \sum_{e \in E} c_2(e)y(e) \\ \text{s.t.} \quad & x \in \mathcal{B}(f_1) \\ & y \in \mathcal{B}(f_2) \\ & \|x - y\|_1 \leq k' \end{aligned}$$

which we denote by $(\hat{P}_{\|\cdot\|_1})$. This holds true due to the connection $|x| + |y| = 2|x \wedge y| + \|x - y\|_1$ between $|x \wedge y|$, the size of the meet of x and y, and the 1-norm of $x - y$. It is an interesting open question whether the problem $(\hat{P}_{\|\cdot\|_1})$ is also tractable if one replaces $\|x-y\|_1 \leq k'$ by arbitrary norms or specifically the 2-norm. We conjecture that methods based on convex optimization could work in this case, likely leading to a polynomial, but not strongly polynomial, running time.

6.2.4.2 Hardness of the Polymatroid Base Problems $(\hat{P}_{\leq k})$ and $(\hat{P}_{=k})$

Let us consider the decision problem associated to problem $(\hat{P}_{\leq k})$ which can be formulated as follows: given an instance (f_1, f_2, c_1, c_2, k) of $(\hat{P}_{\leq k})$ together with some target value $T \in \mathbb{R}$, decide whether or not there exists a base pair $(x, y) \in \mathcal{B}(f_1) \times \mathcal{B}(f_2)$ with $|x \wedge y| \leq k$ of cost $c_1^T x + c_2^T y \leq T$. Clearly, this decision problem belongs to the complexity class NP, since we can verify in polynomial time whether a given pair (x, y) of vectors satisfies the three conditions (i) $|x \wedge y| \leq k$, (ii) $c_1^T x + c_2^T y \leq T$, and (iii) $(x, y) \in \mathcal{B}(f_1) \times \mathcal{B}(f_2)$. To verify (iii), we assume, as usual, the existence of an evaluation oracle.

Reduction from partition. To show that the problem $(\hat{P}_{\leq k})$ is NP-hard, we show that any instance of the NP-complete problem PARTITION can be polynomially reduced to an instance of $(\hat{P}_{\leq k})$-decision. Recall the problem PARTITION: given a set E of n real numbers $a_1, \ldots, a_n$, the task is to decide whether or not the n numbers can be partitioned into two sets I and $\bar{I}$ with $E = I \cup \bar{I}$ and $I \cap \bar{I} = \emptyset$ such that $\sum_{j \in I} a_j = \sum_{j \in \bar{I}} a_j$.

Now, given an instance $a_1, \ldots, a_n$ of partition with $B := \sum_{j \in E} a_j$, we construct the following polymatroid rank function

$$f(U) = \min\{\sum_{j \in U} a_j, \frac{B}{2}\} \quad \forall U \subseteq E.$$

It is not hard to see that f is indeed a polymatroid rank function as it is normalized, monotone, and submodular. Moreover, we observe that the answer to PARTITION on instance $a_1, \ldots, a_n$ is "yes" if and only if there exists two bases x and y in polymatroid base polytope $\mathcal{B}(f)$ satisfying $|x \wedge y| \leq 0$.

Similarly, it can be shown that any instance of PARTITION can be reduced to an instance of the decision problem associated to $(\hat{P}_{=k})$, since the answer to PARTITION is "yes" if and only if for the polymatroid rank function f as constructed above there exist two bases x and y in the polymatroid base polytope $\mathcal{B}(f)$ satisfying $|x \wedge y| = 0$.

6.2.5 The Recoverable Robust Matroid Base Problem under Uncertainty Degrees

There is a strong connection between the model described in this section and the recoverable robust matroid base problem (RecRobMatroid) mentioned in the introduction. In RecRobMatroid, we are given a matroid $\mathcal{M} = (E, \mathcal{B})$ on a ground set E with base set $\mathcal{B}$, some integer $k \in \mathbb{N}$, a first stage cost function c_1 and an uncertainty set $\mathcal{U}$ that contains different scenarios S, where each scenario $S \in \mathcal{U}$ gives a possible second stage cost $S = (c_S(e))_{e \in E}$.

RecRobMatroid then consists of two stages. In the first stage one needs to pick a base $X \in \mathcal{B}$. Then, the scenario $S \in \mathcal{U}$ is revealed and there is a recovery stage, where one needs to pick a second basis Y that differs in at most k elements from the original basis X, i.e., we require $|X \cap Y| \geq rk(\mathcal{M}) - k$. The goal is to minimize the worst-case cost $c_1(X) + c_S(Y)$. The recoverable robust matroid basis problem can be written as follows:

$$\min_{X \in \mathcal{B}} \left(c_1(X) + \max_{S \in \mathcal{U}} \min_{\substack{Y \in \mathcal{B} \\ |X \cap Y| \geq rk(\mathcal{M}) - k}} c_S(Y) \right) \tag{6.3}$$

There are several ways in which the uncertainty set $\mathcal{U}$ can be represented, and one popular way is the *interval uncertainty representation*. In this representation, we are given a function $d : E \to \mathbb{R}$ and assume that the uncertainty set $\mathcal{U}$ can be represented by a set of $|E|$ intervals:

$$\mathcal{U} = \left\{ S = (c_S(e))_{e \in E} \mid c_S \in [c'(e), c'(e) + d(e)], e \in E \right\}$$

In the worst-case scenario $\bar{S}$ we have that $c_{\bar{S}}(e) = c'(e) + d(e)$ for all $e \in E$. Setting $c_2(e) := c_{\bar{S}}(e)$, it is easy to see that Problem (6.3) is a special case of $(P_{\geq})$, where $\mathcal{B}_1 = \mathcal{B}_2$.

In this section we focus on variations of the interval uncertainty representation. One such popular variation was introduced by Bertsimans and Sim in [12]. This new scenario set, denoted by $\mathcal{U}_1(\Gamma)$, is a subset of $\mathcal{U}$ in which there are at most Γ resources in which $c_S(e) > c'(e)$. Formally:

$$\mathcal{U}_1(\Gamma) = \left\{ S = (c_S(e))_{e\in E} \mid c_S \in [c'(e), c'(e) + \delta_e d(e)], \delta_e \in \{0,1\}, e \in E, \sum_{e\in E} \delta_e \leq \Gamma \right\}.$$

Here, $\Gamma \in \mathbb{N}$ represents the degree of uncertainty.

A second interesting way of defining the scenario set is to impose *budget constraints* on the amount of uncertainty (See e.g., Nasrabadi and Orlin [101]). In this case, an rational uncertainty parameter $\Gamma \in \mathbb{R}_+$ defines the following set of scenarios:

$$\mathcal{U}_2(\Gamma) = \left\{ S = (c_S(e))_{e\in E} \mid c_S \in [c'(e), c'(e) + \delta_e d(e)], \delta_e \in [0,1], e \in E, \sum_{e\in E} \delta_e \leq \Gamma \right\}.$$

These alternative uncertainty sets lead to new variants of our original problem. For $i \in \{1,2\}$ we define:

$$P^i_{\leq k}(\Gamma) := \min_{X\in\mathcal{B}_1} \left(c_1(X) + \max_{S\in\mathcal{U}_i(\Gamma)} \min_{\substack{Y\in\mathcal{B}_2 \\ |X\cap Y|\geq rk(\mathcal{M})-k}} c_S(Y) \right) \tag{6.4}$$

Similarly, we define $P^i_{\geq k}(\Gamma)$ and $P^i_{=k}(\Gamma)$ for $i \in \{1,2\}$. Hradovich, Kasperski and Zieliński [72] show that for any problem for which the recoverable version is polynomially solvable, an approximation algorithm under uncertainty sets $\mathcal{U}_1(\Gamma)$ and $\mathcal{U}_2(\Gamma)$ exists. Let

$$\alpha := \max\{a \in (0,1] \mid c(e) \geq a(c(e) + d(e)) \text{ for all } e \in E\}.$$

The following theorem follows directly from [72, Theorem 5].

Theorem 6.28. *Problems $P^1_{\leq k}(\Gamma)$, $P^1_{\geq k}(\Gamma)$ and $P^1_{=k}(\Gamma)$ under scenario set $\mathcal{U}_1(\Gamma)$ are approximable within $\frac{1}{\alpha}$.*

Moreover, for each instance of $P^2_{\leq k}(\Gamma)$, $P^2_{\geq k}(\Gamma)$ and $P^2_{=k}(\Gamma)$ we define

$$\beta := \max\{b \in (0,1] \mid \Gamma \geq b \sum_{e\in E} d(e)\} \quad \text{and} \quad \gamma := \min\{c \in [0,1) \mid \Gamma \leq cF(\hat{X})\},$$

where $F(\hat{X})$ is the optimal value of the corresponding optimization problem under the original uncertainty set $\mathcal{U}$. Then again, the following theorem follows directly from [72, Theorem 5].

Theorem 6.29. *Problems $P^2_{\leq k}(\Gamma)$, $P^2_{\geq k}(\Gamma)$ and $P^2_{=k}(\Gamma)$ under scenario set $\mathcal{U}_2(\Gamma)$ are approximable within* $\min\{\frac{1}{\alpha}, \frac{1}{\beta}, \frac{1}{1-\gamma}\}$.

6.3 Conclusion and Open Problems

In this chapter we have resolved the computational complexity of the Recoverable Selection problem and obtained the first strongly polynomial time algorithm for the Recoverable Matroid Basis problem. We also generalized the model of recoverable robustness to polymatroids and obtained a strongly polynomial time algorithm for this more general setting.

A major open problem in the are of recoverable robust optimization is whether the recoverable robust spanning tree problem for the uncertainty set $\mathcal{U}_2(\Gamma)$ can be solved in polynomial time.

Besides the mentioned results, there are still many further classic combinatorial optimization problems for which recoverable robust optimization has not yet been studied.

Bibliography

[1] S. Abravaya and M. Segal. Maximizing the number of obnoxious facilities to locate within a bounded region. *Computers and Operations Research*, 37:163–171, 2010.

[2] Heiner Ackermann, Heiko Röglin, and Berthold Vöcking. On the impact of combinatorial structure on congestion games. *Journal of the ACM (JACM)*, 55(6):25:1–25:22, 2008.

[3] Warren P. Adams and Hanif D. Sherali. Mixed-integer bilinear programming problems. *Mathematical Programming*, 59(1):279–305, 1993.

[4] Kim Allemand, Komei Fukuda, Thomas M. Liebling, and Erich Steiner. A polynomial case of unconstrained zero-one quadratic optimization. *Mathematical Programming*, 91(1):49–52, 2001.

[5] Srinivasa Rao Arikati and C. Pandu Rangan. Linear algorithm for optimal path cover problem on interval graphs. *Information Processing Letters*, 35(3):149–153, 1990.

[6] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *Journal of Algorithms*, 12(2):308–340, 1991.

[7] Sanjeev Arora and Boaz Barak. *Computational complexity: a modern approach*. Cambridge University Press, 2009.

[8] Arjang Assad and Wei Xuan Xu. The quadratic minimum spanning tree problem. *Naval Research Logistics*, 39(3):399–417, 1992.

[9] Mokhtar S. Bazaraa and Jamie J. Goode. A cutting-plane algorithm for the quadratic set-covering problem. *Operations Res.*, 23(1):150–158, 1975.

[10] Xavier Berenguer. A characterization of linear admissible transformations for the m-travelling salesmen problem. *European Journal of Operational Research*, 3(3):232–238, 1979.

[11] Attila Bernáth and Zoltán Király. On the tractability of some natural packing, covering and partitioning problems. *Discrete Applied Mathematics*, 180:25–35, 2015.

[12] Dimitris Bertsimas and Melvyn Sim. Robust discrete optimization and network flows. *Mathematical Programming*, 98(1):49–71, Sep 2003.

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

[14] Manuel Blum, Robert W. Floyd, Vaughan Pratt, Ronald L. Rivest, and Robert E. Tarjan. Time bounds for selection. *Journal of computer and system sciences*, 7(4):448–461, 1973.

[15] Fritz Bökler and Petra Mutzel. Output-sensitive algorithms for enumerating the extreme nondominated points of multiobjective combinatorial optimization problems. In *Algorithms-ESA 2015*, pages 288–299. Springer, 2015.

[16] Paul Bonsma. The complexity of the matching-cut problem for planar graphs and other graph classes. *Journal of Graph Theory*, 62(2):109–126, 2009.

[17] A. Bouras. *Problème d'affectation quadratique de petit rang; modèles, compléxite, et applications*. PhD thesis, L'Université Joseph Fourier, Grenoble, France, 1996.

[18] Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[19] Richard A. Brualdi. *Combinatorial Matrix Classes*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 2006.

[20] Richard A. Brualdi and Herbert J. Ryser. *Combinatorial Matrix Theory*. Encyclopedia of Mathematics and its Applications. Cambridge University Press, 1991.

[21] Maurizio Bruglieri, Francesco Maffioli, and Matthias Ehrgott. Cardinality constrained minimum cut problems: complexity and algorithms. *Discrete Applied Mathematics*, 137(3):311–341, 2004.

[22] Tobias Brunsch and Heiko Röglin. Improved smoothed analysis of multiobjective optimization. *Journal of the ACM (JACM)*, 62(1):4:1–4:58, 2015.

[23] Rainer E. Burkard. Efficiently solvable special cases of hard combinatorial optimization problems. *Mathematical programming*, 79(1-3):55–69, 1997.

[24] Rainer E. Burkard. Admissible tranformations and assignment problems. *Vietnam Journal of Mathematics*, 35(4):373–386, 2007.

[25] Rainer E. Burkard, Bettina Klinz, and Rüdiger Rudolf. Perspectives of monge properties in optimization. *Discrete Applied Mathematics*, 70(2):95–161, 1996.

[26] Christina Büsing. *Recoverable robustness in combinatorial optimization*. Cuvillier Verlag, 2011.

[27] Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational mathematics*, 9(6):717, 2009.

[28] P. Cappanera. A survey on obnoxious facility location problems. Technical report, Dipartimento di Informatica, Universitá di Pisa, Italy, 2010.

[29] P.J. Carstensen. Parametric cost shortest path problems. *unpublished Bellcore memo*, 1984.

[30] E. Çela, V.G. Deineko, and G.J. Woeginger. Linearizable special cases of the QAP. *Journal of Combinatorial Optimization*, 31:1269–1279, 2016.

[31] Maw-Shang Chang, Sheng-Lung Peng, and Jenn-Liang Liaw. Deferred-query: An efficient approach for some problems on interval graphs. *Networks*, 34(1):1–10, 1999.

[32] R.L. Church and R.S. Garfinkel. Locating an obnoxious facility on a network. *Transportation Science*, 12:107–118, 1978.

[33] Charles J. Colbourn. The complexity of completing partial latin squares. *Discrete Applied Mathematics*, 8(1):25–30, 1984.

[34] Augustin Cosse and Laurent Demanet. Rank-one matrix completion is solved by the sum-of-squares relaxation of order two. In *Computational Advances in Multi-Sensor Adaptive Processing (CAMSAP), 2015 IEEE 6th International Workshop on*, pages 9–12. IEEE, 2015.

[35] W.R. Cunningham. On submodular function minimization. *Combinatorica*, 5:185–192, 1985.

[36] Ante Ćustić and Bettina Klinz. The constant objective value property for multi-dimensional assignment problems. *Discrete Optimization*, 19:23–35, 2016.

[37] Ante Ćustić and Stefan Lendl. On streaming algorithms for the Steiner cycle and path cover problem on interval graphs and falling platforms in video games, 2018. submitted.

[38] Ante Ćustić and Abraham P. Punnen. Average value of solutions of the bipartite quadratic assignment problem and linkages to domination analysis. *Operations Research Letters*, 45(3):232–237, 2017.

[39] Ante Ćustić and Abraham P. Punnen. A characterization of linearizable instances of the quadratic minimum spanning tree problem. *Journal of Combinatorial Optimization*, 35(2):436–453, 2018.

[40] Ante Ćustić, Vladyslav Sokol, Abraham P Punnen, and Binay Bhattacharya. The bilinear assignment problem: complexity and polynomially solvable special cases. *Mathematical Programming*, 166(1-2):185–205, 2017.

[41] H.N. De Ridder et al. Information system on graph classes and their inclusions (isgci), 2016.

[42] Vladimir Deineko, Eranda Dragoti-Çela, Bettina Klinz, Stefan Lendl, and Gerhard J. Woeginger. Matrix completion problems, 2019. in preparation.

[43] Eranda Dragoti-Çela. *The quadratic assignment problem.* Kluwer Academic Publishers, Dordrecht, 1998.

[44] Abraham Duarte, Manuel Laguna, Rafael Martí, and Jesús Sánchez-Oro. Optimization procedures for the bipartite unconstrained 0-1 quadratic programming problem. *Computers & Operations Research*, 51:123–129, 2014.

[45] J. Edmonds. Paths, trees, and flowers. *Canadian Journal of Mathematics*, 17:449–467, 1965.

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

[47] E. Erkut and S. Neuman. Analytical models for locating undesirable facilities. *European Journal of Operational Research*, 40:275–291, 1989.

[48] Robert W. Floyd and Ronald L. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18(3):165–172, 1975.

[49] Steven Fortune, John Hopcroft, and James Wyllie. The directed subgraph homeomorphism problem. *Theoretical Computer Science*, 10(2):111–121, 1980.

[50] András Frank. *Packing paths, circuits and cuts: a survey.* Forschungsinst. für Diskrete Mathematik, 1988.

[51] András Frank. *Connections in combinatorial optimization*, volume 38. OUP Oxford, 2011.

[52] Andras Frank, Stefan Lendl, Britta Peis, and Veerle Timmermans. Minimum cost matroid bases with cardinality constraints on the intersection, 2019. preprint.

[53] Alexandre S. Freire, Eduardo Moreno, and Juan Pablo Vielma. An integer linear programming approach for bilinear integer programming. *Operations Research Letters*, 40:74–77, 2012.

[54] A. M. Frieze. Complexity of a 3-dimensional assignment problem. *European Journal of Operational Research*, 13(2):161–164, 1983.

[55] Satoru Fujishige. *Submodular functions and optimization*, volume 58. Elsevier, 2005.

[56] Harold N. Gabow and Herbert H. Westermann. Forests, frames, and games: algorithms for matroid sums and applications. *Algorithmica*, 7(1-6):465–497, 1992.

[57] Tomas Gal. *Postoptimal analyses, parametric programming and related topics.* Walter de Gruyter, 1995.

[58] T. Gallai. Kritische Graphen II. *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 8:373–395, 1963.

[59] T. Gallai. Maximale Systeme unabhängiger Kanten. *A Magyar Tudományos Akadémia Matematikai Kutató Intézetének Közleményei*, 9:401–413, 1964.

[60] Joseph L. Ganley, Mordecai J. Golin, and Jeffrey S. Salowe. The multi-weighted spanning tree problem. In *International Computing and Combinatorics Conference*, pages 141–150. Springer, 1995.

[61] M. R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[62] Pawel Gawrychowski, Nadav Krasnopolsky, Shay Mozes, and Oren Weimann. Dispersion on trees. In *25th Annual European Symposium on Algorithms (ESA 2017)*, pages 40:1–40:13, 2017.

[63] Fred Glover, Tao Ye, Abraham P. Punnen, and Gary Kochenberger. Integrating tabu search and VLSN search to develop enhanced algorithms: A case study using bipartite boolean quadratic programs. *European Journal of Operational Research*, 241(3):697–707, 2015.

[64] A.J. Goldman and P.M. Dearing. Concepts of optimal location for partially noxious facilities. *ORSA Bulletin*, 23:B–31, 1975.

[65] Martin Charles Golumbic. Matrix sandwich problems. *Linear algebra and its applications*, 277(1-3):239–251, 1998.

[66] Martin Charles Golumbic, Haim Kaplan, and Ron Shamir. Graph sandwich problems. *Journal of Algorithms*, 19(3):449–473, 1995.

[67] Vineet Goyal, Latife Genc-Kaya, and R Ravi. An fptas for minimizing the product of two non-negative linear cost functions. *Mathematical programming*, 126(2):401–405, 2011.

[68] Alexander Grigoriev, Tim A. Hartmann, Stefan Lendl, and Gerhard J. Woeginger. Dispersing obnoxious facilities on a graph. In *36th International Symposium on Theoretical Aspects of Computer Science, STACS 2019, March 13-16, 2019, Berlin, Germany*, pages 33:1–33:11, 2019.

[69] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric algorithms and combinatorial optimization*. Springer Verlag, 1988.

[70] Daniel Mier Gusfield. *Sensitivity analysis for combinatorial optimization*. PhD thesis, University of California, Berkeley, 1980.

[71] Mikita Hradovich, Adam Kasperski, and Paweł Zieliński. Recoverable robust spanning tree problem under interval uncertainty representations. *Journal of Combinatorial Optimization*, 34(2):554–573, 2017.

[72] Mikita Hradovich, Adam Kasperski, and Paweł Zieliński. The recoverable robust spanning tree problem with interval costs is polynomially solvable. *Optimization Letters*, 11(1):17–30, 2017.

[73] Ruo-Wei Hung and Maw-Shang Chang. Linear-time certifying algorithms for the path cover and hamiltonian cycle problems on interval graphs. *Applied Mathematics Letters*, 24(5):648–652, 2011.

[74] Alon Itai, Christos H Papadimitriou, and Jayme Luiz Szwarcfiter. Hamilton paths in grid graphs. *SIAM Journal on Computing*, 11(4):676–686, 1982.

[75] Gerold Jäger and Paul Molitor. Algorithms and experimental study for the traveling salesman problem of second order. In *Combinatorial optimization and applications*, volume 5165 of *Lecture Notes in Comput. Sci.*, pages 211–224. Springer, Berlin, 2008.

[76] S. Kabadi and A. Punnen. An $O(n^4)$ algorithm for the QAP linearization problem. *Mathemathics of Operations Research*, 36:754–761, 2011.

[77] Daniel Karapetyan, Abraham P. Punnen, and Andrew J. Parkes. Markov chain methods for the bipartite boolean quadratic programming problem. *European Journal of Operational Research*, 260(2):494–506, 2017.

[78] David R. Karger. Enumerating parametric global minimum cuts by random interleaving. In *STOC*, pages 542–555, 2016.

[79] Adam Kasperski and Paweł Zieliński. Robust discrete optimization under discrete and interval uncertainty: A survey. In *Robustness analysis in decision aiding, optimization, and analytics*, pages 113–143. Springer, 2016.

[80] Adam Kasperski and Paweł Zieliński. Robust recoverable and two-stage selection problems. *Discrete Applied Mathematics*, 233:52–64, 2017.

[81] Arman Kaveh. Algorithms and theoretical topics on selected combinatorial optimization problems. Master's thesis, Simon Fraser University, 2010.

[82] J. Mark Keil. Finding hamiltonian circuits in interval graphs. *Information Processing Letters*, 20(4):201–206, 1985.

[83] Bettina Klinz, Rüdiger Rudolf, and Gerhard J. Woeginger. On the recognition of permuted bottleneck monge matrices. *Discrete applied mathematics*, 63(1):43–74, 1995.

[84] Bettina Klinz, Rüdiger Rudolf, and Gerhard J. Woeginger. Permuting matrices to avoid forbidden submatrices. *Discrete applied mathematics*, 60(1-3):223–248, 1995.

[85] Donald E. Knuth. The art of computer programming: Volume 3/sorting and searching, 1973.

[86] Hiroshi Konno. An algorithm for solving bilinear knapsack problem. *Journal of the Operations Research Society of Japan*, 24(4):360–374, 1981.

[87] Bernhard Korte and Clyde L. Monma. Some remarks on a classification of oracle-type-algorithms. In *Numerische Methoden bei graphentheoretischen und kombinatorischen Problemen*, pages 195–215. Springer, 1979.

[88] Bernhard Korte and Jens Vygen. *Combinatorial optimization: theory and algorithms*. Springer, 2018.

[89] Panos Kouvelis and Gang Yu. *Robust discrete optimization and its applications*, volume 14. Springer Science & Business Media, 2013.

[90] Thomas Lachmann and Stefan Lendl. Efficient algorithms for the recoverable (robust) selection problem. *17th Cologne-Twente Workshop on Graphs & Combinatorial Optimization*, 2019. accepted.

[91] Thomas Lachmann, Stefan Lendl, and Gerhard J. Woeginger. Efficient algorithms for the recoverable (robust) selection problem, 2019. in preparation.

[92] Stefan Lendl, Ante Ćustić, and Abraham P. Punnen. Combinatorial optimization problems with interaction costs: Complexity and solvable cases. *Discrete Optimization*, 2019. to appear.

[93] Stefan Lendl, Britta Peis, and Veerle Timmermans. Matroid sum with cardinality constraints on the intersection. *17th Cologne-Twente Workshop on Graphs & Combinatorial Optimization*, 2019. accepted.

[94] Christian Liebchen, Marco Lübbecke, Rolf Möhring, and Sebastian Stiller. The concept of recoverable robustness, linear programming recovery, and railway applications. In *Robust and online large-scale optimization*, pages 1–27. Springer, 2009.

[95] L. Lovász and M.D. Plummer. *Matching Theory*. Annals of Discrete Mathematics 29, North-Holland, Amsterdam, 1986.

[96] László Lovász and Michael D. Plummer. *Matching theory*, volume 367. American Mathematical Soc., 2009.

[97] Glenn K. Manacher, Terrance A. Mankus, and Carol Joan Smith. An optimum $\Theta(n \log n)$ algorithm for finding a canonical hamiltonian path and a canonical hamiltonian circuit in a set of intervals. *Information Processing Letters*, 35(4):205–211, 1990.

[98] N. Megiddo and A. Tamir. New results on the complexity of p-center problems. *SIAM Journal on Computing*, 12:751–758, 1983.

[99] Nimrod Megiddo. Linear-time algorithms for linear programming in $\mathbb{R}^3$ and related problems. *SIAM journal on computing*, 12(4):759–776, 1983.

[100] Shashi Mittal and Andreas S. Schulz. An FPTAS for optimizing a class of low-rank functions over a polytope. *Mathematical Programming*, pages 1–18, 2013.

[101] Ebrahim Nasrabadi and James B. Orlin. Recoverable robust shortest path problems. *CoRR*, 2013.

[102] Jaroslav Opatrny. Total ordering problem. *SIAM Journal on Computing*, 8(1):111–114, 1979.

[103] James G. Oxley. *Matroid theory*, volume 3. Oxford University Press, USA, 2006.

[104] Christos H. Papadimitriou. *Computational complexity*. John Wiley and Sons Ltd., 2003.

[105] René Peeters. Orthogonal representations over finite fields and the chromatic number of graphs. *Combinatorica*, 16(3):417–431, 1996.

[106] A. Punnen and S. Kabadi. A linear time algorithm for the Koopmans-Beckmann QAP linearization and related problems. *Discrete Optimization*, 10:200–209, 2013.

[107] Abraham P. Punnen, Piyashat Sripratak, and Daniel Karapetyan. The bipartite unconstrained 0–1 quadratic programming problem: Polynomially solvable cases. *Discrete Applied Mathematics*, 193:1–10, 2015.

[108] Abraham P. Punnen and Yang Wang. The bipartite quadratic assignment problem and extensions. *European Journal of Operational Research*, 250(3):715–725, 2016.

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

[110] James Roskind and Robert E. Tarjan. A note on finding minimum-cost edge-disjoint spanning trees. *Mathematics of Operations Research*, 10(4):701–708, 1985.

[111] Alexander Schrijver. *Combinatorial optimization: polyhedra and efficiency*, volume 24. Springer Science & Business Media, 2003.

[112] M. Segal. Placing an obnoxious facility in geometric networks. *Nordic Journal of Computing*, 10:224–237, 2003.

[113] Frits C. R. Spieksma. Multi index assignment problems: complexity, approximation, applications. In *Nonlinear assignment problems*, pages 1–12. Kluwer Academic Publishers, Dordrecht, 2000.

[114] Piyashat Sripratak. *The Bipartite Boolean Quadratic Programming Problem*. PhD thesis, Simon Fraser University, 2014.

[115] A. Tamir. Obnoxious facility location on graphs. *SIAM Journal on Discrete Mathematics*, 4:550–567, 1991.

[116] Jens Vygen. Disjoint paths. report no. 94816. *Research Institute for Discrete Mathematics, University of Bonn*, 1994.

[117] Renato Werneck, Joao Setubal, and Arlindo da Conceicao. Finding minimum congestion spanning trees. *Journal of Experimental Algorithmics (JEA)*, 5:11, 2000.

[118] Yasutoshi Yajima and Hiroshi Konno. Outer approximation algorithms for lower rank bilinear programming problems. *Journal of the Operations Research Society of Japan*, 38(2):230–239, 1995.

[119] Takayuki Yato and Takahiro Seta. Complexity and completeness of finding another solution and its application to puzzles. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 86(5):1052–1060, 2003.